The Weblate Manual

发行版本 *4.16.2*

Michal Čihař

2023 年 03 月 08 日

用户文档

CHAPTER 1

用户文档

1.1 Weblate 基础知识

1.1.1 项目和部件架构

在 Weblate 中，翻译被组织成项目和部件。每个项目可以包含多个部件，这些部件又包含各个语言的翻译。部件对应的是一个可翻译的文件（例如*GNU gettext* 或*Android 字符串资源*）。项目可以帮助您将部件组织成逻辑集合（例如，将一个应用中用到的所有翻译集合在一起）。

默认情况下，每个项目内都有对跨部件传播的公共字符串的翻译。这减轻了重复和多版本翻译的负担。但假如翻译应当有所不同，可以使用允许同步翻译 通过部件配置 禁用翻译传播。

参见：

../devel/integration

1.2 注册和用户个人资料

1.2.1 注册

默认情况下，每个人都可以浏览项目、查看翻译或建议翻译。只有注册用户才能实际保存更改，并且每次翻译都会获得积分。

您可以按照以下几个简单步骤进行注册：

1. 使用您的凭据填写注册表。
2. 使用电子邮件中收到的链接来激活注册账号。
3. 可以选择调整您的个人资料以选择您知道的语言。

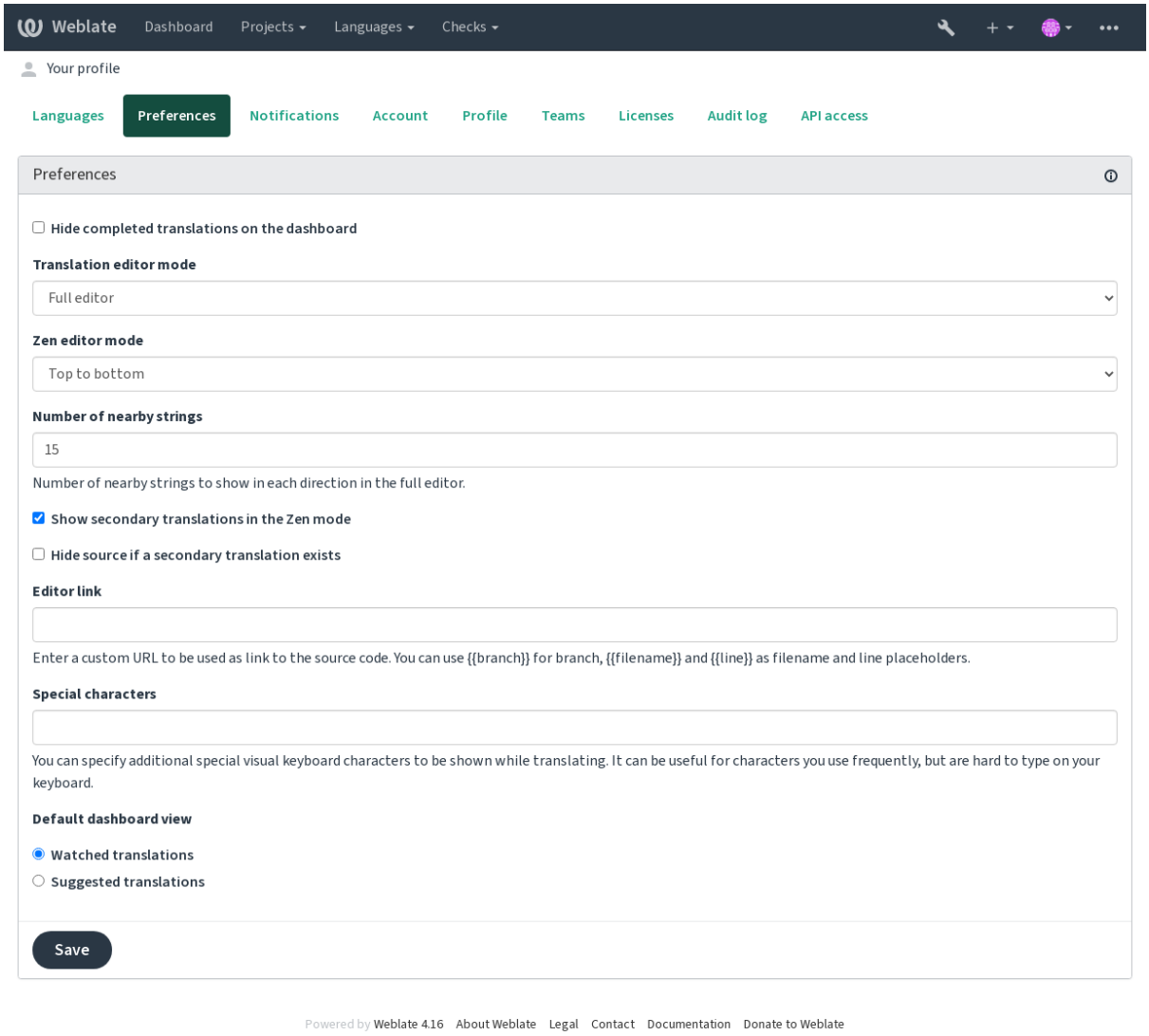
1.2.2 操作面板

登录后，您将看到项目和组件的概述，以及它们各自的翻译进度。

在 2.5 版本加入.

默认情况下，将显示您正在监视的项目的组件，并与您的首选语言交叉引用。

提示： 您可以使用导航选项卡切换到不同的视图。



菜单有以下选项：

- 项目 > 浏览所有项目在主菜单中显示 Weblate 实例上每个项目的翻译状态。
- 在 语言 菜单中选择一种语言将显示所有项目的翻译状态，按您的一种主要语言过滤。
- 观察的翻译在操作面板中将仅显示您正在观看的项目的翻译状态，按您的主要语言过滤。

此外，下拉菜单还可以显示任意数量的 * 部件列表 *，由 Weblate 管理员预先配置的项目组件集，请参阅部件列表。

你可以在你的用户个人资料的 首选项部分配置你的个人操作面板默认视图。

备注： 当使用 `settings.py` 文件中的*SINGLE_PROJECT* 为单个项目配置 Weblate 时（请参阅配置），

操作面板将不会显示，因为用户将被重定向到一个项目或部件。

1.2.3 用户个人资料

通过单击顶部菜单右上角的用户图标，然后单击 设置菜单，可以访问用户配置文件。

用户配置文件包含您的首选项。名称和电子邮箱地址在版本控制系统（VCS）提交中使用，因此请保持此信息准确无误。

备注： 所有语言选择仅提供当前翻译的语言。

提示： 通过单击按钮请求或添加要翻译的其他语言，以使其也可用。

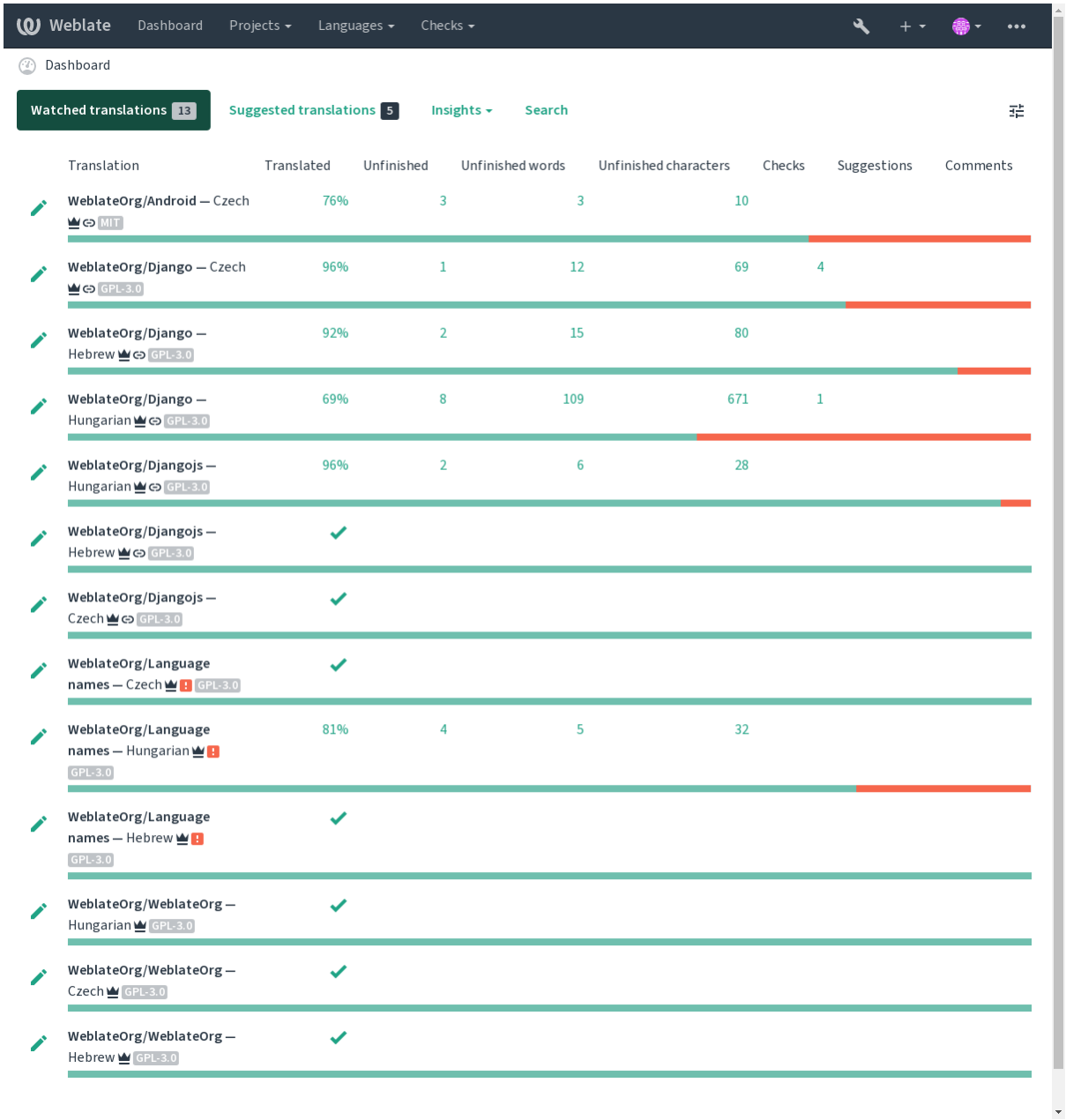
语言

1.2.4 界面语言

请选择你想要在用户界面中展示的语言。

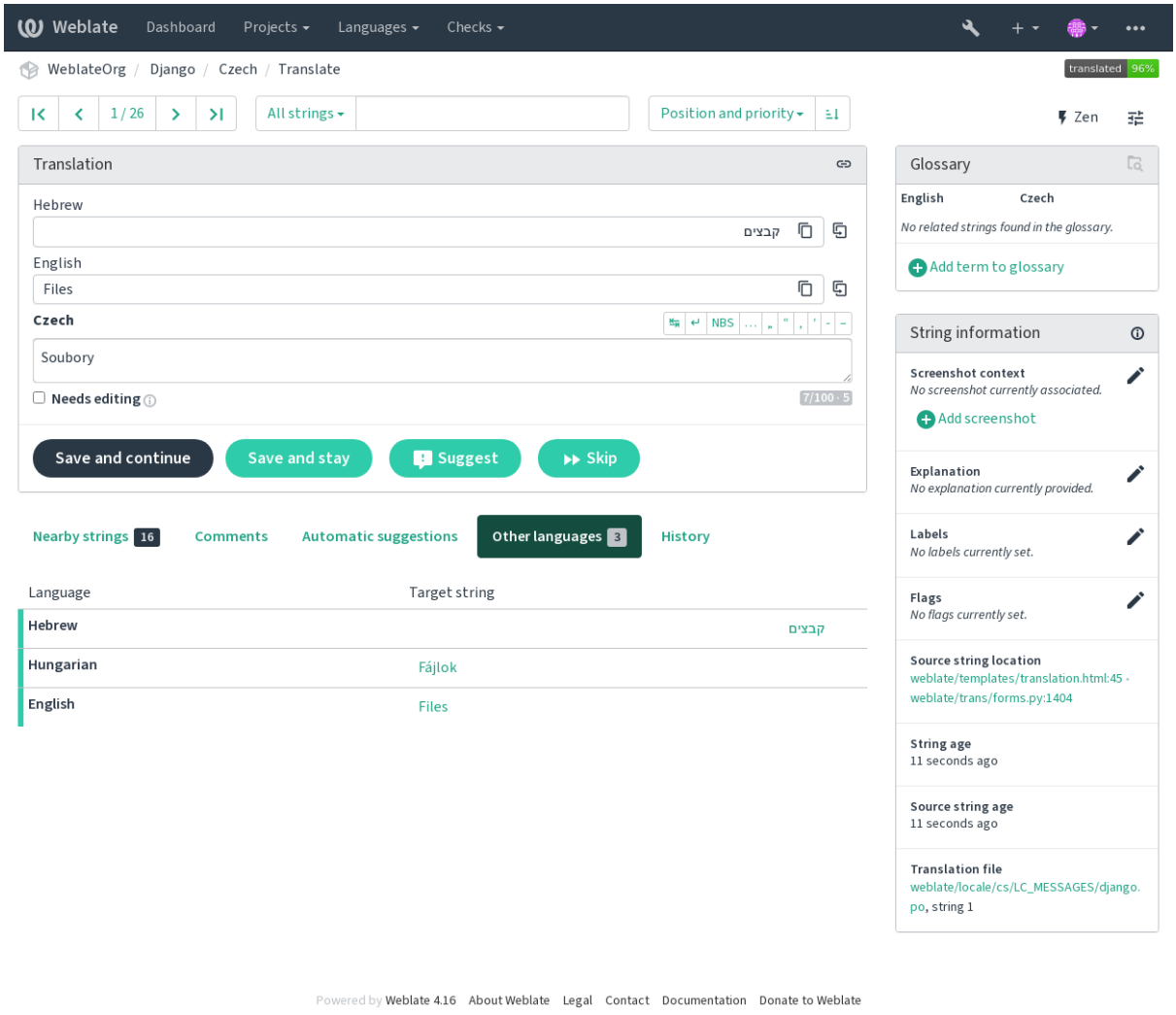
翻译语言

选择您喜欢翻译的语言，它们将在观看项目的主页上提供，以便您可以更轻松地访问每种语言的所有翻译。



第二语言

您可以定义在翻译时向您显示哪些辅助语言作为指南。在下图中可以看到一个示例，其中希伯来语显示为次要语言：



1.2.5 首选项

操作面板默认视图

在 首选项选项卡，你可以选择默认显示的操作面板视图。如果你选择 部件列表，你必须从 默认部件列表下拉菜单中选择要显示哪个部件列表。

参见：

部件列表

编辑器链接

默认情况下，源代码链接显示在部件配置 中配置的 Web 浏览器中。

提示： 通过设置 编辑器链接，您可以使用本地编辑器打开翻译字符串的版本控制系统（VCS）源代码文件。你可以使用模板标记。

通常像 `editor://open/?file={{filename}}&line={{line}}` 是一个不错的选择。

参见：

您可以在 Nette 文档 中找到有关为编辑器注册自定义 URL 协议的更多信息。

特殊字符

虚拟键盘 中要包括的其他特殊字符。

1.2.6 通知

从 通知选项卡订阅各种通知。有关已观看或管理的项目的选定事件的通知将通过电子邮件发送给您。

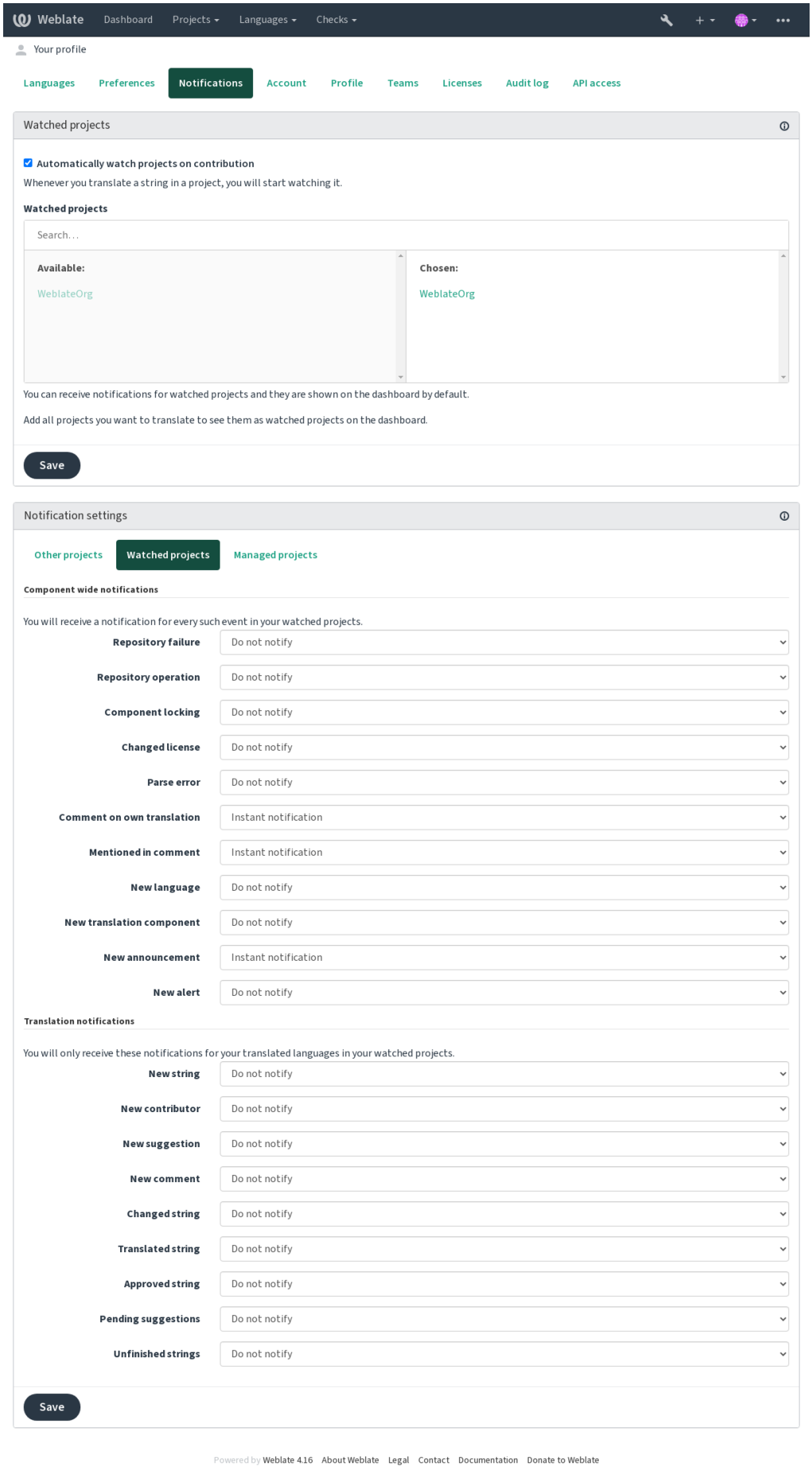
某些通知仅针对您语言的事件发送（例如，关于要翻译的新字符串），而某些通知在组件级别触发（例如合并错误）。这两组通知在设置中在视觉上是分开的。

您可以切换监视项目和管理项目的通知，并且可以进一步调整（或静音）每个项目和部件。访问组件概览页面并从 已关注菜单中选择适当的选项。

如果启用了 自动关注作出贡献的项目，您将在翻译字符串时自动开始关注项目。默认值取决于*DEFAULT_AUTO_WATCH*。

备注： 您不会收到有关您自己操作的通知。

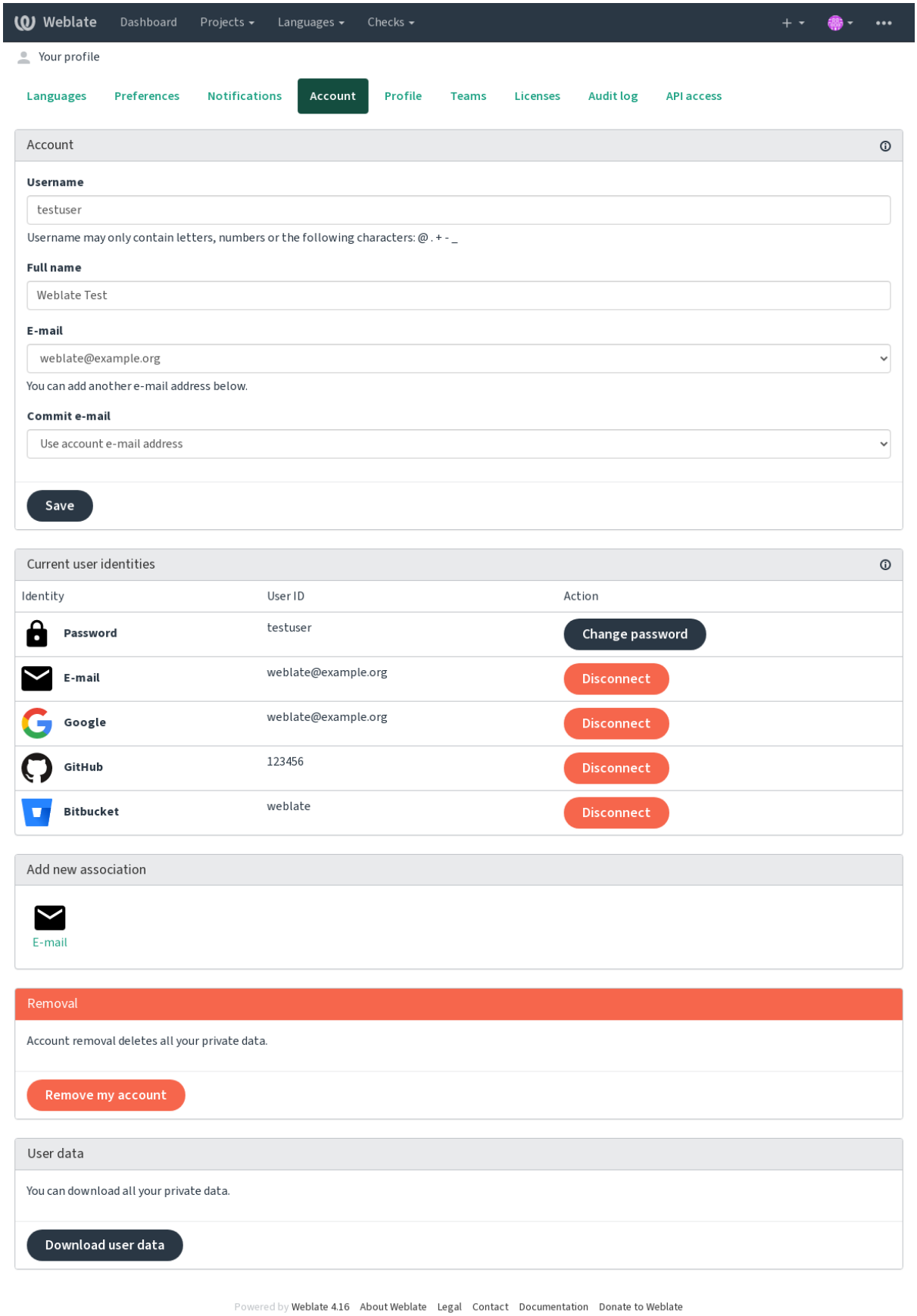
提示： 发送通知的数量是有限制的，你每天收到的电子邮件不会超过 1000 封。第 1001 封及之后发给你的邮件通知都将被丢弃。


Weblate
Dashboard
Projects
Languages
Checks
Your profile
Languages
Preferences
Notifications
Account
Profile
Teams
Licenses
Audit log
API access
Watched projects
Automatically watch projects on contribution
Whenever you translate a string in a project, you will start watching it.
Watched projects
Search...
Available:
WeblateOrg
Chosen:
WeblateOrg
You can receive notifications for watched projects and they are shown on the dashboard by default.
Add all projects you want to translate to see them as watched projects on the dashboard.
Save
Notification settings
Other projects
Watched projects
Managed projects
Component wide notifications
You will receive a notification for every such event in your watched projects.
Repository failure
Repository operation
Component locking
Changed license
Parse error
Comment on own translation
Mentioned in comment
New language
New translation component
New announcement
New alert
Do not notify
Instant notification
Translation notifications
You will only receive these notifications for your translated languages in your watched projects.
New string
New contributor
New suggestion
New comment
Changed string
Translated string
Approved string
Pending suggestions
Unfinished strings
Save
Powered by Weblate 4.16
About Weblate
Legal
Contact
Documentation
Donate to Weblate

1.2.7 账户

账户选项卡可让您设置基本账户详细信息、连接可用于登录 Weblate 的各种服务、完全删除您的账户或下载您的用户数据（请参阅*Weblate 用户数据导出*）。

备注： 服务列表取决于您的 Weblate 配置，但可以设置为包括 GitLab、GitHub、Google、Facebook 或 Bitbucket 或其他 OAuth 2.0 提供商等热门站点。


Weblate
Dashboard
Projects
Languages
Checks
Your profile
Languages
Preferences
Notifications
Account
Profile
Teams
Licenses
Audit log
API access
Account
Username
testuser
Username may only contain letters, numbers or the following characters: @ . + - _
Full name
Weblate Test
E-mail
weblate@example.org
You can add another e-mail address below.
Commit e-mail
Use account e-mail address
Save
Current user identities
Identity
User ID
Action
Password
testuser
Change password
E-mail
weblate@example.org
Disconnect
Google
weblate@example.org
Disconnect
GitHub
123456
Disconnect
Bitbucket
weblate
Disconnect
Add new association
E-mail
Removal
Account removal deletes all your private data.
Remove my account
User data
You can download all your private data.
Download user data
Powered by Weblate 4.16
About Weblate
Legal
Contact
Documentation
Donate to Weblate

1.2.8 个人资料

本页的所有字段都是选填的，可以随时删除。填写这些字段即表示您同意我们在您的用户个人资料出现的任何地方共享此数据。

在版本控制提交中，将使用代码提交电子邮箱而不是您的账户电子邮箱。使用这个可以避免泄露你的真实电子邮箱。请注意，使用不同的电子邮箱可能会断开你在其他服务器上的做出的贡献（例如您的贡献将不再链接到你的 Github 个人资料）。可使用*PRIVATE_COMMIT_EMAIL_OPT_IN* 在全站范围内开启私密电子邮箱。

可以为每个用户显示头像（取决于*ENABLE_AVATARS*）。这些图像是使用 https://gravatar.com/ 获得的。

1.2.9 许可协议

1.2.10 API 访问

你可以在这里获取或者重置你的 API 访问令牌。

1.2.11 审计日志

审核日志跟踪使用您的账户执行的操作。它会记录您账户中每个重要操作的 IP 地址和浏览器。关键操作还会触发对主电子邮箱地址的通知。

参见:

在反向代理之后运行

1.3 使用 Weblate 进行翻译工作

感谢您对使用 Weblate 进行翻译的兴趣。可以将项目设置为直接翻译，也可以通过接受没有账户的用户提出的建议来设置项目。

总体而言，有两种翻译模式：

- 该项目接受直接翻译
- 该项目只接受建议，一旦达到规定的投票数，这些建议就会自动验证

有关翻译工作流的更多信息，请见翻译工作流。

翻译项目可见性选项：

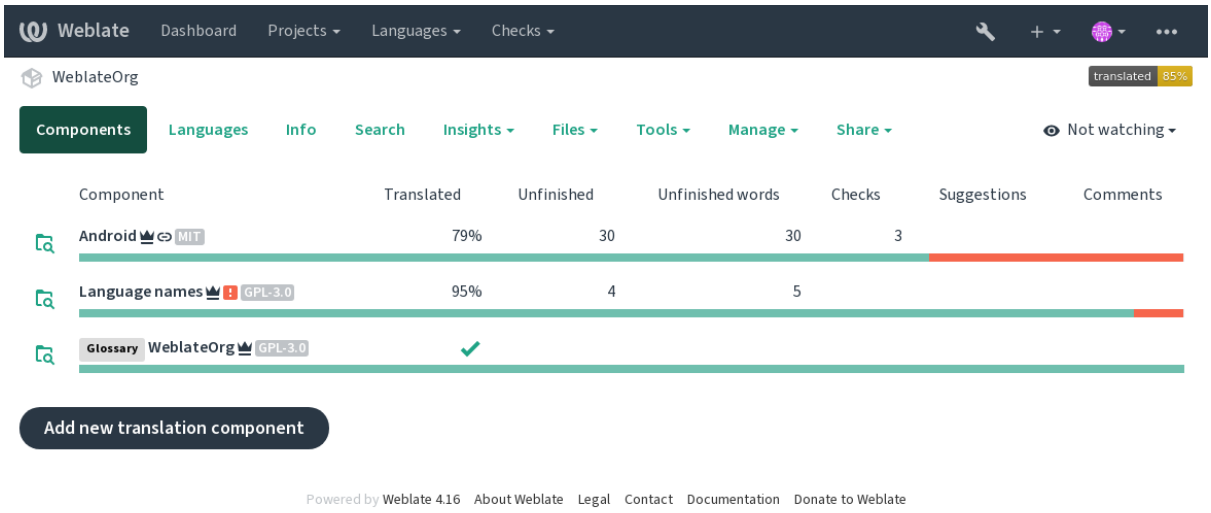
- 公开可见
- 仅指定的译者群组可见

参见:

访问控制, 翻译工作流

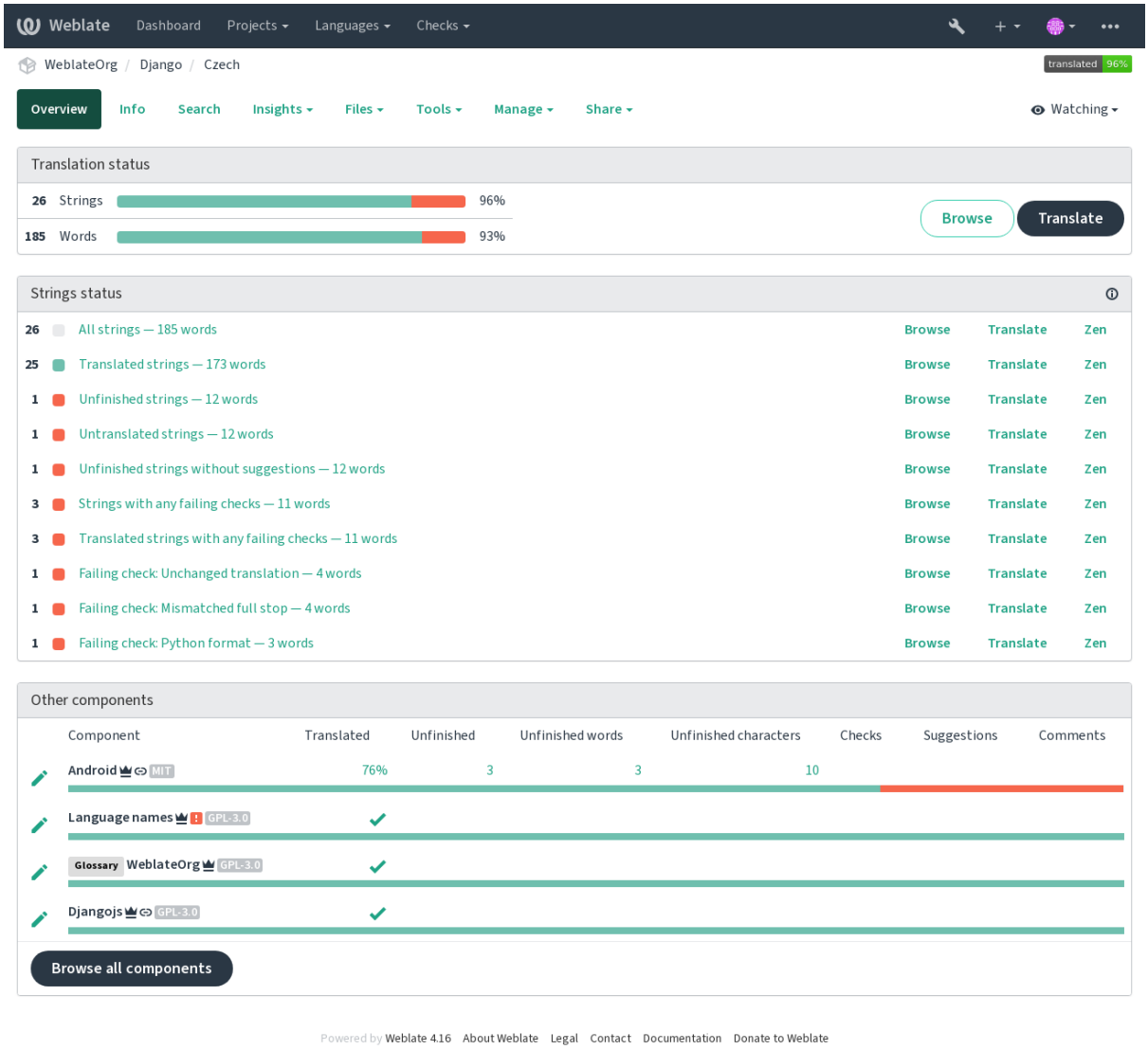
1.3.1 翻译项目

翻译项目包含相关组件；同一软件、书籍或项目的资源。



1.3.2 翻译链接

导航到一个组件后，一组链接会导致它的实际翻译。翻译进一步分为单独的检查，如 *未翻译的字符串*或 *未完成的字符串*。如果整个项目被翻译，没有错误，*所有字符串*仍然可用。或者，您可以使用搜索字段来查找特定的字符串或术语。



1.3.3 建议

备注： 实际权限可能因 Weblate 配置而异。

匿名用户只能（默认情况下）发送建议。如果翻译出现不确定性，登录用户仍然可以这样做，从而促使其他翻译人员对其进行审阅。

每天都会扫描这些建议，以删除与当前翻译匹配的重复项和建议。

1.3.4 评论

可以发布三种类型的评论：用于译文、源字符串或在使用启用原文审校 启用此功能时报告源字符串错误。选择一个适合您要讨论的主题。源字符串注释无论如何都有助于提供对原始字符串的反馈，例如应该改写它或提出有关它的问题。

你可以在所有评论中使用 Markdown 语法，并使用 `@mention` 提及其他用户。

参见：

report-source, 源字符串复查, 启用原文审校

1.3.5 变体

变体用于对字符串的不同长度变体进行分组。然后，项目的前端可以根据屏幕或窗口大小使用不同的字符串。

参见：

variants, 变体

1.3.6 标签

标签用于对项目中的字符串进行分类，以进一步自定义本地化工作流程（例如定义字符串的类别）。

Weblate 使用以下标签：

自动翻译
: 字符串是使用自动翻译 翻译的。

原文需要审校
: 字符串是使用源字符串复查 标记为需要复查的。

参见：

labels

1.3.7 翻译

在翻译页面上，将显示源字符串及其译文的编辑区域。如果译文是复数的，则会显示多个源字符串和编辑区域，每个源字符串和编辑区域都以翻译语言具有的复数形式的数量进行描述和标记。

所有特殊空白字符都会添加红色下划线，并用灰色符号表示。多个接续的空格也会添加红色下划线，以提醒译者潜在的格式问题。

此页面上可以显示各种额外信息，其中大部分来自项目源代码（如上下文、注释或消息的使用位置）。译者在首选项中选择的任何第二语言的翻译字段都将显示在源字符串上方（参见第二语言）。

在译文下方，译者可以看到其他人提出的建议，可以接受（✓），也可以接受修改（✏），或者删除（🗑）。

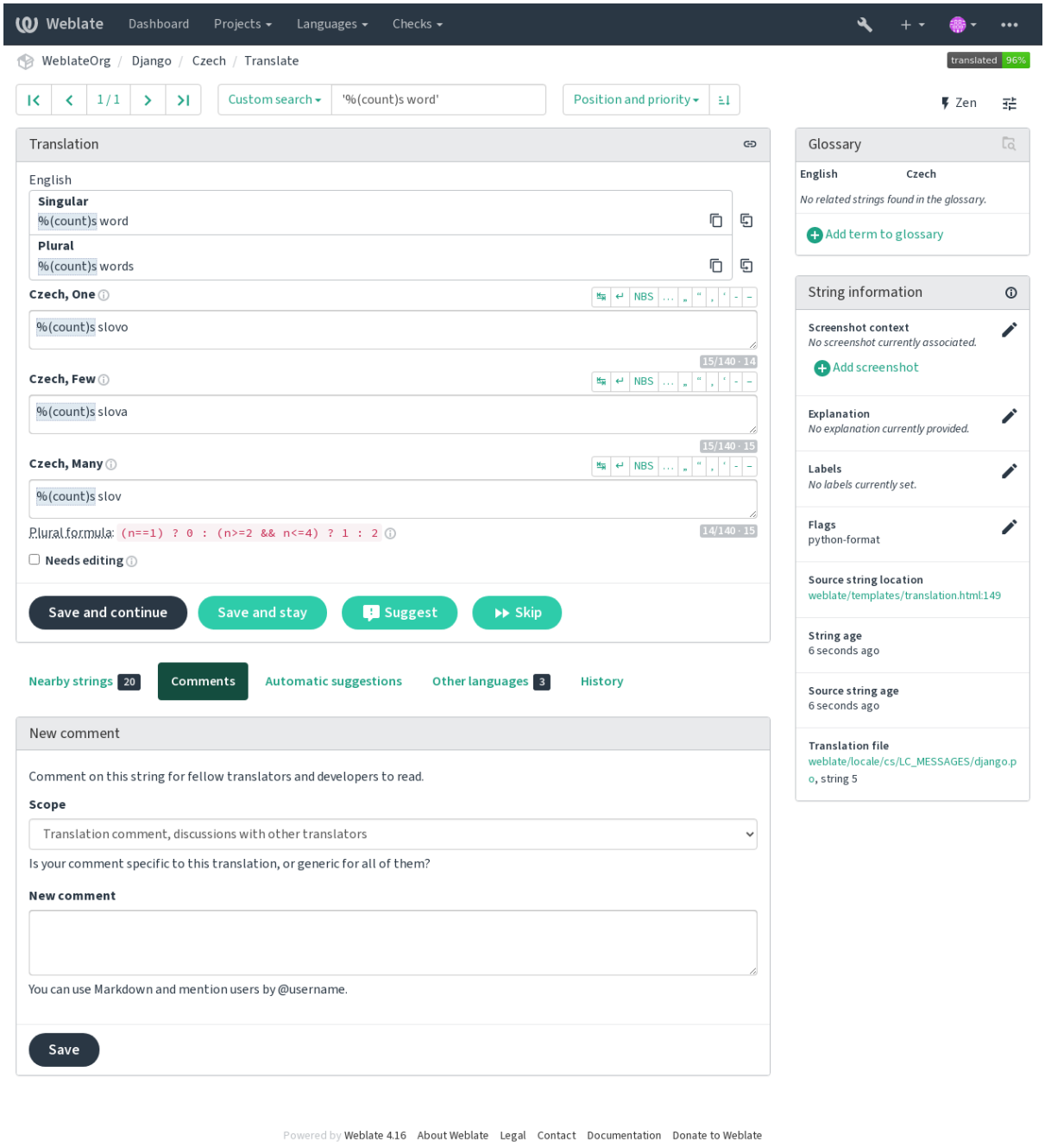
复数形式

改变形式的单词以解释其数字名称称为复数。每种语言都有自己的复数定义。例如，英语支持一种。在例如“car”的单数定义中，隐含地引用了一辆汽车，在复数定义中，“cars”引用了两辆或两辆以上的汽车（或将汽车的概念作为名词）。例如捷克语或阿拉伯语等语言具有更多的复数，并且它们的复数规则也不同。

Weblate 完全支持每种形式的每种语言（通过分别翻译每个复数形式）。字段的数量以及它在翻译的应用程序或项目中的使用方式取决于配置的复数公式。Weblate 显示了基本信息，Unicode 联盟的 语言复数规则 是更详细的描述。

参见：

复数公式



备选翻译

在 4.13 版本加入.

备注： 目前只有多值 CSV 文件 支持。

对于某些格式，单条字符串可以有多种翻译。您可以使用 工具菜单添加更多替代翻译。保存后将自动删除任何空白的替代翻译。

键盘快捷键

在 2.18 版本发生变更: 键盘快捷键在 2.18 版本中进行了改进，不太可能与浏览器或系统默认值发生冲突。

以下键盘快捷键可以在翻译中使用:

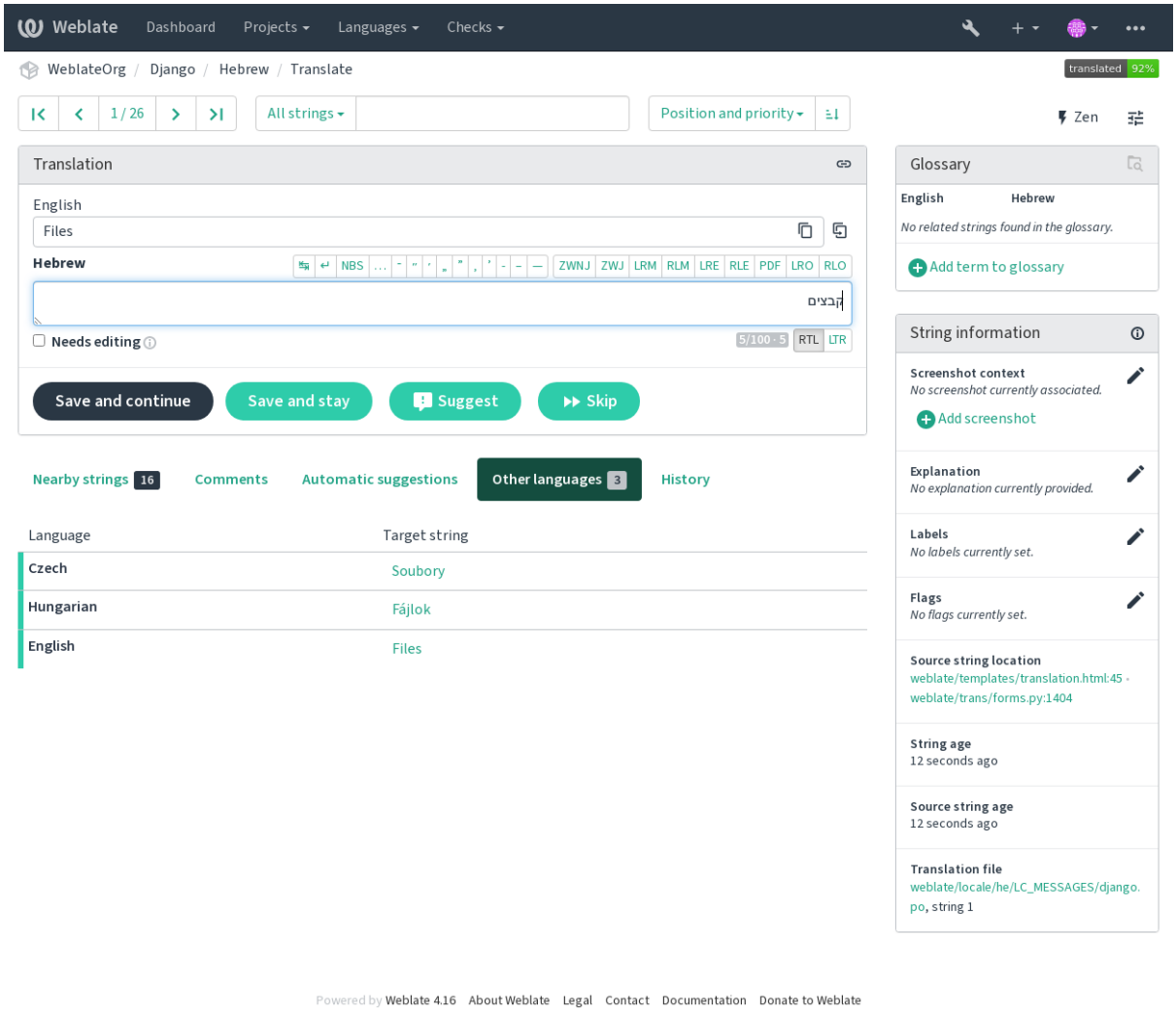
键盘快捷键	说明
Alt+Home	导航到当前搜索中的第一个翻译。
Alt+End	导航到当前搜索中的最后一个翻译。
Alt+PageUp 或 Ctrl+↑ 或 Alt+↑ 或 Cmd+↑	导航到当前搜索中的前一处翻译。
Alt+PageDown 或 Ctrl+↓ 或 Alt+↓ 或 Cmd+↓	导航到当前搜索中的下一处翻译。
Ctrl+Enter 或 Cmd+Enter	提交当前表格；这与在编辑翻译时按 保存并继续相同。
Ctrl+Shift+Enter 或 Cmd+Shift+Enter	取消标记翻译为“需要编辑”并提交它。
Alt+Enter 或 Option+Enter	将字符串提交为建议；这等同于在编辑译文时按下 建议。
Ctrl+E 或 Cmd+E	焦点翻译编辑器。
Ctrl+U 或 Cmd+U	焦点评论编辑器。
Ctrl+M 或 Cmd+M	显示 自动建议选项卡，请参见自动建议。
Ctrl+1 到 Ctrl+9 或 Cmd+1 到 Cmd+9	从源字符串复制给定数字的可放置。
Ctrl+M+1 到 9 或 Cmd+M+1 到 9	将给定数字的机器翻译复制到当前翻译。
Ctrl+I+1 到 9 或 Cmd+I+1 到 9	忽略未通过的检查列表中的一个项目。
Ctrl+J 或 Cmd+J	显示 附近字符串选项卡。
Ctrl+S 或 Cmd+S	定位到搜索框。
Ctrl+O 或 Cmd+O	复制源字符串。
Ctrl+Y 或 Cmd+Y	勾选/取消勾选 需要编辑复选框。

虚拟键盘

一个小的可视键盘行显示在翻译字段的正上方。这对于记住本地标点符号（因为行是每种语言的本地行）或具有难以键入的字符非常有用。

显示的符号分为三类:

- 用户配置的特殊字符 定义在用户个人资料 中
- Weblate 提供的每种语言字符（例如引号或 RTL 特定字符）
- 使用 *SPECIAL_CHARS* 配置的字符



翻译上下文

此上下文说明提供有关当前字符串的相关信息。

字符串属性
: 诸如消息 ID、上下文（`msgctxt`）或源代码中的位置之类的东西。

截屏
: 可以将屏幕截图上传到 Weblate，以便更好地告知翻译人员该字符串的使用位置和方式，请参阅字符串的可视化上下文。

附近字符串
: 显示翻译文件中相邻信息。这些信息通常是类似的上下文，并保证翻译的一致性。

其它的出现位置
: 如果一条信息出现在多个地方（例如多个部件），若发现它们不一致，这个选项卡会显示所有的信息（参见不一致的）。你可以选择使用其中之一。

翻译记忆库
: 查找过去翻译过的相似字符串，参见翻译记忆库。

术语表
: 显示当前信息中用到的项目术语表中的术语。

近期变化
: 显示最近通过 Weblate 更改了此信息的人员列表。

项目
项目信息，如翻译人员的说明，或项目使用的版本控制系统仓库中字符串的目录或链接。

如果你想要直接链接，翻译格式必须支持它。

翻译历史记录

默认情况下，每个更改（除非在部件设置中关闭）都保存在数据库中，并且可以还原。（可选）仍然可以还原基础版本控制系统中的任何内容。

已翻译的字符串长度

Weblate 可以通过多种方式限制翻译的长度，以确保翻译后的字符串不会太长：

- 翻译的默认限制是源字符串的十倍。这可以通过*LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH* 关闭。如果您遇到这种情况，也可能是由于单语翻译错误地设置为双语翻译，导致 Weblate 将翻译键误认为是实际的源字符串。更多信息参见双语和单语格式。
- 由翻译文件或标志定义的字符的最大长度，请参阅译文最大长度。
- 由标志定义的以像素为单位的最大渲染大小，请参阅最大译文长度。

1.3.8 自动建议

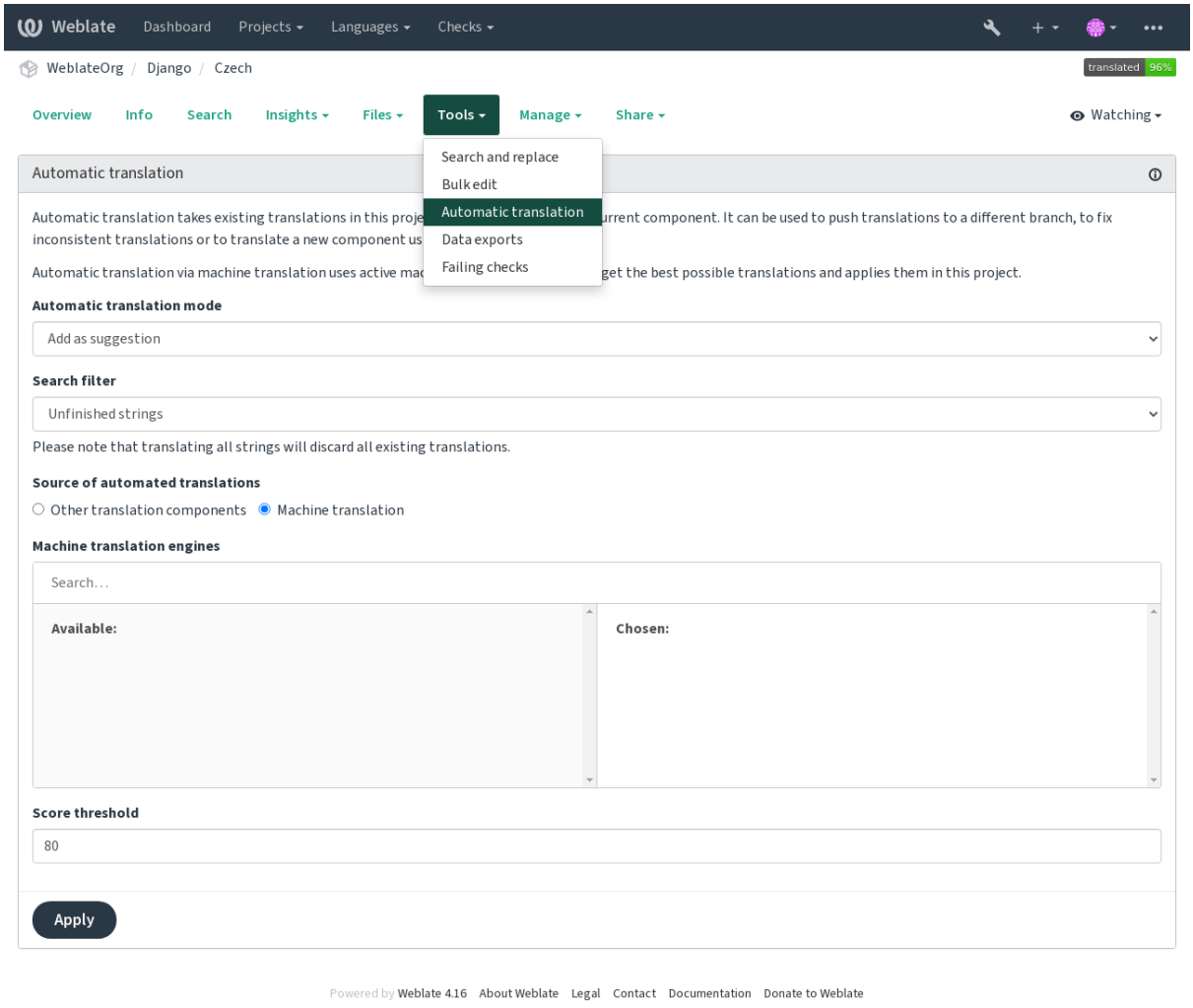
根据配置和您的翻译语言，Weblate 提供来自多个机器翻译工具和翻译记忆库 的建议。所有机器翻译都在每个翻译页面的一个选项卡中提供。

参见：

您可以在配置自动建议 中找到支持的工具列表。

1.3.9 自动翻译

你可以使用自动翻译来引导基于外部资源的翻译。这个工具叫做 *自动翻译*，一旦你选择了一个部件和一种语言，就可以在 *工具* 菜单中访问：



有两种操作模式可供选择：

- 使用其他 Weblate 部件作为翻译的来源。
- 使用选定的机器翻译服务，翻译高于特定质量阈值。

您还可以选择要自动翻译的字符串。

警告： 请注意，如果使用诸如 *所有字符串* 之类的宽过滤器，这将覆盖现有的翻译。

在以下几种情况下非常有用，例如在不同组件之间合并翻译（例如应用程序及其网站），或者使用现有翻译（翻译记忆库）为新组件引导翻译。

自动翻译的字符串会被打上 *自动翻译* 标签。

参见：

跨部件保持翻译一致

1.3.10 频次限制

为了避免滥用界面，速率限制适用于多个操作，如搜索，发送联系表单或翻译。如果受其影响，您将被阻止一段时间，直到您可以再次执行该操作。

默认限制和微调在管理手册中有描述，参见频次限制 。

1.3.11 搜索并替换

使用 工具菜单中的 搜索并替换有效地更改术语或执行字符串的批量修复。

提示： 不用担心会弄乱字符串。此过程分为两步：先显示要编辑字符串的预览，再确认实际改动。

1.3.12 批量编辑

批量编辑允许对字符串数量执行一个操作。您可以通过搜索字符串来定义字符串，并为匹配字符串设置要执行的操作。支持以下操作：

- 更改字符串状态（例如，批准所有未审阅的字符串）。
- 调整翻译标记（请参见使用标记定制行为）
- 调整字符串标签（请参见 labels）

提示： 这个工具叫做 批量编辑，可以在每个项目、部件或翻译的 工具菜单中访问。

参见：

批量编辑附加组件

1.3.13 矩阵视图

要有效地比较不同的语言，您可以使用矩阵视图。它在 工具菜单下的每个部件页面上都可用。首先选择您要比较的所有语言并确认您的选择，然后您可以单击任何翻译以快速打开和编辑它。

矩阵视图也是一个非常好的起点，可以找到不同语言中缺少的翻译，并从一个视图中快速添加它们。

1.3.14 禅模式

禅编辑器可以通过在翻译部件时单击右上角的 禅模式按钮来启用。它简化了布局并移除了额外的界面元素，如 附近字符串或 术语表。

您可以使用用户个人资料 上的首选项 选项卡选择禅编辑器作为默认编辑器。在这里，您还可以根据您的个人喜好选择到底是 从上到下还是 并排列出翻译。

1.4 下载和上传译文

您可以从翻译中导出文件，进行更改，然后再次导入。这允许脱机工作，然后将更改合并回现有译文中。即使在此期间已更改，这也有效。

备注： 可用选项可能会受到访问控制 设置的限制。

1.4.1 下载译文

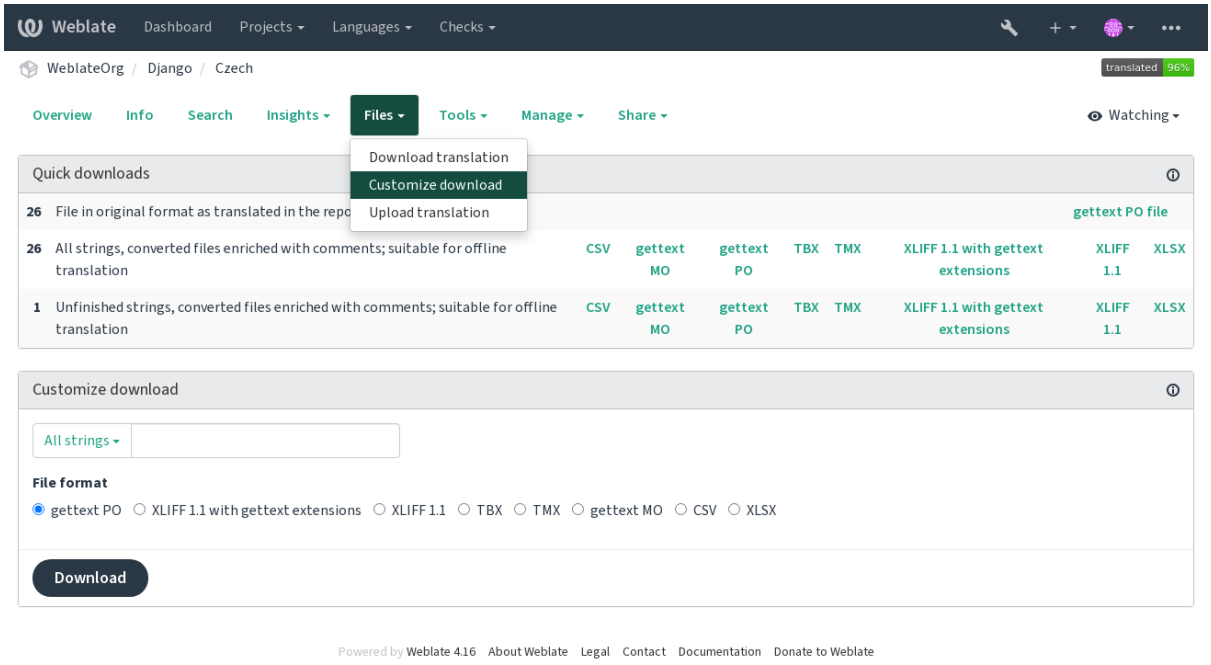
从项目或组件仪表板，可翻译文件可以在 文件菜单中下载。

第一个选项是下载存储在仓库中的原始格式的文件。在这种情况下，将提交翻译中的任何挂起更改，并且生成最新文件而不进行任何转换。

您还可以下载转换为一种广泛使用的本地化格式的翻译。转换后的文件将使用 Weblate 中提供的数据进行丰富；例如附加上下文、评论或标记。以下各种文件格式可通过 文件 ↓ 自定义下载菜单获得：

- gettext PO
- 带有 gettext 扩展的 XLIFF
- XLIFF 1.1
- TermBase eXchange（术语库交换）
- Translation Memory eXchange（翻译记忆库交换）
- gettext MO（仅当翻译使用 gettext PO 时可用）
- CSV
- Excel Open XML
- JSON（仅适用于单语翻译）
- Android 字符串资源（仅适用于单语翻译）
- iOS 字符串（仅适用于单语翻译）

提示： 转换后的文件中可用的内容因文件格式特性而异，您可以在翻译类型功能 中找到概述。

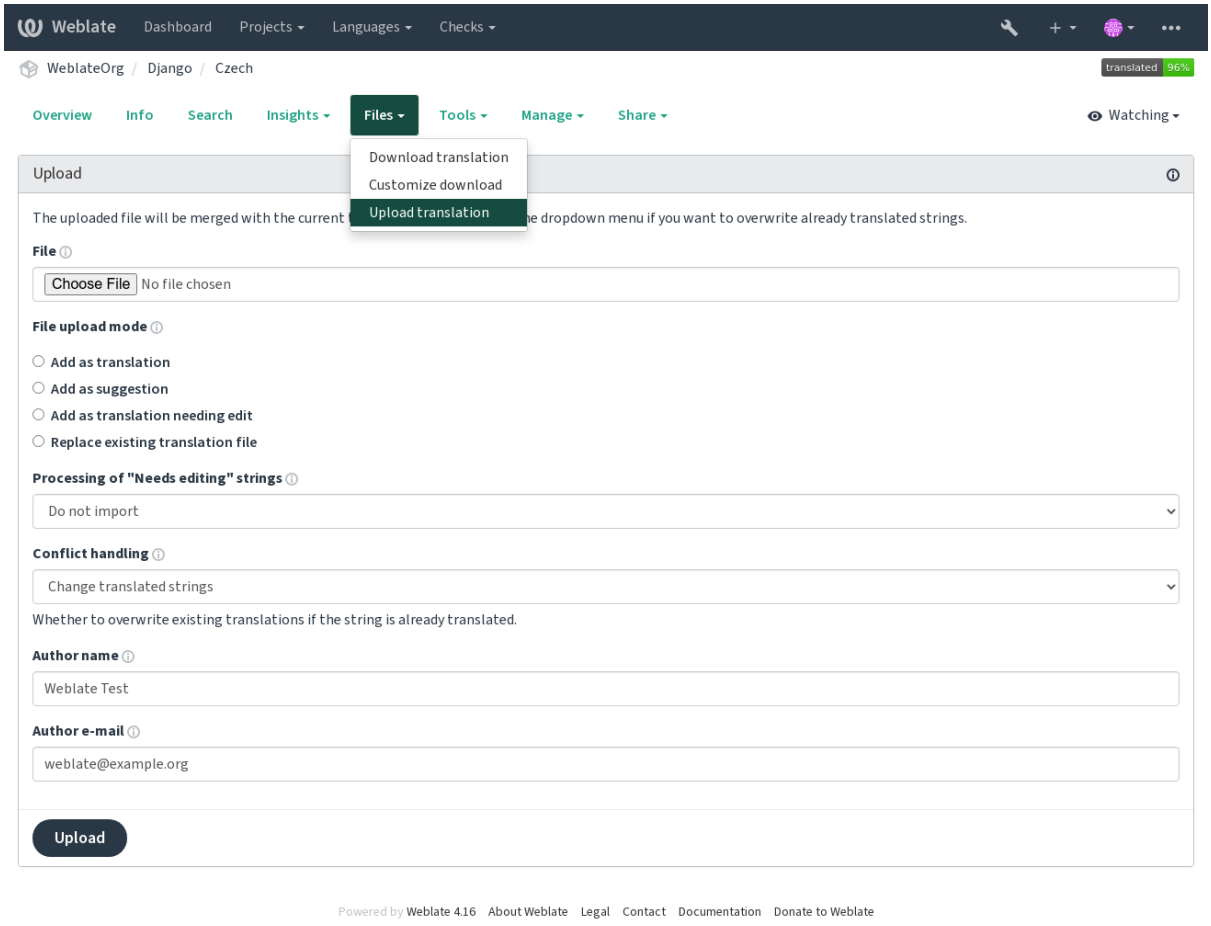


参见:

GET /api/translations/(string:project)/(string:component)/(string:language)/file/

1.4.2 上传译文

进行更改后，请使用 文件菜单中的 上传译文。



支持的文件格式

可以上传任何受支持文件格式的文件，但仍建议使用与用于翻译的文件格式相同的文件格式，否则某些功能可能无法正确翻译。

参见：

支持的文件格式, 下载和上传译文

导入方式

以下是上传翻译文件时显示的选项：

添加为译文（`translate`）
: 导入的字符串将作为译文添加到现有字符串中。这是最常见的用例，也是默认行为。

  仅使用上传文件中的译文，不使用其他内容。

添加为建议（`suggest`）
: 导入的字符串将添加为建议，当您希望审核上传的字符串时，请执行此操作。

  仅使用上传文件中的译文，不使用其他内容。

添加为需要编辑的译文（`fuzzy`）
: 导入的字符串将添加为需要编辑的译文。当您希望使用的译文也被审阅时，这可能很有用。

  仅使用上传文件中的译文，不使用其他内容。

替换现有翻译文件（`replace`）
: 现有文件将替换为新内容。这可能会导致现有译文丢失，请谨慎使用。

更新源字符串（source）
更新双语翻译文件中的源字符串。这类似于*更新 PO 文件以匹配 POT 文件 (msgmerge)* 所做的。

仅某些文件格式支持此选项。

添加新字符串（add）
将新字符串添加到翻译中。它将跳过已经存在的那个。

如果您想添加新字符串并更新现有译文，请使用 *添加为译文*第二次上传文件。

此选项仅在打开*管理字符串* 时可用。

仅使用上传文件中的原文、译文和键（上下文）。

参见:

POST /api/translations/(string:project)/(string:component)/(string:language)/file/

冲突处理

定义如何处理已经翻译的上传字符串。

需要编辑的字符串

还有一个选项可用于处理导入文件中需要编辑的字符串。可以通过以下三种方式之一处理此类字符串：“不导入”、“导入为‘需要编辑’”或“导入为已翻译”。

覆盖作者身份

使用管理员权限，您还可以指定上传文件的作者身份。如果您以其他方式收到文件并希望将其合并到现有翻译中，同时正确注明实际作者，这可能很有用。

1.5 术语表

每个项目可以包含一个或多个术语表，作为存储术语的简写。术语表易于保持译文的一致性。

每种语言的术语表可以单独管理，但它们作为单个组件存储在一起，这有助于项目管理员和多语言翻译人员保持一些跨语言的一致性。术语表中包含当前已翻译字符串中单词的术语将显示在翻译编辑器的边栏中。

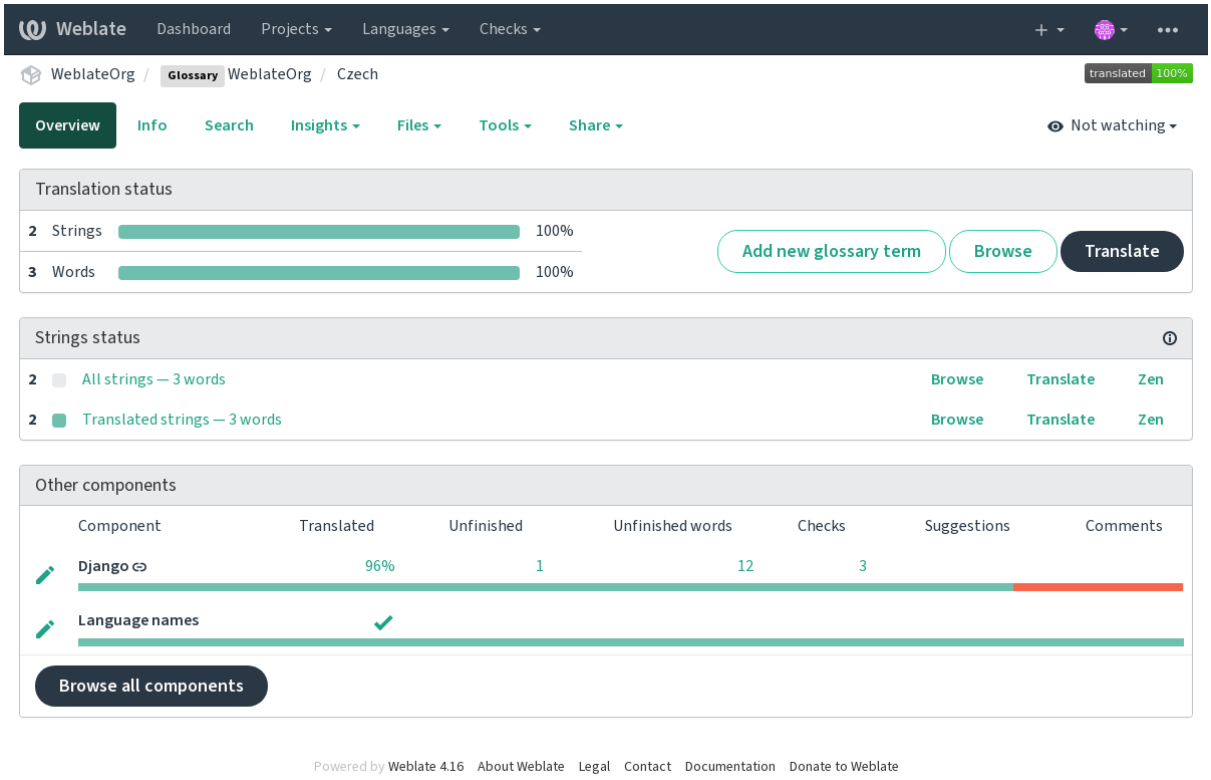
1.5.1 管理术语表

在 4.5 版本发生变更: 术语表现在是常规的翻译部件，您可以在其中使用所有 Weblate 功能——评论、存储在远程仓库中或添加解释。

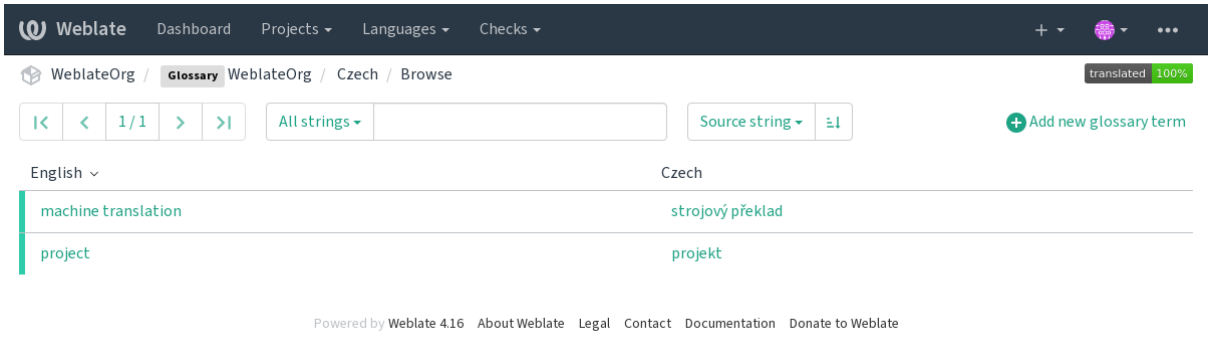
使用任何部件作为术语表，方法是开启*用作术语表*。你可以为一个项目创建多个术语表。

给定项目的空术语表会随项目自动创建。术语表在同一个项目的所有部件之间共享，并且可以选择使用来自相应术语表部件的*在项目中分享* 与其他项目共享。

术语表部件看起来与 Weblate 中的其他部件一样，只是多了个彩色标签：



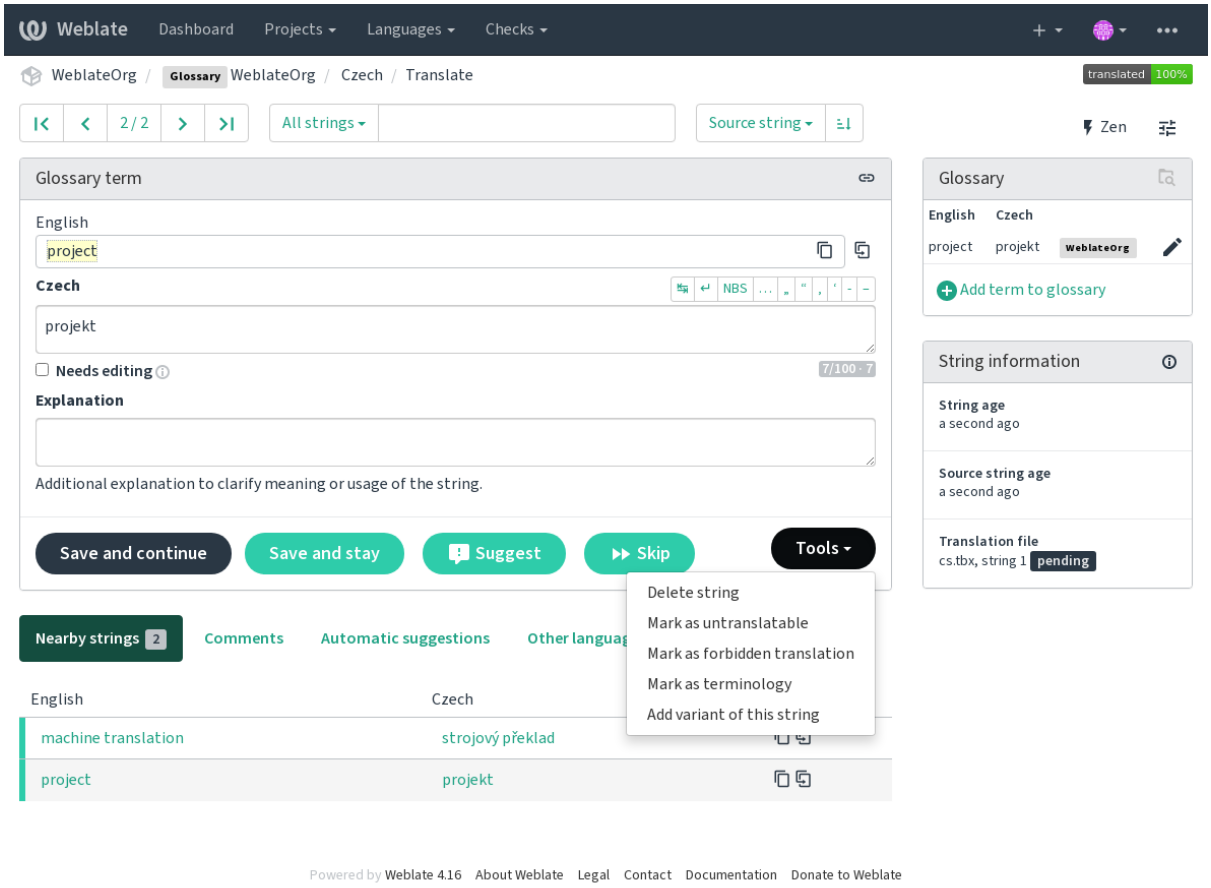
你可以浏览所有术语表术语:



或将它们当作任意翻译进行编辑。

1.5.2 术语表术语

术语表的翻译方式与普通字符串的翻译方式相同。你可以使用每个术语的 工具菜单为每个术语切换附加功能。



不可翻译的术语

在 4.5 版本加入.

通过批量编辑、输入标记或使用 工具 ↓ 标记为不可翻译来标记某些术语的译文为 `read-only` 意味着它们不能被翻译。对于品牌名称或其他不应该在其他语言中被改变的术语，可以使用这个方法。这样的术语会在术语表侧边栏中直观地突出显示。

参见:

使用标记定制行为

禁止的译文

在 4.5 版本加入.

通过批量编辑、输入标记或使用 工具 ↓ 标记为禁止的译文来将某些术语表术语译文标记为 `forbidden` 意味着它们 **不能**被使用。当某些单词模棱两可或可能具有意想不到的含义时，使用它来澄清译文。

参见:

使用标记定制行为

专业术语

在 4.5 版本加入.

通过批量编辑、键入标志或使用 工具 ↓ 标记为专业术语来将某些术语表术语标记为 terminology 以将条目添加到所有语言的术语表中。将此用于应该经过深思熟虑的重要术语，并在所有语言中保持一致的含义。

参见:

使用标记定制行为

变体

变体是将字符串组合在一起的通用方法。翻译时，所有术语变体都会列在术语表侧边栏中。

提示: 您可以使用它为术语添加缩写词或更短的表达方式。

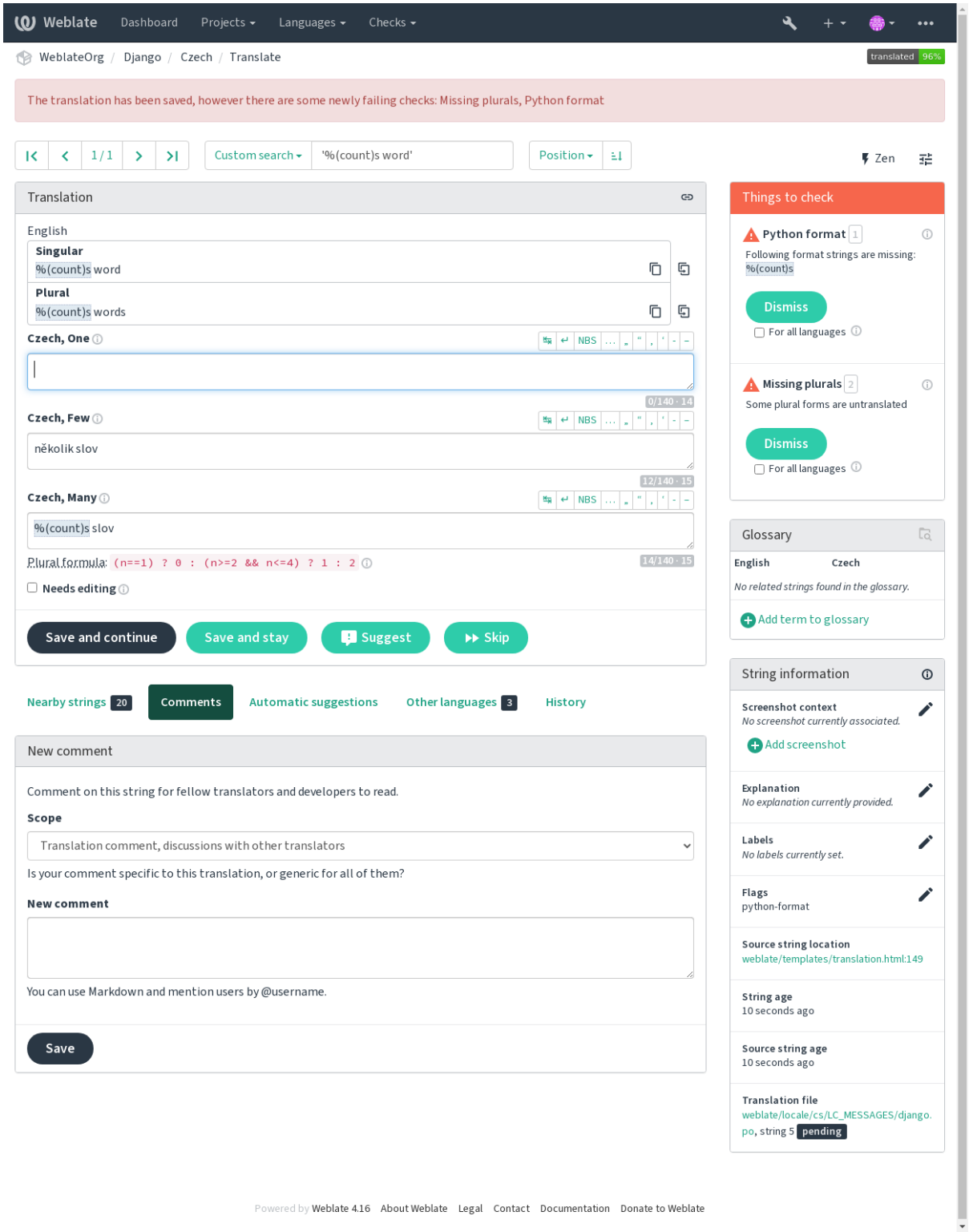
参见:

variants

1.6 检查和修正

质量检查有助于发现常见的翻译错误，确保翻译质量良好。如果出现误报，则可以忽略这些检查。

一旦提交了未通过检查的译文，会立即向用户显示：


Weblate
Dashboard
Projects
Languages
Checks
WeblateOrg / Django / Czech / Translate
translated 96%
The translation has been saved, however there are some newly failing checks: Missing plurals, Python format
1 / 1
Custom search
'%(count)s word'
Position
Zen
Translation
English
Singular
%(count)s word
Plural
%(count)s words
Czech, One
0/140 · 14
Czech, Few
několik slov
12/140 · 15
Czech, Many
%(count)s slov
14/140 · 15
Plural formula: (n==1) ? 0 : (n>=2 && n<=4) ? 1 : 2
Needs editing
Save and continue
Save and stay
Suggest
Skip
Nearby strings 20
Comments
Automatic suggestions
Other languages 3
History
New comment
Comment on this string for fellow translators and developers to read.
Scope
Translation comment, discussions with other translators
Is your comment specific to this translation, or generic for all of them?
New comment
You can use Markdown and mention users by @username.
Save
Things to check
Python format 1
Following format strings are missing: %(count)s
Dismiss
For all languages
Missing plurals 2
Some plural forms are untranslated
Dismiss
For all languages
Glossary
English
Czech
No related strings found in the glossary.
Add term to glossary
String information
Screenshot context
No screenshot currently associated.
Add screenshot
Explanation
No explanation currently provided.
Labels
No labels currently set.
Flags
python-format
Source string location
weblate/templates/translation.html:149
String age
10 seconds ago
Source string age
10 seconds ago
Translation file
weblate/locale/cs/LC_MESSAGES/django.po, string 5
pending
Powered by Weblate 4.16
About Weblate
Legal
Contact
Documentation
Donate to Weblate

1.6.1 自动修正

除了质量检查 外，Weblate 还可以自动修复翻译字符串中的一些常见错误。谨慎使用它，不要使其增加翻译错误。

参见：

AUTOFIX_LIST

1.6.2 质量检查

Weblate 对字符串进行了广泛的质量检查。以下部分将对它们进行更详细的描述。还有针对特定语言的检查。如果有错误报告，请将缺陷提交。

参见：

CHECK_LIST, *使用标记定制行为*

1.6.3 译文检查

在每次翻译更改时执行，帮助译者提交高质量的翻译。

BBCode 标记

概要
译文中的 BBCode 和原文不一致

范围
已翻译字符串

检查的类
`weblate.checks.markup.BBCodeCheck`

检查的标识符
`bbcode`

忽略的标记
`ignore-bbcode`

BBCode 表示简单的标记，例如以粗体或斜体突出显示消息的重要部分。

此检查确保在翻译中也找到它们。

备注： 目前检测 BBCode 的方法非常简单，所以此检查可能会产生误报。

连续重复的单词

在 4.1 版本加入.

概要
一行文本中包含同一单词两次：

范围
已翻译字符串

检查的类
`weblate.checks.duplicate.DuplicateCheck`

检查的标识符
`duplicate`

忽略的标记
`ignore-duplicate`

检查译文中是否有连续重复的单词出现。这通常表示译文中存在错误。

提示： 此检查包括特定语言的规则，以避免误报。如果在您的情况下错误触发，请告诉我们。请参阅*在 Weblate 中汇报问题*。

不遵循术语表

在 4.5 版本加入.

概要
译文未遵循术语表中定义的术语。

范围
已翻译字符串

检查的类
`weblate.checks.glossary.GlossaryCheck`

检查的标识符
`check_glossary`

启用的标记
`check-glossary`

忽略的标记
`ignore-check-glossary`

必须使用标记 `check-glossary` 以开启此检查（请参阅*使用标记定制行为*）。请在启用它之前考虑以下事项：

- 它进行精确的字符串匹配。预计术语表将包含所有变体中的术语。
- 根据术语表检查每条字符串的成本很高，它会减慢 Weblate 中涉及运行检查（如导入字符串或翻译）的任何操作。

参见：

术语表, *使用标记定制行为*, *翻译标记*

双空格

概要
译文包含双空格

范围
已翻译字符串

检查的类
`weblate.checks.chars.DoubleSpaceCheck`

检查的标识符
`double_space`

忽略的标记
`ignore-double-space`

检查翻译中是否存在双空格，以避免其他与空格相关的检查出现误报。

当在原文中找到双空格时，检查为假，这意味着双空格是故意的。

格式化字符串

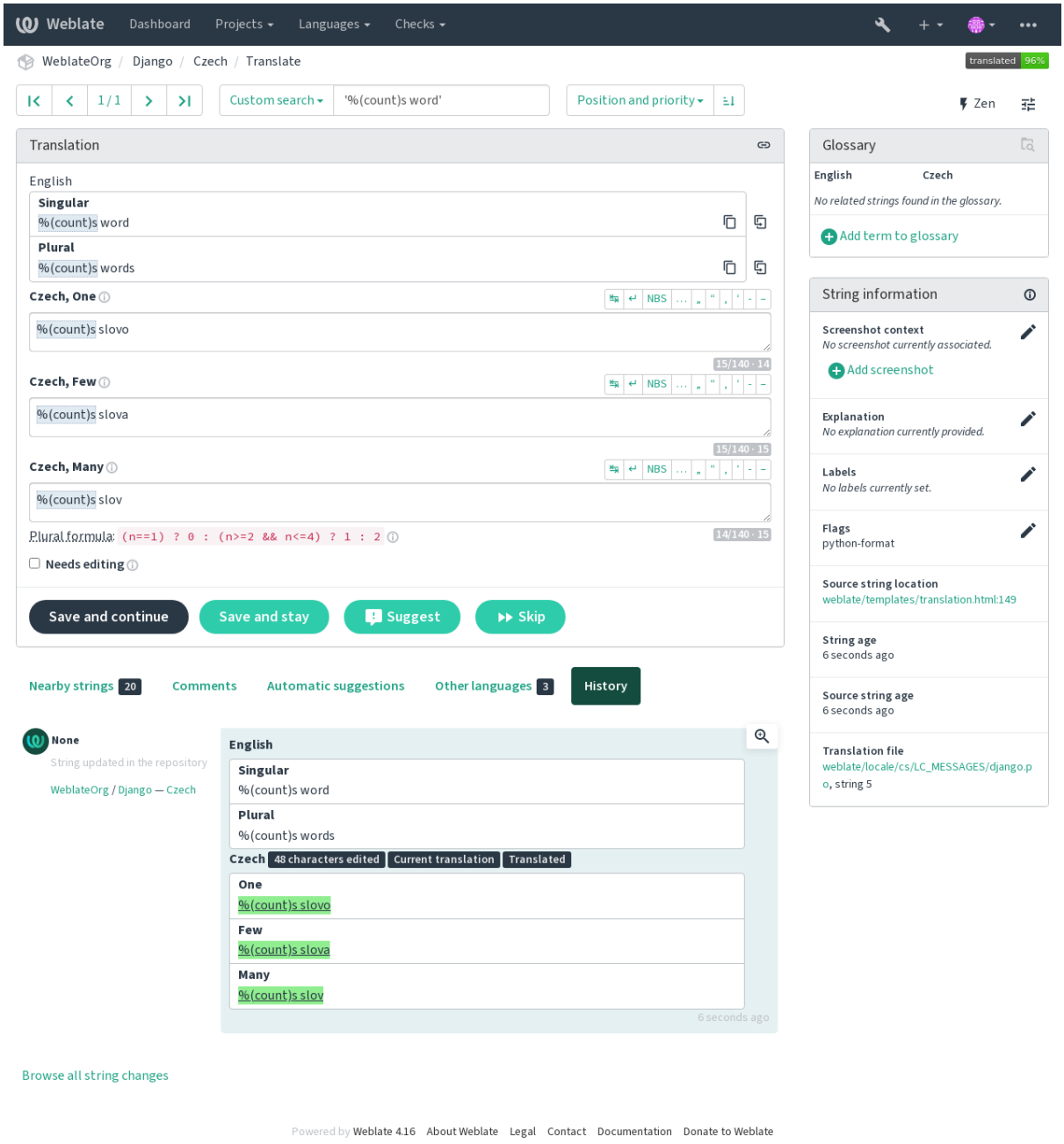
检查字符串中的格式化是否在原文和译文中都得到了复制。在译文中省略格式字符串通常会导致严重的问题，所以字符串中的格式化通常应与原文匹配。

Weblate 支持检查多种语言的格式字符串。仅当适当地标记了字符串时（例如 C 格式为 *c-format*），才会自动启用该检查。Gettext 会自动添加它，但是对于其他文件格式，或者如果您的 PO 文件不是由 **xgettext** 生成的，您可能必须手动添加它。

可以按每单位（请参阅源字符串另外的信息）或在部件配置 中完成此操作。为每个部件定义它比较简单，但是如果该字符串未解释为格式化字符串，而碰巧使用了格式化字符串语法，则可能导致误报。

提示： 如果 Weblate 中不提供特定格式的检查，则可以使用通用占位符。

除了检查，这也将高亮格式化字符串，方便将它们插入到已翻译字符串:



AngularJS 插值字符串

概要
AngularJS 插值字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.angularjs.AngularJSInterpolationCheck`

检查的标识符
`angularjs_format`

启用的标记
`angularjs-format`

忽略的标记
`ignore-angularjs-format`

命名格式字符串示例
`您的余额是 {{amount}} {{ currency }}`

参见:

格式化字符串, AngularJS 文本插值

C 格式

概要
C 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.CFormatCheck`

检查的标识符
`c_format`

启用的标记
`c-format`

忽略的标记
`ignore-c-format`

简单格式字符串示例
`这里有%d 个苹果`

位置格式字符串示例
`您的余额是%1$d %2$s`

参见:

格式化字符串,
C 格式字符串（英文）, C printf 格式（英文）

C# 格式

概要
C# 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.CSharpFormatCheck`

检查的标识符
`c_sharp_format`

启用的标记
`c-sharp-format`

忽略的标记
`ignore-c-sharp-format`

位置格式字符串示例
`这里有 {0} 个苹果`

参见:

格式化字符串 , C# 字符串格式

ECMAScript 模板字面量

概要
ECMAScript 模板字面量和原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.ESTemplateLiteralsCheck`

检查的标识符
`es_format`

启用的标记
`es-format`

忽略的标记
`ignore-es-format`

插值示例
`这里有 ${number} 个苹果`

参见:

格式化字符串, 模板字面量

i18next 插值

在 4.0 版本加入.

概要
i18next 插值和原文不一致

范围
已翻译字符串

检查的类
`weblate.checks.format.I18NextInterpolationCheck`

检查的标识符
`i18next_interpolation`

启用的标记
`i18next-interpolation`

忽略的标记
`ignore-i18next-interpolation`

插值示例
`这里有 {{number}} 个苹果`

嵌套示例
`这里有 $t(number) 个苹果`

参见:

格式化字符串, i18next 插值

ICU MessageFormat

在 4.9 版本加入.

概要
ICU MessageFormat 字符串语法错误和/或占位符不匹配。

范围
已翻译字符串

检查的类
`weblate.checks.icu.ICUMessageFormatCheck`

检查的标识符
`icu_message_format`

启用的标记
`icu-message-format`

忽略的标记
`ignore-icu-message-format`

插值示例
`这里有 {number, plural, one {1 个苹果} other {# 个苹果}}.`

此检查支持纯 ICU MessageFormat 消息以及带有简单 XML 标记的 ICU。您可以使用配置此检查的行为 `icu-flags:*`, 方法是选择支持 XML 或禁用某些子检查。例如，以下标记启用 XML 支持，同时禁用多个子消息的验证：

```
icu-message-format, icu-flags:xml:-plural_selectors
```

xml	启用对简单 XML 标记的支持。默认情况下，XML 标记被松散地解析。如果杂散 < 字符不是标签的合理部分，则它们将被忽略。
strict-xml	启用对严格的 XML 标签的支持。所有 < 字符如果不是标签的一部分，必须被转义。
-highlight	停用在编辑器中突出显示占位符。
-require_other	禁用要求子信息有 other 选择器。
-submessage_se	跳过检查子消息选择器是否与原文匹配。
-types	跳过检查占位符类型是否与原文匹配。
-extra	跳过检查是否存在源字符串中不存在的占位符。
-missing	跳过检查是否有源字符串中存在的占位符丢失。

此外当 strict-xml 未启用但 xml 已启用时，您可以使用该 icu-tag-prefix:PREFIX 标志来要求所有 XML 标记都以特定字符串开头。例如，以下标志只允许匹配以开头的 XML 标记 <x::

```
icu-message-format, icu-flags:xml, icu-tag-prefix:"x:"
```

这将匹配 <x:link> 点击此处</x:link> 但不匹配 这里 。

参见:

ICU MessageFormat 语法，格式化字符串，ICU：格式化消息，Format.JS：消息语法

Java 格式

概要
Java 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
weblate.checks.format.JavaFormatCheck

检查的标识符
java_printf_format

启用的标记
java-printf-format

忽略的标记
ignore-java-printf-format

简单格式字符串示例
这里有%d 个苹果

位置格式字符串示例
您的余额是%1$d %2$s

在 4.14 版本发生变更: 这曾经由 java-format 标记切换，为了与 GNU gettext 保持一致而进行了更改。

参见:

格式化字符串，Java 格式字符串

Java MessageFormat

概要
Java MessageFormat 字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.JavaMessageFormatCheck`

检查的标识符
`java_format`

无条件启用的标记
`java-format`

启用自动检测的标记
`auto-java-messageformat` 仅当源中存在格式字符串时才启用检查

忽略的标记
`ignore-java-format`

位置格式字符串示例
```
这里有 {0} 个苹果
```

在 4.14 版本发生变更: 这曾经由 `java-messageformat` 标记切换，为了与 GNU gettext 保持一致而进行了更改。

该检查验证格式字符串对 Java MessageFormat 类有效。除了匹配花括号中的格式字符串外，它还验证单引号，因为它们具有特殊含义。无论何时，都应把单引号写成 `''`. 如果没有配对，它将被视为引用的开始，并且在呈现字符串时不会显示。

参见:

格式化字符串, Java MessageFormat

JavaScript 格式

概要
JavaScript 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.JavaScriptFormatCheck`

检查的标识符
`javascript_format`

启用的标记
`javascript-format`

忽略的标记
`ignore-javascript-format`

简单格式字符串示例
```
这里有%d 个苹果
```

参见:

格式化字符串, JavaScript 格式字符串

Lua 格式

概要
Lua 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.LuaFormatCheck`

检查的标识符
`lua_format`

启用的标记
`lua-format`

忽略的标记
`ignore-lua-format`

简单格式字符串示例
`这里有%d 个苹果`

参见:

格式化字符串, Lua 格式字符串

Object Pascal 格式

概要
Object Pascal 格式字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.ObjectPascalFormatCheck`

检查的标识符
`object_pascal_format`

启用的标记
`object-pascal-format`

忽略的标记
`ignore-object-pascal-format`

简单格式字符串示例
`这里有%d 个苹果`

参见:

格式化字符串, Object Pascal 格式化字符串, Free Pascal 格式化字符串 Delphi 格式化字符串

百分比占位符

在 4.0 版本加入.

概要
提供的百分比占位符与原文不一致

范围
已翻译字符串

检查的类
`weblate.checks.format.PercentPlaceholdersCheck`

检查的标识符
`percent_placeholders`

启用的标记
`percent-placeholders`

忽略的标记
`ignore-percent-placeholders`

简单格式字符串示例
`这里有%number% 个苹果`

参见:

格式化字符串,

Perl 格式

概要
Perl 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.PerlFormatCheck`

检查的标识符
`perl_format`

启用的标记
`perl-format`

忽略的标记
`ignore-perl-format`

简单格式字符串示例
`这里有%d 个苹果`

位置格式字符串示例
`您的余额是%1$d %2$s`

参见:

格式化字符串, Perl sprintf, Perl 格式字符串

PHP 格式

概要
PHP 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.PHPFormatCheck`

检查的标识符
`php_format`

启用的标记
`php-format`

忽略的标记
`ignore-php-format`

简单格式字符串示例
`这里有%d 个苹果`

位置格式字符串示例
`您的余额是%1$d %2$s`

参见:

格式化字符串, PHP sprintf 文档, PHP 格式字符串

Python brace 格式

概要
Python 大括号格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.PythonBraceFormatCheck`

检查的标识符
`python_brace_format`

启用的标记
`python-brace-format`

忽略的标记
`ignore-python-brace-format`

简单格式字符串
`这里有 {} 个苹果`

命名格式字符串示例
`您的余额是 {amount} {currency}`

参见:

格式化字符串, Python brace 格式, Python 格式字符串

Python 格式

概要
Python 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.PythonFormatCheck`

检查的标识符
`python_format`

启用的标记
`python-format`

忽略的标记
`ignore-python-format`

简单格式字符串
`这里有%d 个苹果`

命名格式字符串示例
`您的余额是%(amount)d %(currency)s`

参见:

格式化字符串 , Python 字符串格式 , Python 格式字符串

Qt 格式

概要
Qt 格式字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.qt.QtFormatCheck`

检查的标识符
`qt_format`

启用的标记
`qt-format`

忽略的标记
`ignore-qt-format`

位置格式字符串示例
`这里有%1 个苹果`

参见:

格式化字符串, Qt QString::arg()

Qt 复数格式

概要
Qt 复数格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.qt.QtPluralCheck`

检查的标识符
`qt_plural_format`

启用的标记
`qt-plural-format`

忽略的标记
`ignore-qt-plural-format`

复数格式字符串示例
```
这里有%Ln 个苹果
```

参见:

格式化字符串, Qt i18n 指南

Ruby 格式

概要
Ruby 格式的字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.ruby.RubyFormatCheck`

检查的标识符
`ruby_format`

启用的标记
`ruby-format`

忽略的标记
`ignore-ruby-format`

简单格式字符串示例
```
这里有%d 个苹果
```

位置格式字符串示例
```
您的余额是%1$f %2$s
```

命名格式字符串示例
```
您的余额是%+.2<amount>f %<currency>s
```

命名模板字符串
```
您的余额是%{amount} %{currency}
```

参见:

格式化字符串, Ruby Kernel#sprintf

Scheme 格式

概要
Scheme 格式字符串与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.SchemeFormatCheck`

检查的标识符
`scheme_format`

启用的标记
`scheme-format`

忽略的标记
`ignore-scheme-format`

简单格式字符串示例
`这里有 ~d 个苹果`

参见:

格式化字符串 , Srfi 28 , Chicken Scheme 格式 , Guile Scheme 格式化输出

Vue I18n 格式化

概要
Vue I18n 格式化和原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.format.VueFormattingCheck`

检查的标识符
`vue_format`

启用的标记
`vue-format`

忽略的标记
`ignore-vue-format`

已命名格式
`这里有 {count} 个苹果`

Rails i18n 格式化
`这里有%{count} 个苹果`

链接的语言环境消息
`@:message.dio @:message.the_world!`

参见:

格式化字符串, Vue I18n 格式化, Vue I18n 链接区域设置信息

曾被翻译过

概要
这条字符串在过去曾被翻译过

范围
所有字符串

检查的类
`weblate.checks.consistency.TranslatedCheck`

检查的标识符
`translated`

忽略的标记
`ignore-translated`

表示一条字符串已经被翻译。当译文已在版本控制系统（VCS）中恢复或以其他方式丢失时，可能会发生这种情况。

不一致的

概要
此字符串在此项目中有不止一种译文，或者在某些部件中未翻译。

范围
所有字符串

检查的类
`weblate.checks.consistency.ConsistencyCheck`

检查的标识符
`inconsistent`

忽略的标记
`ignore-inconsistent`

Weblate 检查一个项目内所有翻译中相同字符串的译文，以帮助您保持翻译的一致性。

在一个项目中对一条字符串的不同译文，检查会失败。这也会导致检查结果的不一致。你可以在 *其它的出现位置* 选项卡上找到这条字符串的其他译文。

此检查应用于项目中所有开启了*允许同步翻译* 的部件。

提示： 出于性能原因，检查可能不会发现所有不一致，它限制了匹配的数量。

备注： 如果字符串在一个部件中被翻译，而在另一个部件中没有被翻译，这个检查也会启动。它可以作为一种快速的方法来手动处理在某些部件中未被翻译的字符串，只需点击显示在 *其它的出现位置* 选项卡中的每一行的 *使用此译文* 按钮。

您可以使用*自动翻译* 附加部件来自动翻译已经在另一个部件中已翻译的新添加的字符串。

参见：

跨部件保持翻译一致

使用了 Kashida 字母

在 3.5 版本加入.

概要
不应使用装饰性的 kashida 字母

范围
已翻译字符串

检查的类
`weblate.checks.chars.KashidaCheck`

检查的标识符
`kashida`

忽略的标记
`ignore-kashida`

装饰性的 Kashida 字母不能用于译文。这些也被称为 Tatweel。

参见:

维基百科上的 Kashida（英文）

Markdown 链接

在 3.5 版本加入.

概要
Markdown 链接和原文不一致

范围
已翻译字符串

检查的类
`weblate.checks.markup.MarkdownLinkCheck`

检查的标识符
`md-link`

启用的标记
`md-text`

忽略的标记
`ignore-md-link`

Markdown 链接和原文不一致。

参见:

Markdown 链接

Markdown 引用

在 3.5 版本加入.

概要
Markdown 链接引用和原文不一致

范围
已翻译字符串

检查的类
`weblate.checks.markup.MarkdownRefLinkCheck`

检查的标识符
`md-reflink`

启用的标记
`md-text`

忽略的标记
`ignore-md-reflink`

Markdown 链接引用和原文不一致。

参见:

Markdown 链接

Markdown 语法

在 3.5 版本加入.

概要
Markdown 语法与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.markup.MarkdownSyntaxCheck`

检查的标识符
`md-syntax`

启用的标记
`md-text`

忽略的标记
`ignore-md-syntax`

Markdown 语法与原文不匹配

参见:

Markdown span 元素

译文最大长度

概要
译文不应超过给定长度

范围
已翻译字符串

检查的类
`weblate.checks.chars.MaxLengthCheck`

检查的标识符
`max-length`

启用的标记
`max-length`

忽略的标记
`ignore-max-length`

检查翻译的长度是否符合可用空间的要求。这只会检查翻译字符的长度。

与其他检查不同，该标记应设置为 `key:value` 对，如 `max-length:100`。

提示： 这个检查看的是字符数，当使用比例字体来渲染文本时，这可能不是最好的衡量标准。最大译文长度 检查会检查文本的实际渲染。

`replacements`：标志可能也有助于在检查字符串之前扩展可放置对象。

当同时使用 `xml-text` 标志时，长度计算会忽略 XML 标记。

最大译文长度

概要
译文不应超过给定长度

范围
已翻译字符串

检查的类
`weblate.checks.render.MaxSizeCheck`

检查的标识符
`max-size`

启用的标记
`max-size`

忽略的标记
`ignore-max-size`

在 3.7 版本加入.

译文呈现的文本不应超过给定的大小。它使用换行呈现文本，并检查文本是否适合给定的边界。

此检查需要一个或两个参数 - 最大宽度和最大行数。如果未提供行数，则考虑一行文本。

您还可以通过 `font-*` 指令配置使用的字体（请参阅使用标记定制行为），例如以下翻译标志说以 ubuntu 字体大小 22 呈现的文本应该适合两行和 500 像素：

```
max-size:500:2, font-family:ubuntu, font-size:22
```

提示： 您可能希望在部件配置 中设置 `font-*` 指令，以便为组件中的所有字符串配置相同的字体。如果您需要为每条字符串自定义它，您可以覆盖每条字符串的这些值。

`replacements`：标志可能也有助于在检查字符串之前扩展可放置对象。

当同时使用 `xml-text` 标志时，长度计算会忽略 XML 标记。

参见：

管理字型，使用标记定制行为，译文最大长度

不匹配的 \n

概要
译文中的 \n 个数与原文不匹配

范围
已翻译字符串

检查的类
`weblate.checks.chars.EscapedNewlineCountingCheck`

检查的标识符
`escaped_newline`

忽略的标记
ignore-escaped-newline

通常转义的换行符对于格式化程序输出很重要。如果译文中的 \n 字面量与原文不匹配，则检查失败。

不匹配的冒号

概要
原文与译文没有都以冒号结尾

范围
已翻译字符串

检查的类
weblate.checks.chars.EndColonCheck

检查的标识符
end_colon

忽略的标记
ignore-end-colon

检查冒号是否在原文和译文之间复制。还检查了冒号是否存在于不属于它们的各种语言（中文或日语）。

参见:

维基百科上的冒号（英文）

不匹配的省略号

概要
原文与译文没有都以省略号结尾

范围
已翻译字符串

检查的类
weblate.checks.chars.EndEllipsisCheck

检查的标识符
end_ellipsis

忽略的标记
ignore-end-ellipsis

检查结尾省略号是否在原文和译文之间复制。这只检查真正的省略号（…），而不检查三个点（...）。

在打印中，省略号通常比三个点更好，并且使用文本到语音转换听起来更好。

参见:

维基百科上的省略号（英文）

不匹配的感叹号

概要
原文与译文没有都以感叹号结尾

范围
已翻译字符串

检查的类
weblate.checks.chars.EndExclamationCheck

检查的标识符
`end_exclamation`

忽略的标记
`ignore-end-exclamation`

检查感叹号是否在原文和译文之间复制。还会检查感叹号是否存在不属于它们的各种语言（中文，日语，韩语，亚美尼亚语，林布语，缅甸语或 Nko）。

参见：

维基百科上的感叹号（英文）

不匹配的句号

概要
原文与译文没有都以句号结尾

范围
已翻译字符串

检查的类
`weblate.checks.chars.EndStopCheck`

检查的标识符
`end_stop`

忽略的标记
`ignore-end-stop`

检查是否在原文和译文之间复制了句号。检查句号是否存在不属于它们的各种语言（中文，日语，梵文或乌尔都语）。

参见：

维基百科上的句号（英文）

不匹配的问号

概要
原文与译文没有都以问号结尾

范围
已翻译字符串

检查的类
`weblate.checks.chars.EndQuestionCheck`

检查的标识符
`end_question`

忽略的标记
`ignore-end-question`

检查问号是否在原文和译文之间复制。对于不属于它们的各种语言（亚美尼亚语，阿拉伯语，中文，韩语，日语，埃塞俄比亚语，瓦伊语或科普特语），也会检查问号的存在。

参见：

维基百科上的问号（英文）

不匹配的分号

概要
原文与译文没有都以分号结尾

范围
已翻译字符串

检查的类
`weblate.checks.chars.EndSemicolonCheck`

检查的标识符
`end_semicolon`

忽略的标记
`ignore-end-semicolon`

检查句子末尾的分号是否在原文和译文之间复制。

参见:

维基百科上的分号（英文）

不匹配的换行符

概要
译文中的换行数量和原文不一致

范围
已翻译字符串

检查的类
`weblate.checks.chars.NewLineCountCheck`

检查的标识符
`newline-count`

忽略的标记
`ignore-newline-count`

通常换行符对于格式化程序输出很重要。如果译文中的 `\n` 字面量与原文不匹配，则检查失败。

缺少复数形式

概要
某些复数形式未翻译

范围
已翻译字符串

检查的类
`weblate.checks.consistency.PluralsCheck`

检查的标识符
`plurals`

忽略的标记
`ignore-plurals`

检查源字符串的所有复数形式是否已翻译。有关如何使用每个复数形式的详细信息，请参阅字符串定义。在某些情况下，未能填写复数形式将导致在使用复数形式时不显示任何内容。

占位符

在 3.9 版本加入.

概要
译文缺少一些占位符

范围
已翻译字符串

检查的类
`weblate.checks.placeholders.PlaceholderCheck`

检查的标识符
`placeholders`

启用的标记
`placeholders`

忽略的标记
`ignore-placeholders`

在 4.3 版本发生变更: 你可以使用正则表达式作为占位符。

在 4.13 版本发生变更: 使用 `case-insensitive` 标志时，占位符不区分大小写。

译文缺少一些占位符。这些可以从翻译文件中提取或使用 `placeholders` 标志手动定义，更多可以用冒号分隔，带空格的字符串可以引用:

```
placeholders:$URL$:$TARGET$:"some long text"
```

如果您有一些占位符语法，则可以使用正则表达式:

```
placeholders:r"%[^% ]%"
```

您还可以使用不区分大小写的占位符:

```
placeholders:$URL$:$TARGET$,case-insensitive
```

参见:

使用标记定制行为

标点间距

在 3.9 版本加入.

概要
双标点符号前缺少不可换行的空格

范围
已翻译字符串

检查的类
`weblate.checks.chars.PunctuationSpacingCheck`

检查的标识符
`punctuation_spacing`

忽略的标记
`ignore-punctuation-spacing`

检查双标点符号（感叹号、问号、分号和冒号）之前是否存在不可中断的空格。此规则仅在法语或布列塔尼语等少数选定语言中使用，其中双标点符号前的空格是排版规则。

参见:

维基百科上的法语和英语间距（英文）

正则表达式

在 3.9 版本加入.

概要
译文与正则表达式不匹配

范围
已翻译字符串

检查的类
`weblate.checks.placeholders.RegexCheck`

检查的标识符
`regex`

启用的标记
`regex`

忽略的标记
`ignore-regex`

译文与正则表达式不匹配。该表达式从翻译文件中提取或使用 `regex` 标志手动定义:

```
regex:^foo|bar$
```

相同的复数形式

概要
一些复数形式采用了相同的译文

范围
已翻译字符串

检查的类
`weblate.checks.consistency.SamePluralsCheck`

检查的标识符
`same-plurals`

忽略的标记
`ignore-same-plurals`

如果译文中某些复数形式重复，则检查失败。在大多数语言中，它们必须不同。

换行符开头

概要
原文与译文没有都以换行符开头

范围
已翻译字符串

检查的类
`weblate.checks.chars.BeginNewlineCheck`

检查的标识符
`begin_newline`

忽略的标记
ignore-begin-newline

换行符通常出现在源字符串中是有充分理由的，在使用翻译文本时，省略或添加可能会导致格式问题。

参见:

换行符结尾

空格开头

概要
原文与译文的开头空格数量不一致

范围
已翻译字符串

检查的类
weblate.checks.chars.BeginSpaceCheck

检查的标识符
begin_space

忽略的标记
ignore-begin-space

字符串开头的空格通常用于界面中的缩进，因此保留很重要。

换行符结尾

概要
原文与译文没有都以换行结符尾

范围
已翻译字符串

检查的类
weblate.checks.chars.EndNewlineCheck

检查的标识符
end_newline

忽略的标记
ignore-end-newline

换行符通常出现在源字符串中是有充分理由的，在使用翻译文本时，省略或添加可能会导致格式问题。

参见:

换行符开头

空格结尾

概要
原文与译文没有都以空格结尾

范围
已翻译字符串

检查的类
weblate.checks.chars.EndSpaceCheck

检查的标识符
end_space

忽略的标记
`ignore-end-space`

检查原文和译文之间是否复制了尾部的空格。

拖尾空格通常用于分隔相邻元素，因此删除它可能会破坏布局。

未更改的译文

概要
原文和译文毫无差异

范围
已翻译字符串

检查的类
`weblate.checks.same.SameCheck`

检查的标识符
`same`

忽略的标记
`ignore-same`

如果原文和对应的译文字符串相同，至少到复数形式中的一种，则会发生这种情况。在所有语言中常见的一些字符串被忽略，并且各种标记被剥离。这减少了误报的数量。

此检查可以帮助查找错误未翻译的字符串。

此检查的默认行为是从检查中排除内置黑名单中的单词。这些单词经常被翻译。这对于避免短字符串的误报非常有用，短字符串仅由在多种语言中相同的单个单词组成。可以通过向字符串或组件添加 `strict-same` 标志来禁用此黑名单。

参见：

部件配置，使用标记定制行为

不安全的 HTML

在 3.9 版本加入.

概要
该译文使用了不安全的 HTML 标记

范围
已翻译字符串

检查的类
`weblate.checks.markup.SafeHTMLCheck`

检查的标识符
`safe-html`

启用的标记
`safe-html`

忽略的标记
`ignore-safe-html`

译文使用了不安全的 HTML 标记。必须使用 `safe-html` 标志启用此检查（请参阅使用标记定制行为）。还有附带的自动修复程序，可以自动清理标记。

提示： 当同时使用 `md-text` 标志时，也允许使用 Markdown 样式链接。

参见：

HTML 检查的执行者是 Ammonia 库。

URL

在 3.5 版本加入.

概要
译文未包含 URL

范围
已翻译字符串

检查的类
`weblate.checks.markup.URLCheck`

检查的标识符
`url`

启用的标记
`url`

忽略的标记
`ignore-url`

译文不包含 URL。仅当单元被标记为包含 URL 时才会触发。在这种情况下，译文必须是有效的 URL。

XML 标记

概要
译文中的 XML 标签和原文不一致

范围
已翻译字符串

检查的类
`weblate.checks.markup.XMLTagsCheck`

检查的标识符
`xml-tags`

忽略的标记
`ignore-xml-tags`

这通常意味着生成的输出看起来会有所不同。在大多数情况下，这不是更改译文所期望的结果，但有时确实如此。

检查 XML 标记是否在原文和译文之间复制。

备注： 此检查被 `safe-html` 标记禁用，因为它完成的 HTML 清理可能会生成无效 XML 的 HTML 标记。

XML 语法

在 2.8 版本加入.

概要
该译文不是有效的 XML

范围
已翻译字符串

检查的类
`weblate.checks.markup.XMLValidityCheck`

检查的标识符
`xml-invalid`

忽略的标记
`ignore-xml-invalid`

此 XML 标记无效。

备注： 此检查被 `safe-html` 标记禁用，因为它完成的 HTML 清理可能会生成无效 XML 的 HTML 标记。

零宽空格

概要
译文中含有额外的零宽空格字符

范围
已翻译字符串

检查的类
`weblate.checks.chars.ZeroWidthSpaceCheck`

检查的标识符
`zero-width-space`

忽略的标记
`ignore-zero-width-space`

零宽空格 (<U+200B>) 字符用于分隔单词中的消息（自动换行）。

由于它们通常是被错误插入的，一旦它们出现在翻译中，就会触发这个检查。一些程序在使用这个字符时可能会出现问题。

参见：

维基百科上的零宽空格（英文）

1.6.4 原文检查

原文检查可以帮助开发者提高源字符串的质量。

省略号

概要
这条字符串使用三个句点 (…) 代替了省略号 (…)

范围
源字符串

检查的类
`weblate.checks.source.EllipsisCheck`

检查的标识符
`ellipsis`

忽略的标记
`ignore-ellipsis`

当字符串使用三个点 (`...`)，而它应该使用一个省略号字符 (…)，时就会失败。

在大多数情况下，使用 Unicode 字符是更好的方法，并且看起来呈现得更好，并且使用文本到语音转换可能听起来更好。

参见:

维基百科上的省略号（英文）

ICU MessageFormat 语法

在 4.9 版本加入.

概要
ICU MessageFormat 字符串中的语法错误。

范围
源字符串

检查的类
`weblate.checks.icu.ICUSourceCheck`

检查的标识符
`icu_message_format_syntax`

启用的标记
`icu-message-format`

忽略的标记
`ignore-icu-message-format`

参见:

ICU MessageFormat

长期未翻译

在 4.1 版本加入.

概要
这条字符串已经很长时间未被翻译了

范围
源字符串

检查的类
`weblate.checks.source.LongUntranslatedCheck`

检查的标识符
`long_untranslated`

忽略的标记
`ignore-long-untranslated`

当字符串很长时间没有被翻译时，它可能表明源字符串中存在问题，使其难以翻译。

多项未通过的检查

概要
多种语言的译文都有未通过的检查

范围
源字符串

检查的类
`weblate.checks.source.MultipleFailingCheck`

检查的标识符
`multiple_failures`

忽略的标记
`ignore-multiple-failures`

该字符串的许多翻译都未通过质量检查。这通常表明可以采取一些措施来改进源字符串。

这种检查失败往往是由于句子末尾缺少句号或类似的小问题造成的，译者往往在译文中解决这些问题，而最好在源字符串中解决。

多个未命名的变量

在 4.1 版本加入.

概要
字符串中有多个未命名的变量，因此译者无法重新排序它们

范围
源字符串

检查的类
`weblate.checks.format.MultipleUnnamedFormatsCheck`

检查的标识符
`unnamed_format`

忽略的标记
`ignore-unnamed-format`

字符串中有多个未命名的变量，因此译者无法重新排序它们。

考虑使用命名变量来允许翻译人员重新排序。

未复数化

概要
这条字符串应该用作复数，但未使用其复数形式

范围
源字符串

检查的类
`weblate.checks.source.OptionalPluralCheck`

检查的标识符
`optional_plural`

忽略的标记
`ignore-optional-plural`

字符串用作复数形式，但不使用复数形式。如果您的翻译系统支持此功能，则应使用它的复数感知变体。

例如，在 Python 中使用 Gettext，它可以是：

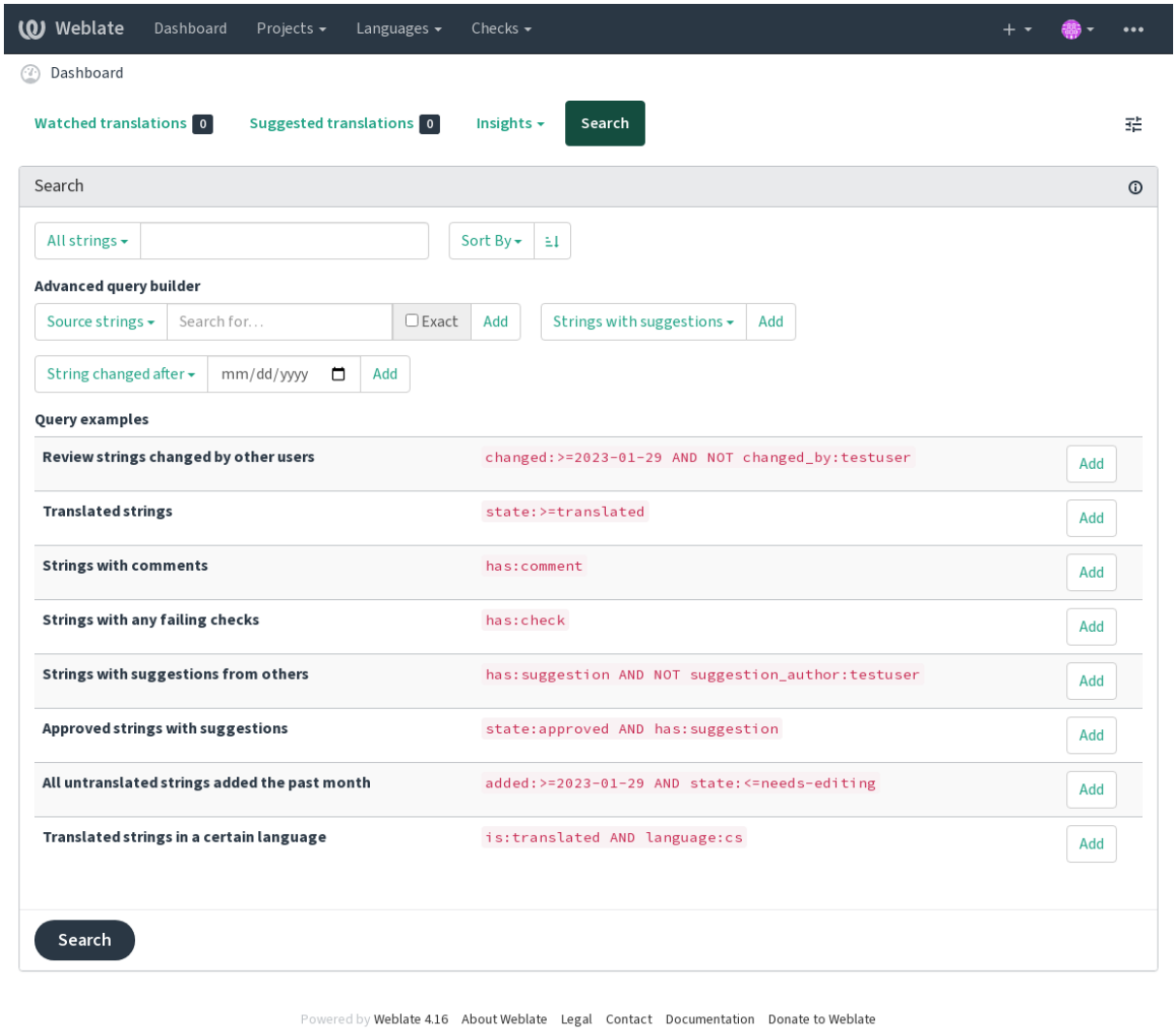
```
from gettext import ngettext

print(ngettext("Selected %d file", "Selected %d files", files) % files)
```

1.7 搜索

在 3.9 版本加入.

使用布尔运算、括号或特定于字段的查找的高级查询可用于查找所需的字符串。

如果未定义字段，则对源、目标和上下文字符串进行查找。



1.7.1 简单搜索

在搜索框中输入的任何短语都会被分割成单词。包含其中任何一个词的字符串都会被显示出来。要寻找一个精确的短语，把“搜索短语“放入引号中（单（‘）和双（“）引号都可以）：`"this is a quoted string"` 或 `'another quoted string'`.

1.7.2 字段

source:TEXT
不区分大小写的源字符串搜索。

target:TEXT
目标字符串不区分大小写的搜索。

context:TEXT
上下文字符串不区分大小写的搜索。

key:TEXT
键符串不区分大小写的搜索。

note:TEXT
不区分大小写的源字符串描述搜索。

location:TEXT
位置字符串不区分大小写的搜索。

priority:NUMBER
字符串优先级。

id:NUMBER
字符串唯一标识符。

added:DATETIME
字符串被添加到 Weblate 时的时间戳。

state:TEXT
搜索字符串状态 (`approved`, `translated`, `needs-editing`, `empty`, `read-only`)，支持字段运算符.

pending:BOOLEAN
待刷新到版本控制系统（VCS）的字符串。

has:TEXT
搜索具有属性的字符串 `-plural`, `context`, `suggestion`, `comment`, `check`, `dismissed-check`, `translation`, `variant`, `screenshot`, `flags`, `explanation`, `glossary`, `note`, `label`.

is:TEXT
搜索待处理的翻译 (`pending`)。还可以搜索所有字符串状态 (`approved`, `translated`, `untranslated`, `needs-editing`, `read-only`)。

language:TEXT
字符串目标语言。

component:TEXT
部件标识串或名称不区分大小写的搜索，请参阅部件标识串 和部件名称。

project:TEXT
项目标识串，参见*URL 标识串*。

changed_by:TEXT
字符串被作者用给定的用户名更改。

changed:DATETIME
字符串已于日期更，支持字段运算符.

change_time:DATETIME
字符串已于日期更改，支持字段运算符，不像 `changed` 这包括不更改内容的事件，您可以使用自定义操作过滤 `change_action` 。

change_action:TEXT
更改操作的过滤器，与 `change_time` 接受更改操作的英文名称，用引号括起来并用空格或小写字母和空格替换为连字符。有关示例，请参见搜索变更。

check:TEXT
字符串有未通过的检查，请参阅检查和修正 了解检查标识符。

dismissed_check:TEXT
字符串有忽略的检查，请参阅检查和修正 了解检查标识符。

comment:TEXT
在用户评论中搜索。

resolved_comment:TEXT
搜索已解决的评论。

comment_author:TEXT
按评论作者筛选。

suggestion:TEXT
在建议中搜索。

suggestion_author:TEXT
按建议作者筛选。

explanation:TEXT
在解释中搜索。

label:TEXT
搜索标签。

screenshot:TEXT
搜索截图。

1.7.3 布尔运算符

你可以使用 AND, OR, NOT 和括号组合查询，形成复杂的查询。例如：`state:translated AND (source:hello OR source:bar)`

1.7.4 字段运算符

您可以为日期或数字搜索指定运算符、范围或部分查找：

state:>=translated
状态是 translated 或更好 (approved).

changed:2019
在 2019 年有所更改。

changed:[2019-03-01 to 2019-04-01]
在两个给定的日期之间发生了更改。

1.7.5 精确运算符

您可以使用 = 运算符对不同的字符串字段进行完全匹配查询。例如，要搜索与完全匹配的所有源字符串 hello world, 请使用：`source:="hello world"` 。要搜索单个单词表达式，您可以跳过引号。例如，要搜索所有匹配的源字符串 hello ，您可以使用：`source:=hello`.

1.7.6 搜索变更

在 4.4 版本加入.

可以使用 change_action 和 change_time 运算符来搜索历史。

例如，搜索在 2018 年被标记为需要编辑的字符串可以输入 `change_time:2018 AND change_action:marked-for-edit` 或 `change_time:2018 AND change_action:"Marked for edit"`。

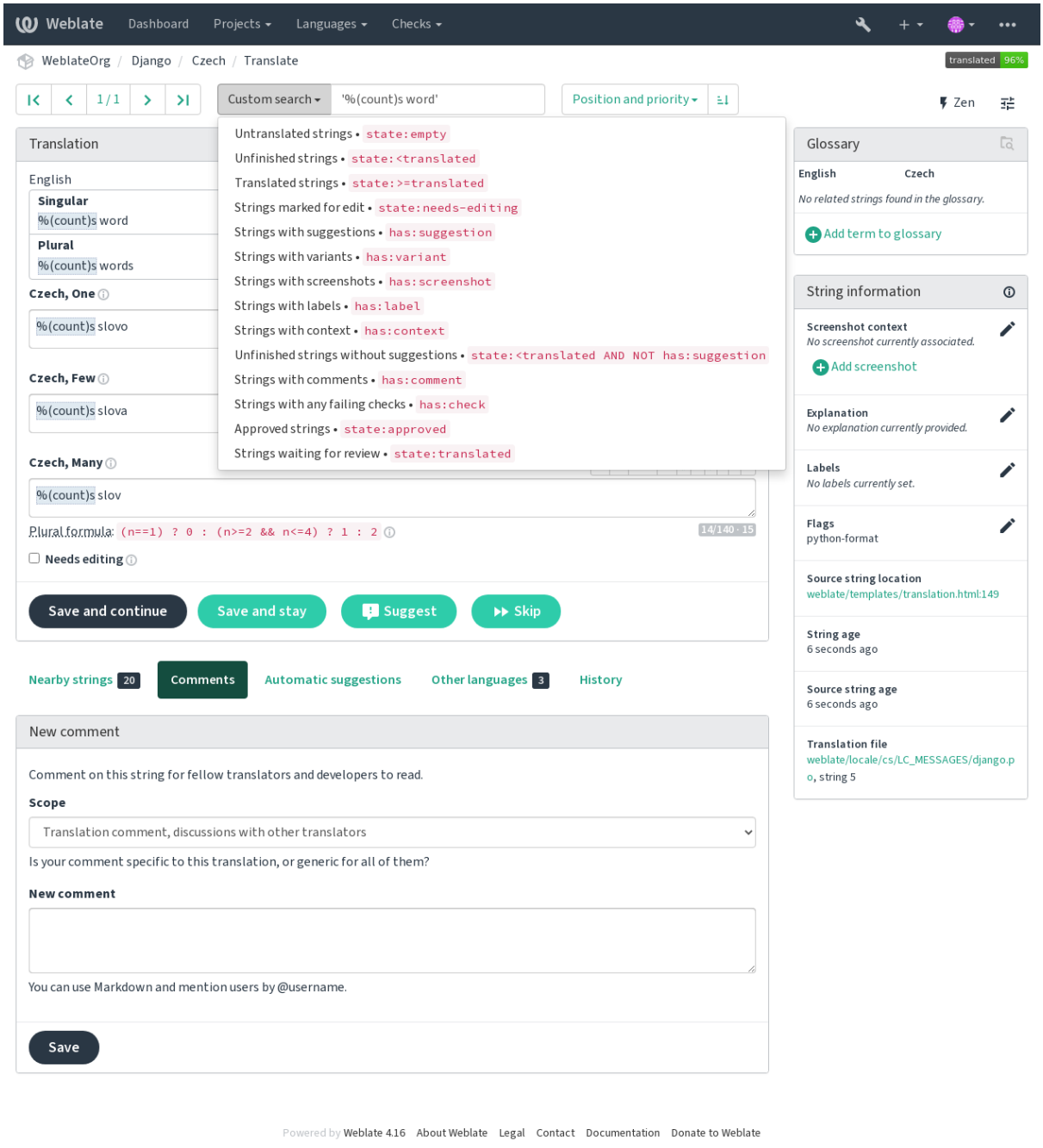
1.7.7 正则表达式

任何地方都可以接受文本，您也可以将正则表达式指定为 `r"regexp"`。

例如，要搜索包含 2 到 5 之间任何数字的所有源字符串，请使用 `source:r"[2-5]"`。

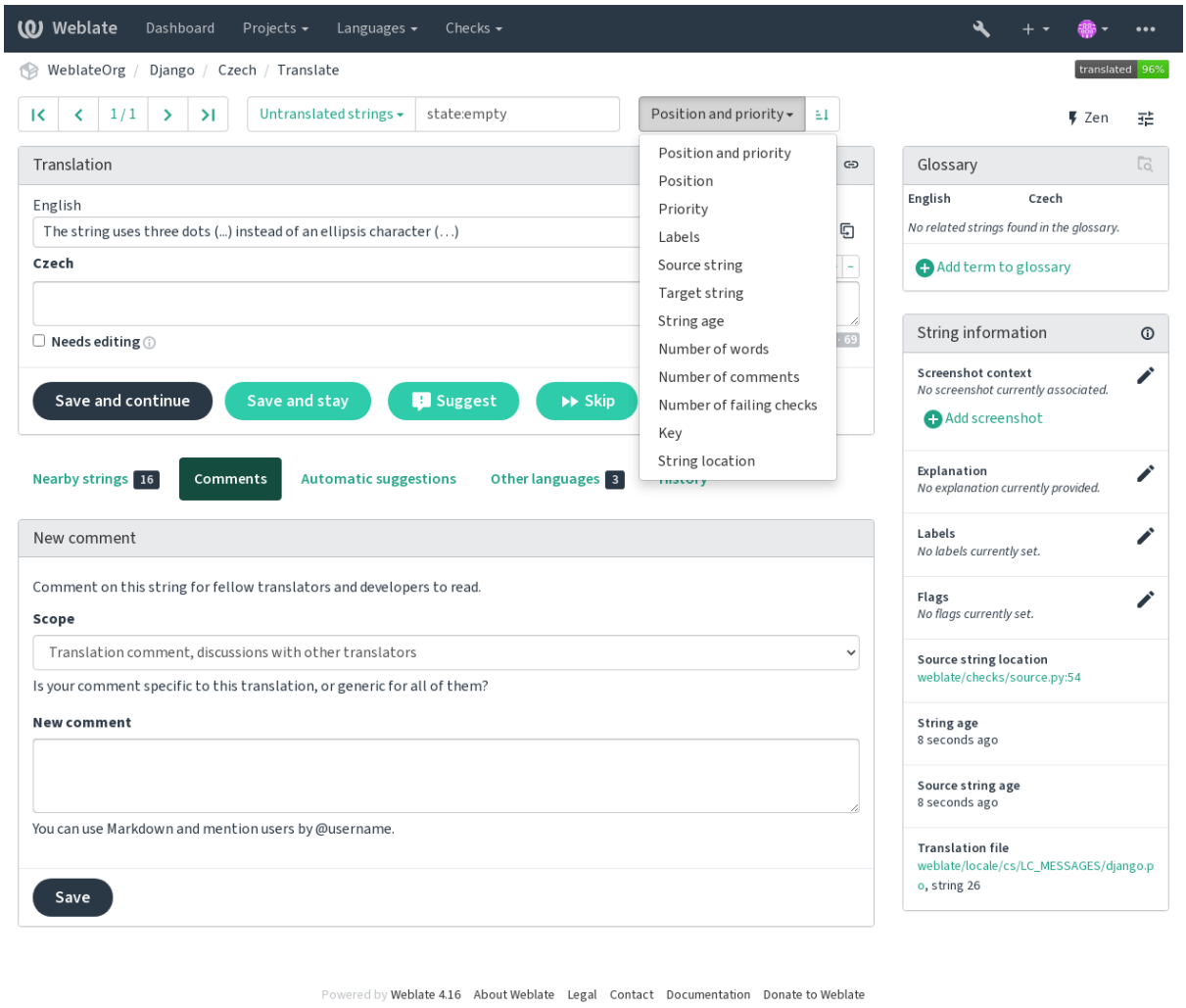
1.7.8 预定义查询

您可以在搜索页面上从预定义的查询中进行选择，这使您可以快速访问最常见的搜索：



1.7.9 对结果进行排序

有许多选项可以根据您的需要对字符串进行排序：



1.8 翻译工作流

使用 Weblate 是一个让您的用户更接近您的过程，它让您更接近您的翻译人员。由您决定要使用多少功能。

以下不是配置 Weblate 的方法的完整列表。您可以将其他工作流基于此处列出的最常见的示例。

1.8.1 翻译访问

访问控制 并没有作为一个整体在工作流程中被详细讨论，因它的大多数选项都可以应用于任何工作流。请参考有关如何管理对翻译的访问的相关文档。

在以下各章中，*任何用户*都是指有权访问翻译的用户。如果项目是公共项目，则可以是任何经过身份验证的用户，也可以是具有项目 *翻译*权限的用户。

1.8.2 翻译状态

每个翻译的字符串可以处于以下状态之一：

未翻译
翻译是空的，取决于文件格式，翻译是否可能存储在文件中。

需要编辑
翻译需要编辑，这通常是源字符串更改、模糊匹配或译者操作的结果。翻译存储在文件中，具体取决于文件格式，它可能被标记为需要编辑（例如，当它在 Gettext 文件中获得一个 `fuzzy` 标记）。

待审校
已翻译，但未进行审核。它作为有效翻译存储在文件中。

已核准
翻译已在审校中被核准。译者不能再更改它，只能由审校员更改。译者只能添加建议。

此状态仅在启用评论时可用。

建议
建议仅存储在 Weblate 中，而不存储在翻译文件中。

状态在可能的情况下会呈现在翻译文件中。

提示： 如果您使用的文件格式不支持存储状态，您可能需要使用将未更改的译文标记为“需要编辑” 附加组件将未更改的字符串标记为需要编辑。

参见：

翻译类型功能, 翻译工作流

1.8.3 直接翻译

这是小型团队最常用的设置，任何人都可以直接翻译。这也是 Weblate 中的默认设置。

- 任何用户都可以编辑翻译。
- 当译者不确定更改时，建议是建议更改的可选方法。

设置	值	备注
启用审校	已关闭	在项目级别配置。
启用建议	已开启	用户可以在不确定时提出建议，这很有用。
建议投票	已关闭	
自动接受建议	0	
译者群组	用户	或者通过每个项目的访问控制 进行 翻译。
审校员群组	不可用	未使用。

1.8.4 同行评审

使用此流程，任何人都可以添加建议，并且需要其他成员的核准才能被接受为翻译。

- 任何用户都可以添加建议。
- 任何用户都可以对建议投票。
- 在获得预定数量的票数后，建议就会成为翻译。

设置	值	备注
启用审校	已关闭	在项目级别配置。
启用建议	已开启	
建议投票	已关闭	
自动接受建议	1	您可以设置更高的值，以要求更多的同行评审。
译者群组	用户	或者通过每个项目的访问控制 进行 翻译。
审校员群组	不可用	未使用，所有译者都审阅。

1.8.5 专门的审校员

在 2.18 版本加入: 从 Weblate 2.18 开始，支持正确的审核工作流。

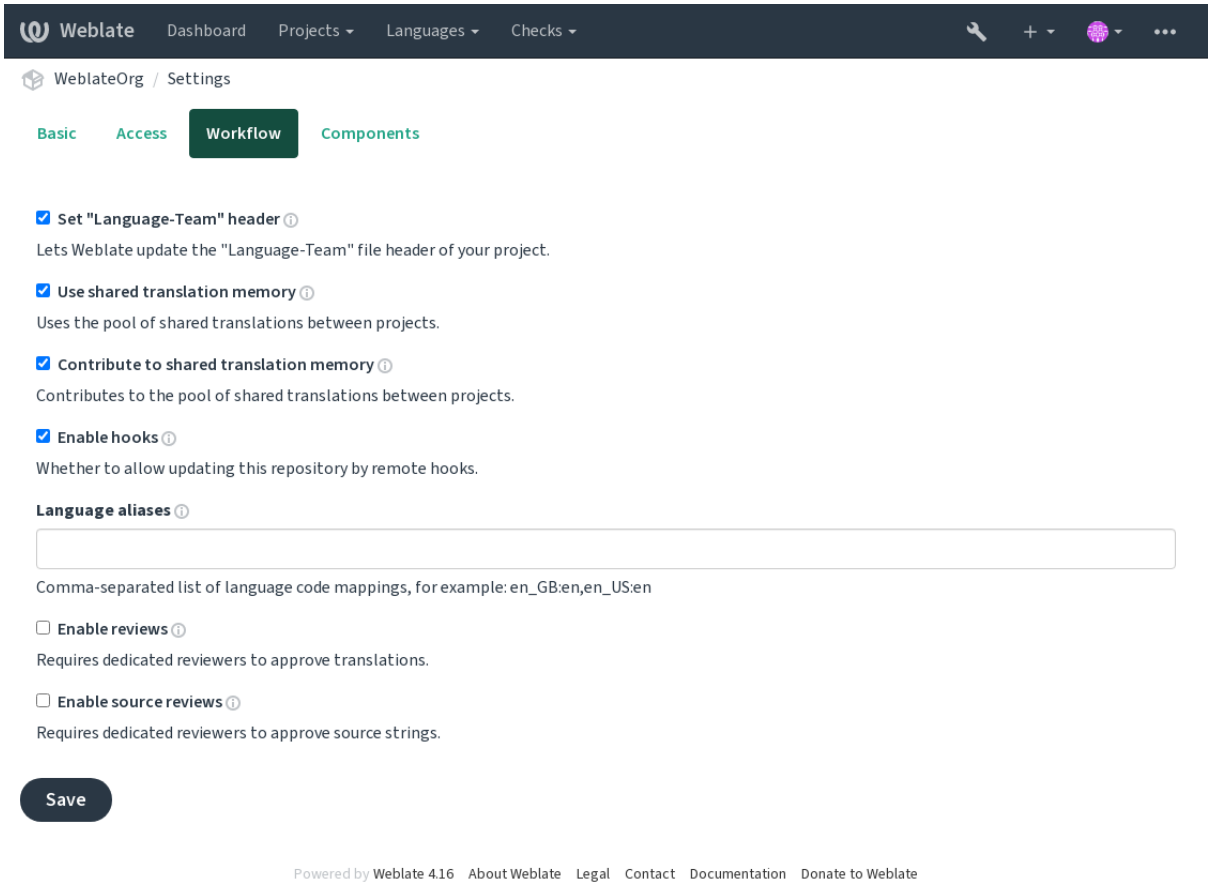
有了专门的审校员，您就有了两组用户，一组可以提交翻译，另一组可以审校翻译，以确保翻译一致且质量良好。

- 任何用户都可以编辑未核准的翻译。
- 审校员可以核准/取消核准字符串。
- 审校员可以编辑所有翻译（包括已核准的翻译）。
- 建议还可以用于建议更改已核准的字符串。

设置	值	备注
启用审校	已开启	在项目级别配置。
启用建议	已关闭	用户可以在不确定时提出建议，这很有用。
建议投票	已关闭	
自动接受建议	0	
译者群组	用户	或者通过每个项目的访问控制 进行 翻译。
审校员群组	审校员	或使用每个项目的访问控制 进行 审校。

1.8.6 开启审校

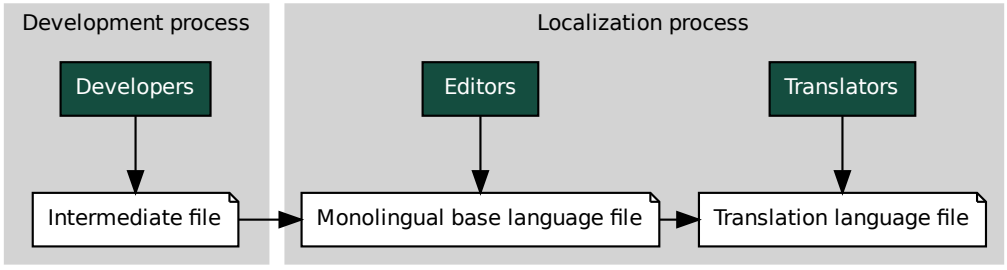
可以在项目设置的 工作流子页面中的项目配置中打开审核（位于 管理 → 设置菜单中）：



1.8.7 源字符串质量把关

在许多情况下，原始源语言字符串来自开发人员，因为他们编写代码并提供初始字符串。但是，开发人员通常不是源语言的母语人士，并且不提供所需质量的源字符串。中间翻译可以帮助您解决这个问题 - 开发人员，翻译人员和用户之间的字符串有额外的质量网关。

通过设置中间语言文件，这个文件将被用作字符串的源，但它会被编辑为源语言来润色它。一旦字符串在源语言中准备就绪，翻译人员也可以将其翻译成其他语言。



参见:

中间语言文件, 单语言译文模版语言文件, 双语和单语格式

1.8.8 源字符串复查

通过启用原文审校，可以对源字符串应用复查过程。启用后，用户可以报告源字符串中的问题。实际过程取决于使用的是双语言还是单语言格式。

对于单语言格式，源字符串复查的行为与专门的审校员 相似——一旦源字符串汇报了情况，就会被标记为 需要编辑。

双语格式不允许直接编辑源字符串（通常都是直接从源代码中提取源字符串）。在这种情况下，原文需要审校标签会被附加到译者报告的字符串上。您应该复查此类字符串，然后编辑原文或删除标签。

参见：

双语和单语格式, 专门的审校员, labels, 评论

1.9 常见问题

1.9.1 配置

如何创建自动化工作流？

Weblate 可以为您半自动处理所有翻译工作。如果授予它对仓库的推送访问权限，则翻译可以在没有交互的情况下进行，除非发生某些合并冲突。

1. 设置好 Git 仓库，以便在发生更改时通知 Weblate，请参阅通知钩子 获取有关如何操作的信息。
2. 在 Weblate 的部件配置 中设置推送 URL，这会允许 Weblate 将更改推送到仓库。
3. 在 Weblate 的部件配置 中开启提交时推送，这将使 Weblate 在 Weblate 发生更改时将更改推送到仓库。

参见：

持续本地化, 避免合并冲突

如何通过 SSH 访问仓库？

设置 SSH 密钥的信息请参阅访问仓库。

如何修复翻译中的合并冲突？

当 Weblate 和上游仓库的翻译文件同时被更改时，合并冲突时有发生。通常可以通过在对翻译文件进行更改前（例如在运行 msgmerge 前）先合并 Weblate 的翻译来避免这种情况。只需告诉 Weblate 提交所有待处理的翻译（可以在 管理菜单中的 仓库维护中执行此操作），并合并仓库（如果没有开启自动推送）。

如果你已经遇到了合并冲突，那么在您的机器上本地解决所有冲突的最简单方法是将 Weblate 添加为远程仓库，将它合并到上游，并修复任何冲突。一旦将更改推送回去，Weblate 就将能够使用合并的版本而无需任何其它特殊操作。

备注： 取决于你的设置，访问 Weblate 仓库可能需要身份验证。使用 Weblate 中内建的*Git 导出器* 时，你需使用用户名和 API 密钥进行身份验证。

```
# Commit all pending changes in Weblate, you can do this in the UI as well:
wlc commit
# Lock the translation in Weblate, again this can be done in the UI as well:
wlc lock
# Add Weblate as remote:
```

(续下页)

(接上页)

```
git remote add weblate https://hosted.weblate.org/git/project/component/
# You might need to include credentials in some cases:
git remote add weblate https://username:APIKEY@hosted.weblate.org/git/project/
↪component/

# Update weblate remote:
git remote update weblate

# Merge Weblate changes:
git merge weblate/main

# Resolve conflicts:
edit …
git add …
…
git commit

# Rebase changes (if Weblate is configured to do rebases)
git rebase origin/main

# Push changes to upstream repository, Weblate will fetch merge from there:
git push

# Open Weblate for translation:
wlc unlock
```

如果在 Weblate 中使用多个分支，那么可以对所有的分支做相同的事：

```
# Add and update Weblate remotes
git remote add weblate-one https://hosted.weblate.org/git/project/one/
git remote add weblate-second https://hosted.weblate.org/git/project/second/
git remote update weblate-one weblate-second

# Merge QA_4_7 branch:
git checkout QA_4_7
git merge weblate-one/QA_4_7
... # Resolve conflicts
git commit

# Merge main branch:
git checkout main
git merge weblates-second/main
... # Resolve conflicts
git commit

# Push changes to the upstream repository, Weblate will fetch the merge from there:
git push
```

在 gettext PO 文件的情况下，有一种方式以半自动方式来合并冲突：

取回并保存 Weblate Git 仓库的本地克隆。还要得到上游 Git 仓库的第二个新的本地克隆（也就是需要上游 Git 仓库的两份复件：完整的复件和工作复件）：

```
# Add remote:
git remote add weblate /path/to/weblate/snapshot/

# Update Weblate remote:
git remote update weblate

# Merge Weblate changes:
git merge weblate/main
```

(续下页)

(接上页)

```
# Resolve conflicts in the PO files:
for PO in `find . -name '*.po'` ; do
    msgcat --use-first /path/to/weblate/snapshot/$PO\
               /path/to/upstream/snapshot/$PO -o $PO.merge
    msgmerge --previous --lang=${PO%.po} $PO.merge domain.pot -o $PO
    rm $PO.merge
    git add $PO
done
git commit

# Push changes to the upstream repository, Weblate will fetch merge from there:
git push
```

参见:

如何导出 *Weblate* 使用的 *Git* 仓库？, 持续本地化, 避免合并冲突, *Weblate* 客户端

如何同时翻译几个分支？

Weblate 支持在一个项目配置 内推送翻译更改。对于每个将其打开的部件配置 （默认行为），所作的更改自动传递给其它部件。即使分支本身已经非常多样化了，也能以这种方式保持同步，并且不能在他们之间简单地合并翻译更改。

一旦从 Weblate 合并了更改，你可能不得不合并这些分支（取决于于您的开发工作流程），并丢弃差异：

```
git merge -s ours origin/maintenance
```

参见:

跨部件保持翻译一致

如何翻译多平台项目？

Weblate 支持多种文件格式（请参见支持的文件格式），最简单的方法就是使用每个平台的原生格式。

一旦在一个项目中将所有平台的翻译文件作为部件添加（请参见添加翻译项目和部件），就可以立刻使用翻译传播特性（默认打开，并且可以在部件配置 中关闭），来翻译所有平台的字符串了。

参见:

跨部件保持翻译一致

如何导出 Weblate 使用的 Git 仓库？

仓库没有什么特殊的，它存在于*DATA_DIR* 目录下面，并被命名为 `vcs/<project>/<component>/`。如果有 SSH 范文这台机器，那么可以直接使用仓库。

对于匿名访问，您会想要运行 Git 服务器，并让它为外部世界提供仓库服务。

此外，可以在 Weblate 内替代使用*Git 导出器* 而使其自动化。

将更改推送回上游的选项是什么？

这在很大程度上取决于您的设置，Weblate 在这方面非常灵活。这里是一些使用 Weblate 的工作流程的示例：

- Weblate 自动推送并合并更改（请参见如何创建自动化工作流？）。
- 您需要手动告诉 Weblate 去推送（推送需要访问上游仓库）。
- 一些人将 Weblate git 仓库的更改手动合并到上游仓库中。
- 一些人重写由 Weblate 生成的历史（例如通过删除合并提交），合并更改，并告诉 Weblate 重置上游仓库中的内容。

当然您可以按您的意愿自由混用所有这些方法。

如何限制 Weblate 仅访问翻译，而不向其暴露源代码？

可以使用 git submodule 将翻译从源代码中分离出来，而仍将它们至于版本控制之下。

1. 用翻译文件创建一个仓库。
2. 将其作为子模块添加到您的代码中：

```
git submodule add git@example.com:project-translations.git path/to/translations
```

3. 将 Weblate 连接到这个仓库上，它不再需要访问包含您源代码的仓库。
4. 可以从 Weblate 更新带有翻译的主仓库，通过：

```
git submodule update --remote path/to/translations
```

更多细节请咨询 git submodule 文档。

如何检查 Weblate 是否被正确地设置？

Weblate 包括一组配置检查，可以在管理面板中看到，只需按照管理面板中的 *Performance report* 连接，或直接打开 `/manage/performance/` URL 即可。

参见：

监测 *Weblate*, 监测 *Celery* 状态

为什么所有提交都是由 Weblate <noreply@weblate.org> 提交的？

这是默认的提交者名称，由*DEFAULT_COMMITER_EMAIL* 和*DEFAULT_COMMITER_NAME* 配置。

（如果下层的版本管理系统 VCS 支持它的话），每个提交的作者仍会被正确地记录为进行翻译的用户。

对于不知道作者身份的提交（例如匿名建议或机器翻译结果），作者身份归匿名用户所有（参见*ANONYMOUS_USER_NAME*）。您可以在管理界面中更改名称和电子邮件。

参见：

部件配置

如何移动仓库中的文件而不丢失 Weblate 中的历史记录？

要在更改文件位置后保留链接到字符串的历史记录，注释或屏幕截图，您需要确保这些字符串永远不会在 Weblate 中删除。如果 Weblate 仓库已更新，但组件配置仍指向旧文件，则可能会发生这些删除操作。这使得 Weblate 认为它应该删除所有翻译。

解决此问题的方法是与 Weblate 同步执行操作：

1. 在 Weblate 锁定受影响的部件。
2. 提交任何待处理的更改，并将它们合并到上游仓库中。
3. 禁用接收 webhook 项目配置；这可以防止 Weblate 立即看到仓库中的更改。
4. 在仓库中进行任何需要的更改（例如使用 **git mv**），将它们推送到上游仓库。
5. 更改部件配置 以匹配新设置；在更改配置时，Weblate 将获取更新的仓库并注意更改的位置，同时保留现有的字符串。
6. 解锁部件并重新启用项目配置中的挂钩。

1.9.2 用法

如何复查其他人的翻译？

- 在 Weblate 中，基于工作流程有几种复查方式，请参见翻译工作流。
- 你可以在通知 中订阅做出的任何更改，然后通过电子邮件查看其他人的贡献。
- 在翻译视图底部有个复查工具，可以在那里选择浏览给定时间以来其他人进行的翻译。

参见：

翻译工作流

如何提供源字符串的反馈？

在翻译下方的上下文选项卡中，可以通过 评论选项卡提供源字符串的反馈，或者与其他译者讨论。

参见：

report-source, 评论

翻译时如何使用现有的翻译？

- 得益于共享的翻译记忆库，Weblate 中的所有翻译都可以使用。
- 你可以将现有翻译记忆库导入 Weblate。
- 使用导入功能将概要导入作为翻译、建议或需要复查的翻译。这是使用概要或相似的翻译数据库一次性翻译的最好方法。
- 可以使用您所具有的所有数据库来设置*tmserver*。在翻译时多次使用的时候这种方法很好。
- 另一个选项是在单一 Weblate 实例中翻译所有相关项目，这样同样可以从其它项目中自动地获取翻译。

参见：

配置自动建议, 自动建议, 翻译记忆库

Weblate 除了更新翻译，还更新翻译文件吗？

Weblate 尝试将翻译文件中的更改限制为最小。对于有些文件格式，很不幸会导致将文件重新格式化。如果想要将文件保持为自己的格式化方式，请为其使用预提交钩子。

参见：

updating-target-files

语言定义来自何处以及如何添加自己的语言定义？

语言定义的基本组包括在 Weblate 和 Translate-toolkit 中。这覆盖了超过 150 种语言，并且包括了复数形式和文本方向的信息。

您可以在管理界面自由定义自己的语言，只需要提供与之相关的信息。

参见：

语言定义

Weblate 能将模糊字符串中的更改高亮吗？

Weblate 支持这个功能，然而它需要数据来显示差异。

对于 Gettext PO 文件，更新 PO 文件时，必须将参数 `--previous` 传递给 **`msgmerge`**，例如：

```
msgmerge --previous -U po/cs.po po/phpmyadmin.pot
```

对于单语言翻译，Weblate 可以通过 ID 找到之前的字符串，因此可以自动显示差异。

当已经更新了模板时，为什么 Weblate 仍然显示旧的字符串？

Weblate 除了允许译者翻译之外，不会尝试以任何方式操作翻译文件。因此当模板或源代码更改时，它也同样不会更新翻译文件。您必须简单地手动去做，并将更改推送到仓库，然后 Weblate 会自动拾取更改。

备注： 在更新翻译文件之前在 Weblate 中完成更改合并，这通常是个好主意，因为否则的话通常会以一些合并冲突来结束。

例如对于 gettext PO 文件，可以使用 **`msgmerge`** 工具来更新翻译文件：

```
msgmerge -U locale/cs/LC_MESSAGES/django.mo locale/django.pot
```

如果你想要自动更新，可以安装附加组件*更新 PO 文件以匹配 POT 文件 (msgmerge)*。

参见：

updating-target-files

1.9.3 故障排除

请求有时失败，错误信息为“too many open files”（打开文件过多）

有时当您的 Git 仓库增长太多并您有太多仓库时会发生。压缩 Git 仓库会改善这种情况。

这样做的最容易方式是运行：

```
# Go to DATA_DIR directory
cd data/vcs
# Compress all Git repositories
for d in */* ; do
    pushd $d
    git gc
    popd
done
```

参见：

DATA_DIR

当访问网站时，“Bad Request (400)”（错误的请求）错误信息

这很可能因为*ALLOWED_HOSTS* 配置不当而产生。需要包含在 Weblate 上访问的所有主机名。例如：

```
ALLOWED_HOSTS = ["weblate.example.com", "weblate", "localhost"]
```

参见：

允许主机设置

“There are more files for the single language (en)”（单一语言‘英语’有多个文件）是什么意思？

当源语言有翻译时这会典型发生。Weblate 跟踪源字符串，并为此保留源字符串。相同语言的附加字符串不会被处理。

- 如果需要对源语言进行翻译，请更改部件设置中的源语言。你也许想把 英语（开发者）用作源语言，或使用源字符串质量把关。
- 如果不需要源语言的翻译文件，请从仓库中将其删除。
- 如果需要源语言的翻译文件，但 Weblate 应该忽略它，请通过调整语言筛选 来排除它。

提示： 对于其它语言可能也会得到相似的错误信息。在这种情况下最可能的原因是在 Weblate 中几个文件映射到单语言上。

这可能是由于同时使用了过时的语言代码和新的语言代码（对于日语是 `ja` 和 `jp`），或者包含了特定国家地区的和通用的语言代码（`fr` 和 `fr_FR`）。更多细节请参见分析语言代码。

1.9.4 功能

Weblate 支持 Git 和 Mercurial 以外的其它版本控制系统（VCS）吗？

Weblate 当前不原生支持除*Git*（扩展支持*GitHub 拉取请求*、*Gerrit* 和*Subversion*）和*Mercurial* 以外的任何版本控制系统（VCS），但可以为其他版本控制系统（VCS）编写后端。

还可以在 Git 中使用*Git 远程帮助程序* 来访问其它 VCS。

Weblate 还支持无版本控制系统（VCS）的操作，请参见本地文件。

备注： 对于其它版本控制系统（VCS）的原生支持，Weblate 需要使用分布式版本控制系统（VCS），并可能会调整为能与除 Git 和 Mercurial 以外的任何东西一起使用，必须有人来实现这项支持。

参见：

版本控制集成

Weblate 如何记录译者？

Weblate 中所做的每个更改都将以译者的名称提交到版本控制系统（VCS）中。这样，每个更改都具有适当的作者身份，您可以使用用于代码的标准版本控制系统（VCS）工具来进行跟踪。

此外，如果翻译文件格式支持，将更新文件标头以包含译者的名字。

参见：

`list_translators`, ../devel/reporting

为什么 Weblate 强制在单一树中显示所有 PO 文件？

Weblate 的设计方式是将每个 PO 文件表示为单个部件。这对翻译人员来说是有好处的，这样他们就知道他们实际上在翻译什么。

在 4.2 版本发生变更: 翻译人员可以将项目的所有部件作为一个整体翻译成一种特定的语言。

Weblate 为什么使用 sr_Latn 或 zh_Hant 这样的语言代码？

这些是由 **RFC 5646** 定义的语言代码，而不是以前错误使用的修饰符（对于 `@latin` 变体）或国家地区代码（对于中文），可以更好地表明它们确实是不同的语言。

Weblate 仍然理解遗留的语言代码，并会将它们映射到当前对应的代码上——例如 `sr@latin` 将被处理为 `sr_Latn` 或者 `zh@CN` 处理为 `zh_Hans`。

备注： Weblate 默认为带有下划线的 POSIX 风格的语言代码，有关更多详细信息，请参阅语言定义。

参见：

语言定义, 语言代码风格, 添加新的翻译

1.10 支持的文件格式

Weblate 支持 translate-toolkit 理解的大多数翻译格式，但是每种格式都略有不同，一些未经充分测试的格式可能会出现问题。

参见：

与翻译相关的文件格式

备注： 为您的应用程序选择文件格式时，最好在您使用的工具包/平台中保留一些支持完善的格式。这样，您的翻译人员可以额外使用他们习惯使用的任何工具，并且更有可能为您的项目作贡献。

1.10.1 双语和单语格式

支持 单语和 双语格式。双语格式在单个文件中存储两种语言文本——原文和译文（典型示例是*GNU gettext*、*XLIFF* 或*苹果 iOS 字符串* 字符串）。另一方面，单语格式通过 ID 识别字符串，每个语言文件仅包含那些语言到任何给定语言（典型为*Android 字符串资源*）的映射。两种变体都使用某些文件格式，请参见下面的详细说明。

为了正确使用单语文件，Weblate 需要访问一个包含要翻译的字符串及其原文的完整列表的文件——该文件在 Weblate 中被称为单语言译文模版语言文件，尽管命名上和你的叫法可能会有所不同。

另外，可以利用中间语言文件 扩展此工作流程，以包括开发人员提供的字符串，但不要在最终的字符串中使用。

1.10.2 自动检测

Weblate 可以自动检测几种常用的文件格式，但是这种检测会损害您的性能，并且会限制特定于给定文件格式的功能（例如，自动添加新翻译）。

1.10.3 翻译类型功能

表 1: 所有受支持格式的功能

格式	语言能力 Page 76, 1	复数 Page 76, 2	描述 Page 76, 3	上下文 Page 76, 4	位置 Page 76, 5	标记 Page 76, 8	更多状态 Page 76, 6
GNU gettext	双语	支持	支持	支持	支持	支持[9]	需要编辑
单语 gettext	单语	支持	支持	支持	支持	支持[9]	需要编辑
XLIFF	二者	支持	支持	支持	支持	支持[10]	需要编辑、已核准
Java 属性	二者	不支持	支持	不支持	不支持	不支持	
mi18n lang 文件	单语	不支持	支持	不支持	不支持	不支持	
GWT 属性	单语	支持	支持	不支持	不支持	不支持	
Joomla 翻译	单语	不支持	支持	不支持	支持	不支持	
Qt Linguist .ts	二者	支持	支持	不支持	支持	支持 Page 76, 10	需要编辑
Android 字符串资源	单语	支持	支持[7]	不支持	不支持	支持 Page 76, 10	

续下页

表 1 – 接上页

格式	语言能力[Page 76, 1]	复数[2]	描述[3]	上下文[4]	位置[5]	标记[8]	更多状态[6]
苹果 iOS 字符串	二者	不支持	支持	不支持	不支持	不支持	
PHP 字符串	单语	不支持[11]	支持	不支持	不支持	不支持	
JSON 文件	单语	不支持	不支持	不支持	不支持	不支持	
JSON i18next 文件	单语	支持	不支持	不支持	不支持	不支持	
go-i18n JSON 文件	单语	支持	支持	不支持	不支持	不支持	
gotext JSON 文件	单语	支持	支持	不支持	支持	不支持	
ARB 文件	单语	支持	支持	不支持	不支持	不支持	
WebExtension JSON	单语	支持	支持	不支持	不支持	不支持	
.XML 资源文件	单语	不支持	支持	不支持	不支持	支持[Page 76, 10]	
ResourceDictionary 文件	单语	不支持	不支持	不支持	不支持	支持[Page 76, 10]	
CSV 文件	二者	不支持	支持	支持	支持	不支持	需要编辑
YAML 文件	单语	不支持	支持	不支持	不支持	不支持	
Ruby YAML 文件	单语	支持	支持	不支持	不支持	不支持	
DTD 文件	单语	不支持	不支持	不支持	不支持	不支持	
Flat XML 文件	单语	不支持	不支持	不支持	不支持	支持[Page 76, 10]	
Windows RC 文件	单语	不支持	支持	不支持	不支持	不支持	
Excel Open XML	单语	不支持	支持	支持	支持	不支持	需要编辑
应用商店元数据文件	单语	不支持	不支持	不支持	不支持	不支持	
字幕文件	单语	不支持	不支持	不支持	支持	不支持	
HTML 文件	单语	不支持	不支持	不支持	不支持	不支持	
OpenDocument 格式	单语	不支持	不支持	不支持	不支持	不支持	
IDML 格式	单语	不支持	不支持	不支持	不支持	不支持	
INI 翻译	单语	不支持	不支持	不支持	不支持	不支持	
Inno Setup INI 翻译	单语	不支持	不支持	不支持	不支持	不支持	
TermBase eXchange 格式	双语	不支持	支持	不支持	不支持	支持[Page 76, 10]	
文本文件	单语	不支持	不支持	不支持	不支持	不支持	
Stringsdict 格式	单语	支持	支持	不支持	不支持	不支持	

续下页

表 1 – 接上页

格式	语言能力[Page 76, 1]	复数[2]	描述[3]	上下文[4]	位置[5]	标记[8]	更多状态[6]
Fluent 格式	单语	不支持[12]	支持	不支持	不支持	不支持	

只读字符串

在 3.10 版本加入.

翻译文件中的只读字符串将被包含在 Weblate 中，但不能在 Weblate 中对其进行编辑。只有少数格式（*XLIFF* 和*Android 字符串资源*）原生支持此功能，但可以通过添加 `read-only` 标记在其他格式中进行模拟，请参阅使用标记定制行为。

1.10.4 GNU gettext

翻译自由软件用得最广泛的格式。

支持通过调整文件标头或链接到相应的源文件，在文件中存储上下文信息。

双语 gettext PO 文件通常如下所示：

```
#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "Monday"
msgstr "Pondělí"

#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "Tuesday"
msgstr "Úterý"

#: weblate/accounts/avatar.py:163
msgctxt "No known user"
msgid "None"
msgstr "Žádný"
```

典型的 Weblate 部件配置	
文件掩码	`po/*.po`
单语言译文模版语言文件	空
新翻译的翻译模版	`po/messages.pot`
文件格式	*Gettext PO* 文件

参见：

devel/gettext，devel/sphinx，维基百科上的 Gettext（英文），PO Files，更新“配置文件”中的 *ALL_LINGUAS* 变量，自定义 *gettext* 输出，更新 *LINGUAS* 文件，生成 *MO* 文件，更新 *PO* 文件以匹配 *POT* 文件 *(msgmerge)*

[1] 请参见双语和单语格式
[2] 复数是正确本地化具有可变计数的字符串所必需的。
[3] 源字符串描述可用于传递要翻译的字符串的附加信息。
[4] 上下文用于区分在不同领域使用的相同字符串（例如 *Sun* 可以是“Sunday”的缩写，也可以是离我们最近的恒星的名字）。
[5] 字符串在源代码中的位置可能会帮助经验丰富的译者弄清楚字符串是如何使用的。
[8] 请参见使用标记定制行为
[6] 除“Untranslated”（未翻译）和“Translated”（已翻译）之外，文件格式支持的更多状态。
[9] gettext 类型的注释用作标记。
[10] 对于所有基于 XML 的格式，标记是从非标准属性 `weblate-flags` 中提取的。此外，还通过 XLIFF 标准中定义的 `maxwidth` 属性 支持了 `max-length:N`，请参阅指定翻译标记。
[7] 放在 `<string>` 元素前的 XML 注释，解析为源字符串描述。
[11] 只有 Laravel 支持复数，它使用字符串语法来定义复数，请参见 Localization in Laravel。
[12] 复数在字符串的语法中处理，而不是在 Weblate 中公开为复数。

单语 gettext

一些项目决定使用 gettext 作为单语格式——它们仅在源代码中编码 ID，然后将字符串翻译成所有语言，包括英语。支持此功能，尽管在将部件导入 Weblate 时必须明确选择此文件格式。

单语言的 gettext PO 文件通常如下所示：

```
#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "day-monday"
msgstr "Pondělí"

#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "day-tuesday"
msgstr "Úterý"

#: weblate/accounts/avatar.py:163
msgid "none-user"
msgstr "Žádný"
```

基本语言文件将是：

```
#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "day-monday"
msgstr "Monday"

#: weblate/media/js/bootstrap-datepicker.js:1421
msgid "day-tuesday"
msgstr "Tuesday"

#: weblate/accounts/avatar.py:163
msgid "none-user"
msgstr "None"
```

典型的 Weblate 部件配置	
文件掩码	`po/*.po`
单语言译文模版语言文件	`po/en.po`
新翻译的翻译模版	`po/messages.pot`
文件格式	*Gettext PO* 文件（单语）

1.10.5 XLIFF

基于 XML 的格式，为标准化翻译文件而生，但最终它也只是这个领域的 众多标准 之一。

XML Localization Interchange File Format (XLIFF) 通常用作双语言格式，但 Weblate 也支持其作为单语言格式。

Weblate 支持数种变体的 XLIFF：

***XLIFF* 翻译文件**
: 简单的 XLIFF 文件，其中元素的内容存储为纯文本（所有 XML 元素都被转义）。

支持可放置对象的 *XLIFF*
: 标准 XLIFF 支持可放置对象和其他 XML 元素。

带 *gettext* 扩展的 *XLIFF*
: XLIFF 使用 XLIFF 1.2 Gettext PO 表示指南 进行丰富，以支持复数。

参见：

XML Localization Interchange File Format (XLIFF) 规范，XLIFF 1.2 Gettext PO 表示指南

翻译状态

在 3.3 版本发生变更: 3.3 版本前的 Weblate 忽略 `state` 属性。

文件中的 `state` 属性在 Weblate 中被部分处理并映射为 “Needs edit”（需要编辑）的状态（如果出现目标的话，后面的状态用于将字符串标记为需要编辑：`new` 、`needs-translation` 、`needs-adaptation` 、`needs-l10n` ）。如果应该省略 `state` 属性，只要 `<target>` 元素存在，那么字符串被认为需要翻译。

如果翻译字符串具有 `approved="yes"`，那么它还将作为 “Approved”（已核准）导入 Weblate，其他的将作为 “Waiting for review”（待审校）导入（符合 XLIFF 规范）。

当存储时，Weblate 如无必要不会添加其它属性：

- `state` 属性只在字符串标记为需要编辑的情况下添加。
- `approved` 属性只在字符串已审校的情况下添加。
- 在其它情况下不添加属性，但在它们出现的情况下更新。

这意味着在使用 XLIFF 格式时，强烈推荐打开 Weblate 审校过程，以便查看和更改字符串的核准状态。

相似地在导入这样的文件时（在上传表单中），应该选择 *处理需要编辑的字符串之下的* *导入为已翻译*。

参见:

专门的审校员

XLIFF 中的空格和换行符

通常空格的类型和数量和在 XML 格式中的一样对待。如果想要保留，必须将 `xml:space="preserve"` 标记添加到字符串中。

例如：

```
    <trans-unit id="10" approved="yes">
        <source xml:space="preserve">hello</source>
        <target xml:space="preserve">Hello, world!
</target>
    </trans-unit>
```

指定翻译标记

还可以通过使用 `weblate-flags` 属性指定附加的翻译标记（请参见使用标记定制行为）。Weblate 还理解来自 XLIFF 规范的 `maxwidth` 和 `font` 属性：

```
<trans-unit id="10" maxwidth="100" size-unit="pixel" font="ubuntu;22;bold">
   <source>Hello %s</source>
</trans-unit>
<trans-unit id="20" maxwidth="100" size-unit="char" weblate-flags="c-format">
   <source>Hello %s</source>
</trans-unit>
```

解析 `font` 属性用于字体集、尺寸和粗细，上面的示例显示了全部，尽管只需要字符集。字符集中的任何空格被转换为下划线，所以 `Source Sans Pro` 变成 `Source_Sans_Pro`，当命名字符集时请记住这些请参见管理字型）。

字符串键

Weblate 通过 resname 属性（如果有的话）识别 XLIFF 文件中的单元，并会回退到 id（如果有的话，则与 file 标签一起）。

resname 属性被认为是人类友好的单元标识符，这也使其比 id 更适合 Weblate 显示。在整个 XLIFF 文件中 resname 必须是唯一的。这是 Weblate 的要求，XLIFF 标准没有涉及到这一点——它没有对此属性施加任何唯一性限制。

用于双语言 XLIFF 的典型 Weblate 部件配置	
文件掩码	localizations/*.xliff
单语言译文模版语言文件	空
新翻译的翻译模版	localizations/en-US.xliff
文件格式	*XLIFF 翻译文件*

用于单语言 XLIFF 的典型 Weblate 部件配置	
文件掩码	localizations/*.xliff
单语言译文模版语言文件	localizations/en-US.xliff
新翻译的翻译模版	localizations/en-US.xliff
文件格式	*XLIFF 翻译文件*

参见：

维基百科上的 XLIFF（英文），XLIFF，XLIFF 1.2 中的字体属性，XLIFF 1.2 中的 maxwidth 属性

1.10.6 Java 属性

Java 原生翻译格式。

Java 属性通常用作单语言翻译。

Weblate 支持这个格式的 ISO-8859-1、UTF-8 和 UTF-16 变体。它们所有都支持存储 Unicode 字符，只是编码不同。在 ISO-8859-1 中，使用了 Unicode 转义序列（例如 zkou\u0161ka），所有其它编码字符直接或者在 UTF-8 中或者在 UTF-16 中。

备注： 加载转义序列也在 UTF-8 模式中工作，因此请小心选择正确的编码组，与您应用的需要匹配。

典型的 Weblate 部件配置	
文件掩码	src/app/Bundle_*.properties
单语言译文模版语言文件	src/app/Bundle.properties
新翻译的翻译模版	空
文件格式	*Java 属性 (ISO-8859-1)*

参见：

维基百科上的 Java 属性（英文），Mozilla 和 Java 属性文件，*mi18n lang 文件*，*GWT 属性*，updating-target-files，格式化 *Java 属性文件*，清理翻译文件

1.10.7 mi18n lang 文件

在 4.7 版本加入.

mi18n 用于 JavaScript 本地化的文件格式。在语法上它匹配*Java 属性*。

典型的 Weblate 部件配置	
文件掩码	`*.lang`
单语言译文模版语言文件	`en-US.lang`
新翻译的翻译模版	空
文件格式	*mi18n lang 文件*

参见:

mi18n，Mozilla 和 Java 属性文件，*Java 属性*，updating-target-files，格式化 *Java 属性文件*，清理翻译文件

1.10.8 GWT 属性

GWT 原生翻译格式。

GWT 属性通常用作单语言翻译。

典型的 Weblate 部件配置	
文件掩码	`src/app/Bundle_*.properties`
单语言译文模版语言文件	`src/app/Bundle.properties`
新翻译的翻译模版	空
文件格式	*GWT 属性*

参见:

GWT 本地化指南，GWT 国际化教程，Mozilla 和 Java 属性文件，updating-target-files，格式化 *Java 属性文件*，清理翻译文件

1.10.9 INI 翻译

在 4.1 版本加入.

用于翻译的 INI 文件格式。

INI 翻译通常用作单语言翻译。

典型的 Weblate 部件配置	
文件掩码	`language/*.ini`
单语言译文模版语言文件	`language/en.ini`
新翻译的翻译模版	空
文件格式	*INI 文件*

备注: Weblate 只从 INI 文件内的章节中提取键。在您的 INI 文件缺乏章节的情况下，会想要替代使用*Joomla 翻译* 或*Java 属性*。

参见:

INI 文件, *Java 属性*, *Joomla 翻译*, *Inno Setup INI 翻译*

1.10.10 Inno Setup INI 翻译

在 4.1 版本加入.

用于翻译的 Inno Setup INI 文件格式。

Inno Setup INI 翻译，通常用作单语言翻译。

备注： 唯一需要注意的差异是*INI 翻译* 支持 `%n` 和 `%t` 占位符用于换行和制表符。

典型的 Weblate 部件配置	
文件掩码	`language/*.islu`
单语言译文模版语言文件	`language/en.islu`
新翻译的翻译模版	空
文件格式	*Inno Setup INI 文件*

备注： 当前只支持 Unicode 文件（`.islu`），当前不支持 ANSI 变体（`.isl`）。

参见：

INI Files, *Joomla 翻译*, *INI 翻译*

1.10.11 Joomla 翻译

在 2.12 版本加入.

Joomla 原生翻译格式。

Joomla 翻译通常用作单语言翻译。

典型的 Weblate 部件配置	
文件掩码	`language/*/com_foobar.ini`
单语言译文模版语言文件	`language/en-GB/com_foobar.ini`
新翻译的翻译模版	空
文件格式	*Joomla 语言文件*

参见：

Mozilla and Java properties files, *INI 翻译*, *Inno Setup INI 翻译*

1.10.12 Qt Linguist .ts

基于 Qt 的应用中使用的翻译格式。

Qt Linguist 文件既用作双语翻译，也用作单语翻译。

用作双语言时典型的 Weblate 部件配置	
文件掩码	`i18n/app.*.ts`
单语言译文模版语言文件	空
新翻译的翻译模版	`i18n/app.de.ts`
文件格式	*Qt Linguist 翻译文件*

用作单语言时典型的 Weblate 部件配置	
文件掩码	`i18n/app.*.ts`
单语言译文模版语言文件	`i18n/app.en.ts`
新翻译的翻译模版	`i18n/app.en.ts`
文件格式	*Qt Linguist* 翻译文件

参见：

Qt Linguist 手册, Qt .ts, 双语和单语格式

1.10.13 Android 字符串资源

用于翻译应用的 Android 特定文件格式。

Android 字符串资源是单语言的，单语言译文模版语言文件 存储在与其他文件不同的位置–`res/values/strings.xml`。

典型的 Weblate 部件配置	
文件掩码	`res/values-*/strings.xml`
单语言译文模版语言文件	`res/values/strings.xml`
新翻译的翻译模版	空
文件格式	*Android* 字符串资源

参见：

Android 字符串资源文档 ， Android 字符串资源

备注： 当前不支持 Android 的 *string-array* 架构。为了解决这个问题，可以将字符串数组分开：

```
<string-array name="several_strings">
    <item>First string</item>
    <item>Second string</item>
</string-array>
```

变为：

```
<string-array name="several_strings">
    <item>@string/several_strings_0</item>
    <item>@string/several_strings_1</item>
</string-array>
<string name="several_strings_0">First string</string>
<string name="several_strings_1">Second string</string>
```

指向 *string* 元素的 *string-array* 应存储在不同文件中，并且不为翻译所用。

这个脚本可以帮助预处理现有的 strings.xml 文件和翻译：https://gist.github.com/paour/11291062

提示： 为了避免翻译某些字符串，可以将它们标记为不可翻译。这对于字符串引用特别有用：

```
<string name="foobar" translatable="false">@string/foo</string>
```

1.10.14 苹果 iOS 字符串

通常用于翻译 Apple 的文件格式 iOS 应用程序，但也由 PWG 5100.13 标准化并用于 NeXTSTEP/OpenSTEP。苹果 iOS 字符串通常用作单语言。

典型的 Weblate 部件配置	
文件掩码	`Resources/*.lproj/Localizable.strings`
单语言译文模版语言文件	`Resources/en.lproj/Localizable.strings` 或 `Resources/Base.lproj/Localizable.strings`
新翻译的翻译模版	空
文件格式	*iOS 字符串 (UTF-8)*

参见：

Stringsdict 格式, Apple “字符串文件” 文档, 消息目录文件 PWG 5100.13 中的格式, Mac OSX strings

1.10.15 PHP 字符串

PHP 翻译通常只包含一种语言，因此建议指定 (最常见的) 带英语字符串的模板文件。

示例文件：

```
<?php
$LANG['foo'] = 'bar';
$LANG['foo1'] = 'foo bar';
$LANG['foo2'] = 'foo bar baz';
$LANG['foo3'] = 'foo bar baz bag';
```

典型的 Weblate 部件配置	
文件掩码	`lang/*/texts.php`
单语言译文模版语言文件	`lang/en/texts.php`
新翻译的翻译模版	`lang/en/texts.php`
文件格式	*PHP 字符串*

Laravel PHP 字符串

在 4.1 版本发生变更.

Laravel PHP 本地化文件也支持复数：

```
<?php
return [
    'welcome' => 'Welcome to our application',
    'apples' => 'There is one apple|There are many apples',
];
```

参见：

PHP, Laravel 上的本地化

1.10.16 JSON 文件

在 2.0 版本加入.

在 2.16 版本发生变更: 自从 Weblate 2.16 和最早 2.2.4 版本的 translate-toolkit，也支持嵌套结构的 JSON 文件。

在 4.3 版本发生变更: 即使对于在之前发布版本中中断的复杂情况，JSON 文件的结构也适当保留。

JSON 格式主要用于翻译用 JavaScript 实现的应用程序。

Weblate 目前支持 JSON 翻译的几种变体:

- 简单的键/值文件，由例如 *vue-i18n* 或 *react-intl* 使用。
- 具有嵌套键的文件。
- *JSON i18next 文件*
- *go-i18n JSON 文件*
- *gotext JSON 文件*
- *WebExtension JSON*
- *ARB 文件*

JSON 翻译通常是单语言的，因此推荐指定带有（最经常使用的）英语字符串的翻译模板文件。

示例文件:

```
{
  "Hello, world!\n": "Ahoj světe!\n",
  "Orangutan has %d banana.\n": "",
  "Try Weblate at https://demo.weblate.org/!\n": "",
  "Thank you for using Weblate.": ""
}
```

也支持嵌套文件（要求请参见上面），这样的文件看起来像:

```
{
  "weblate": {
    "hello": "Ahoj světe!\n",
    "orangutan": "",
    "try": "",
    "thanks": ""
  }
}
```

提示: *JSON 文件*和 *JSON 嵌套结构文件*都可以处理相同类型的文件。翻译时都保留现有的 JSON 架构。

它们之间的唯一区别是使用 Weblate 添加新字符串时。嵌套结构格式解析新添加的键并将新字符串插入到匹配结构中。例如 `app.name` 键被插入为:

```
{
   "app": {
      "name": "Weblate"
   }
}
```

典型的 Weblate 部件配置	
文件掩码	`langs/translation-*.json`
单语言译文模版语言文件	`langs/translation-en.json`
新翻译的翻译模版	空
文件格式	*JSON 嵌套结构文件*

参见:

JSON, updating-target-files, 自定义 *JSON* 输出, 清理翻译文件,

1.10.17 JSON i18next 文件

在 2.17 版本发生变更: 从 Weblate 2.17 和 translate-toolkit 至少 2.2.5 版本开始，也支持带有复数的 i18next JSON 文件。

在 4.15.1 版本发生变更: 添加了对此格式 v4 变体的支持。

提示: 如果你使用复数形式，那么建议你使用 v4 版格式，因其在复数形式处理方面对齐 CLDR。旧版格式对某些语言有不同的复数规则，这些规则是错误的。

i18next 是一个用 JavaScript 编写的国际化框架。Weblate 支持其具有复数等功能的本地化文件。

i18next 翻译是单语的，因此建议使用英语字符串（最常见的是英语字符串）指定基本文件。

备注: Weblate 支持 i18next JSON v3 和 v4 变体。请选择匹配你环境的正确文件格式。

v2 和 v1 变体大体兼容 v3，但复数的处理方式除外。

示例文件:

```
{
  "hello": "Hello",
  "apple": "I have an apple",
  "apple_plural": "I have {{count}} apples",
  "apple_negative": "I have no apples"
}
```

典型的 Weblate 部件配置	
文件掩码	`langs/*.json`
单语言译文模版语言文件	`langs/en.json`
新翻译的翻译模版	空
文件格式	*i18next JSON file v3*

参见:

JSON, i18next JSON 格式, updating-target-files, 自定义 *JSON* 输出, 清理翻译文件

1.10.18 go-i18n JSON 文件

在 4.1 版本加入.

在 4.16 版本发生变更: 添加了对此格式 v2 变体的支持。

go-i18n 翻译是单语言的因此建议使用（最常见的）英文字符串指定一个基本文件。

备注： Weblate 支持 go-i18n JSON v1 和 v2 变体。请选择匹配你环境的正确文件格式。

典型的 Weblate 部件配置 for v1	
文件掩码	`langs/*.json`
单语言译文模版语言文件	`langs/en.json`
新翻译的翻译模版	空
文件格式	*go-i18n v1 JSON* 文件

典型的 Weblate 部件配置 for v2	
文件掩码	`langs/*.json`
单语言译文模版语言文件	`langs/en.json`
新翻译的翻译模版	空
文件格式	*go-i18n v2 JSON* 文件

参见:

JSON, go-i18n, updating-target-files, 自定义 *JSON* 输出, 清理翻译文件,

1.10.19 gotext JSON 文件

在 4.15.1 版本加入.

gotext 翻译是单语的，因此建议使用英语字符串（最常见的是英语字符串）指定基本文件。

典型的 Weblate 部件配置	
文件掩码	`internal/translations/locales/*/messages.gotext.json`
单语言译文模版语言文件	`internal/translations/locales/en-GB/messages.gotext.json`
新翻译的翻译模版	空
文件格式	*gotext JSON* 文件

参见:

JSON, I18n in Go: Managing Translations, updating-target-files, 自定义 *JSON* 输出, 清理翻译文件,

1.10.20 ARB 文件

在 4.1 版本加入.

ARB 翻译是单语言的，因此建议使用（通常是）英文字符串指定一个基本文件。

典型的 Weblate 部件配置	
文件掩码	`lib/l10n/intl_*.arb`
单语言译文模版语言文件	`lib/l10n/intl_en.arb`
新翻译的翻译模版	空
文件格式	*ARB 文件*

参见:

JSON, **‘应用程序资源包 <https://github.com/google/app-resource-bundle/wiki/ApplicationResourceBundleSpecification>‘_**, **‘国际化 Flutter 应用程序 <https://docs.flutter.dev/development/accessibility-and-localization/internationalization>‘_**, updating-target-files, 自定义 *JSON* 输出, 清理翻译文件

1.10.21 WebExtension JSON

在 2.16 版本加入: 从 Weblate 2.16 开始支持此功能，并且使用 ‘translate-toolkit‘_ 至少 2.2.4。

翻译 Mozilla Firefox 或 Google Chromium 的扩展时使用的文件格式。

备注: 虽然这种格式称为 JSON，但其规范允许包含注释，这不是 JSON 规范的一部分。Weblate 当前不支持带注释的文件。

示例文件:

```
{
  "hello": {
    "message": "Ahoj světe!\n",
    "description": "Description",
    "placeholders": {
      "url": {
        "content": "$1",
        "example": "https://developer.mozilla.org"
      }
    }
  },
  "orangutan": {
    "message": "Orangutan has $coUnT$ bananas",
    "description": "Description",
    "placeholders": {
      "count": {
        "content": "$1",
        "example": "5"
      }
    }
  },
  "try": {
    "message": "",
    "description": "Description"
  },
  "thanks": {
    "message": "",
    "description": "Description"
```

(续下页)

(接上页)

```
  }
}
```

典型的 Weblate 部件配置	
文件掩码	`_locales/*/messages.json`
单语言译文模版语言文件	`_locales/en/messages.json`
新翻译的翻译模版	空
文件格式	*WebExtension JSON* 文件

参见：

JSON，Google chrome.i18n，**‘Mozilla 扩展国际化 <https://developer.mozilla.org/zh-CN/docs/Mozilla/Add-ons/WebExtensions/Internationalization >‘_**

1.10.22 .XML 资源文件

在 2.3 版本加入.

.XML 资源文件（.resx）采用了 Microsoft .NET 应用程序中使用的单语 XML 文件格式。当使用与.resx 相同的语法时，它可以与.resw 互换。

典型的 Weblate 部件配置	
文件掩码	`Resources/Language.*.resx`
单语言译文模版语言文件	`Resources/Language.resx`
新翻译的翻译模版	空
文件格式	*.NET* 资源文件

参见：

.NET 资源文件 (.resx), updating-target-files, 清理翻译文件

1.10.23 ResourceDictionary 文件

在 4.13 版本加入.

ResourceDictionary 是一种单语 XML 文件格式，用于为 Windows Presentation Foundation (WPF) 应用程序打包可本地化的字符串资源。

典型的 Weblate 部件配置	
文件掩码	`Languages/*.xaml`
单语言译文模版语言文件	`Language/en.xaml`
新翻译的翻译模版	空
文件格式	*ResourceDictionary* 文件

参见：

Flat XML, *Flat XML* 文件, updating-target-files, 清理翻译文件

1.10.24 CSV 文件

在 2.4 版本加入.

CSV 文件可以包含一个原文和译文的简单列表。Weblate 支持以下文件：

- 带有标头定义字段（`location`, `source`, `target`, `ID`, `fuzzy`, `context`, `translator_comments`, `developer_comments`）的文件。这是推荐的方法，因为它最不容易出错。挑选 *CSV file* 作为一种文件格式。
- 具有两个字段的文件——原文和译文（按此顺序），选择 简单 *CSV* 文件作为文件格式。
- 无标头文件，其字段按 translate-toolkit 定义的顺序排列：`location`、`source`、`target`、`ID`、`fuzzy`、`context`、`translator_comments`、`developer_comments`。选择 *CSV* 文件作为文件格式。
- 当你的文件是单语言时记得定义单语言译文模版语言文件 （参见双语和单语格式 ）。

提示： 默认情况下，CSV 格式会自动检测文件编码。这在某些极端情况下可能不可靠，并导致性能下降。请选择带有编码的文件格式变体以避免这种情况（例如 *CSV* 文件 *(UTF-8)* ）。

警告： CSV 格式当前会自动检测 CSV 文件的方言。在某些情况下，自动检测可能会失败，您会得到不同的结果。对于值中包含换行符的 CSV 文件尤其如此。解决方法是建议省略引号字符。

示例文件：

```
Thank you for using Weblate.,Děkujeme za použití Weblate.
```

双语 CSV 文件的典型 Weblate 部件配置	
文件掩码	`locale/*.csv`
单语言译文模版语言文件	空
新翻译的翻译模版	`locale/en.csv`
文件格式	*CSV* 文件

单语 CSV 文件的典型 Weblate 部件配置	
文件掩码	`locale/*.csv`
单语言译文模版语言文件	`locale/en.csv`
新翻译的翻译模版	`locale/en.csv`
文件格式	简单 *CSV* 文件

多值 CSV 文件

在 4.13 版本加入.

CSV 文件的这种变体允许每条字符串存储多个翻译。

参见：

CSV

1.10.25 YAML 文件

在 2.9 版本加入.

带有字符串键和值的纯 YAML 文件。Weblate 还从列表或字典中提取字符串。

YAML 文件示例：

```
weblate:
  hello: ""
  orangutan": ""
  try": ""
  thanks": ""
```

典型的 Weblate 部件配置	
文件掩码	`translations/messages.*.yml`
单语言译文模版语言文件	`translations/messages.en.yml`
新翻译的翻译模版	空
文件格式	*YAML 文件*

参见：

YAML, *Ruby YAML 文件*

1.10.26 Ruby YAML 文件

在 2.9 版本加入.

以语言为根节点的 Ruby i18n YAML 文件。

Ruby i18n YAML 文件示例：

```
cs:
  weblate:
    hello: ""
    orangutan: ""
    try: ""
    thanks: ""
```

典型的 Weblate 部件配置	
文件掩码	`translations/messages.*.yml`
单语言译文模版语言文件	`translations/messages.en.yml`
新翻译的翻译模版	空
文件格式	*Ruby YAML 文件*

参见：

YAML, *YAML 文件*

1.10.27 DTD 文件

在 2.18 版本加入.

DTD 文件示例：

```
<!ENTITY hello "">
<!ENTITY orangutan "">
<!ENTITY try "">
<!ENTITY thanks "">
```

典型的 Weblate 部件配置	
文件掩码	`locale/*.dtd`
单语言译文模版语言文件	`locale/en.dtd`
新翻译的翻译模版	空
文件格式	*DTD 文件*

参见：

Mozilla DTD format

1.10.28 Flat XML 文件

在 3.9 版本加入.

flat XML 文件示例：

```
<?xml version='1.0' encoding='UTF-8'?>
<root>
  <str key="hello_world">Hello World!</str>
  <str key="resource_key">Translated value.</str>
</root>
```

典型的 Weblate 部件配置	
文件掩码	`locale/*.xml`
单语言译文模版语言文件	`locale/en.xml`
新翻译的翻译模版	空
文件格式	*Flat XML 文件*

参见：

Flat XML

1.10.29 Windows RC 文件

在 4.1 版本发生变更: 对 Windows RC 文件的支持已被重写。

备注： 对这种格式的支持目前处于测试阶段，欢迎测试反馈。

Windows RC 文件示例：

```
LANGUAGE LANG_CZECH, SUBLANG_DEFAULT

STRINGTABLE
BEGIN
    IDS_MSG1                "Hello, world!\n"
    IDS_MSG2                "Orangutan has %d banana.\n"
    IDS_MSG3                "Try Weblate at http://demo.weblate.org/!\n"
    IDS_MSG4                "Thank you for using Weblate."
END
```

典型的 Weblate 部件配置	
文件掩码	`lang/*.rc`
单语言译文模版语言文件	`lang/en-US.rc`
新翻译的翻译模版	`lang/en-US.rc`
文件格式	*RC* 文件

参见:

Windows RC files

1.10.30 应用商店元数据文件

在 3.5 版本加入.

用于在各种应用商店发布应用的元数据可以被翻译目前兼容以下工具:

- Triple-T gradle-play-publisher
- Fastlane
- F-Droid

元数据由几个文本文件组成，Weblate 将把这些文本文件作为单独的字符串进行翻译。

典型的 Weblate 部件配置	
文件掩码	`fastlane/android/metadata/*`
单语言译文模版语言文件	`fastlane/android/metadata/en-US`
新翻译的翻译模版	`fastlane/android/metadata/en-US`
文件格式	应用商店元数据文件

提示: 如果您不想翻译某些字符串（例如变更日志）请将它们标记为只读（参见使用标记定制行为）这可以通过批量编辑 自动完成。

1.10.31 字幕文件

在 3.7 版本加入.

Weblate 可以翻译各种字幕文件:

- SubRip 字幕文件 (`*.srt`)
- MicroDVD 字幕文件 (`*.sub`)
- Advanced SubStation Alpha 字幕文件 (`*.ass`)
- SubStation Alpha 字幕文件 (`*.ssa`)

典型的 Weblate 部件配置	
文件掩码	`path/*.srt`
单语言译文模版语言文件	`path/en.srt`
新翻译的翻译模版	`path/en.srt`
文件格式	*SubRip* 字幕文件

参见:

Subtitles

1.10.32 Excel Open XML

在 3.2 版本加入.

可以导入和导出 Excel Open XML (.xlsx) 文件。

上传 XLSX 文件进行翻译时，请注意只考虑活动工作表，并且必须至少有一个名为 `source` （包含源字符串）的列和一个名为 `target` （包含翻译）的列此外应该有一个名为的列 `context` （其中包含翻译字符串的上下文路径）。如果您使用 XLSX 下载将翻译导出到 Excel 工作簿，您已经获得了具有正确文件格式的文件。

1.10.33 HTML 文件

在 4.1 版本加入.

备注: 对这种格式的支持目前处于测试阶段，欢迎测试反馈。

从 HTML 文件中提取可翻译内容并提供翻译。

参见:

HTML

1.10.34 文本文件

在 4.6 版本加入.

备注: 对这种格式的支持目前处于测试阶段，欢迎测试反馈。

可翻译的内容从纯文本文件中提取出来并提供给翻译。每个段落都被翻译成一个单独的字符串。

这种格式有三种风格:

- 纯文本文件
- DokuWiki 文本文件
- MediaWiki 文本文件

参见:

Simple Text Documents

1.10.35 OpenDocument 格式

在 4.1 版本加入.

备注： 对这种格式的支持目前处于测试阶段，欢迎测试反馈。

可翻译的内容从 OpenDocument 文件中提取出来并提供给翻译。

参见：

OpenDocument Format

1.10.36 IDML 格式

在 4.1 版本加入.

备注： 对这种格式的支持目前处于测试阶段，欢迎测试反馈。

可翻译内容从 Adobe InDesign 标记语言文件中提取并提供翻译。

1.10.37 TermBase eXchange 格式

在 4.5 版本加入.

TBX 是一种用于交换术语数据的 XML 格式。

典型的 Weblate 部件配置	
文件掩码	`tbx/*.tbx`
单语言译文模版语言文件	空
新翻译的翻译模版	空
文件格式	*TermBase eXchange* 文件

参见：

维基百科上的 TBX（英文），TBX，术语表

1.10.38 Stringsdict 格式

在 4.8 版本加入.

备注： 对这种格式的支持目前处于测试阶段，欢迎测试反馈。

Apple 使用的基于 XML 的格式能够存储字符串的复数形式。

典型的 Weblate 部件配置	
文件掩码	`Resources/*.lproj/Localizable.stringsdict`
单语言译文模版语言文件	`Resources/en.lproj/Localizable.stringsdict` 或 `Resources/Base.lproj/Localizable.stringsdict`
新翻译的翻译模版	空
文件格式	*Stringsdict* 文件

参见：

苹果 *iOS* 字符串, Stringsdict 文件格式

1.10.39 Fluent 格式

在 4.8 版本加入.

备注： 对这种格式的支持目前处于测试阶段，欢迎测试反馈。

Fluent 是一种专注于非对称本地化的单语文本格式：一种语言的简单字符串可以映射到另一种语言的复杂多变体翻译。

典型的 Weblate 部件配置	
文件掩码	`locales/*/messages.ftl`
单语言译文模版语言文件	`locales/en/messages.ftl`
新翻译的翻译模版	空
文件格式	*Fluent* 文件

参见：

Fluent 项目网站

1.10.40 支持其他格式

translate-toolkit 支持的大多数支持序列化的格式都可以轻松支持，但它们（尚未）接受任何测试。在大多数情况下，Weblate 需要一些薄层来隐藏不同 translate-toolkit 存储的行为差异。

要添加对新格式的支持，首选方法是首先在 translate-toolkit 中实现对它的支持。

参见：

与翻译相关的文件格式

1.11 版本控制集成

Weblate 当前支持*Git* （扩展支持*GitHub 拉取请求*, *GitLab 合并请求*, *Gitea 拉取请求*, *Gerrit*, *Subversion* 及*Bitbucket 服务器拉取请求*）和*Mercurial* 作为版本控制后端。

1.11.1 访问仓库

您要使用的版本控制系统（VCS）仓库必须可供 Weblate 访问。对于公开可用的仓库，您只需要输入正确的 URL（例如 `https://github.com/WeblateOrg/weblate.git`），但对于私有仓库或推送 URL，设置更加复杂并且需要验证。

从 Hosted Weblate 访问仓库

对于 Hosted Weblate，有一个在 GitHub、Bitbucket、Codeberg 和 GitLab 上注册的专用推送用户（用户名 *weblate*，电子邮件 `hosted@weblate.org` 和，名为 *Weblate 推送用户*）。您需要将此用户添加为协作者并为其授予对您的仓库的适当权限（克隆可以只读，推送需要写入）。根据服务和您的组织设置，这会立即发生，或者需要在 Weblate 端进行确认。

GitHub 上的 *weblate* 用户会在五分钟内自动接受邀请。其他服务可能需要人工处理，请耐心等待。

一旦 *weblate* 用户被添加，你可以使用 SSH 协议（例如 `git@github.com:WeblateOrg/weblate.git`）配置源代码仓库 和仓库推送 *URL* 。

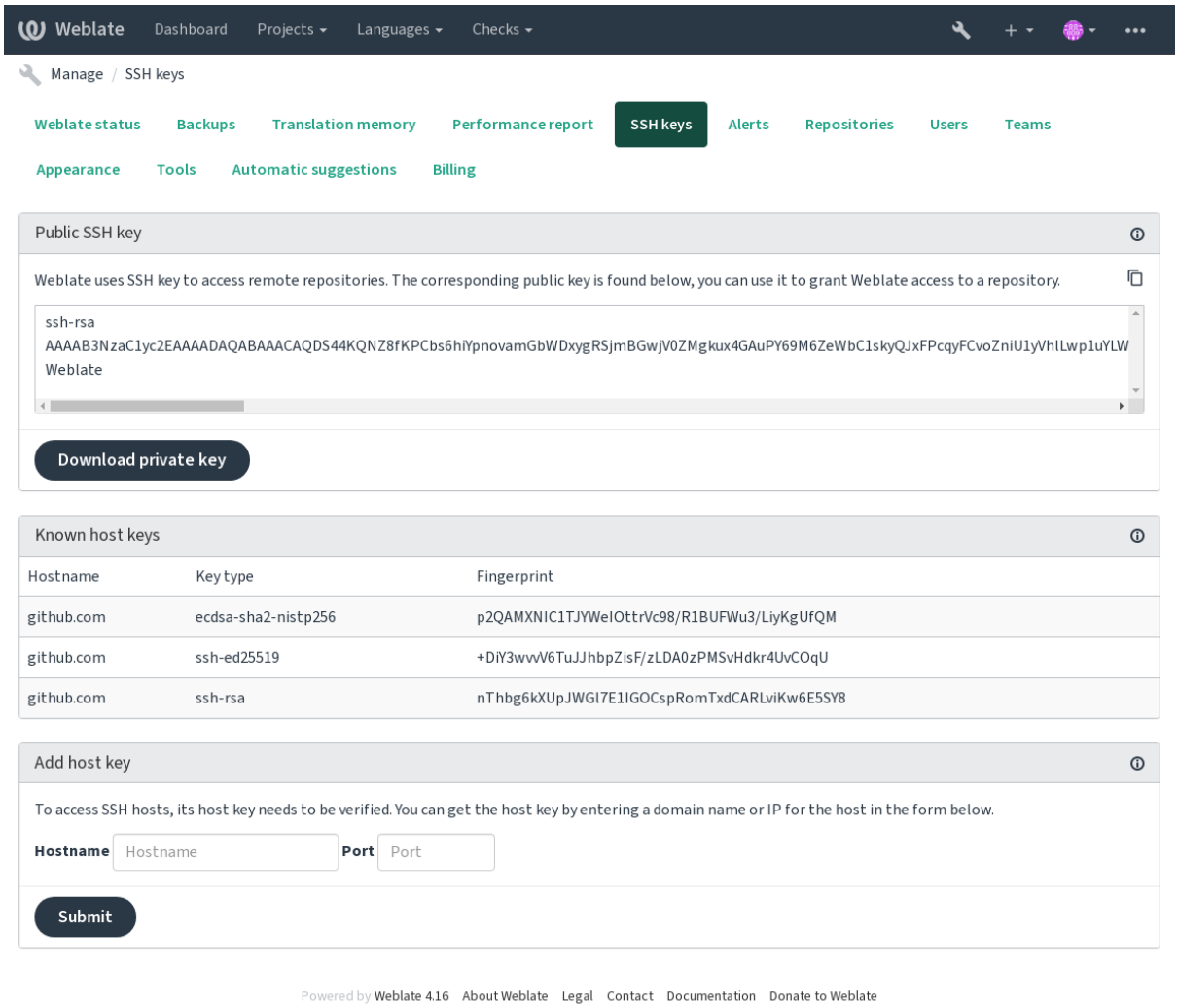
SSH 仓库

访问私有仓库的最常用方法是基于 SSH。授权公共 Weblate SSH 密钥（请参阅*Weblate SSH 密钥*）以这种方式访问上游仓库。

警告： 在 Github 上，每个密钥只能用一次，见*GitHub 仓库* 和*从 Hosted Weblate 访问仓库*。

Weblate 还会在首次连接时存储主机密钥指纹，并且在以后进行更改时将无法连接到主机（请参阅*验证 SSH 主机密钥*）。

如果需要调整，请从 Weblate 管理界面进行：



Weblate SSH 密钥

Weblate 公钥对浏览 关于页面的所有用户可见。

管理员可以在管理界面着陆页的连接部分（从 *SSH* 密钥）生成或显示 Weblate 当前使用的公共密钥。

备注： 相应的私有 SSH 密钥当前无法使用密码，因此请确保已受到良好的保护。

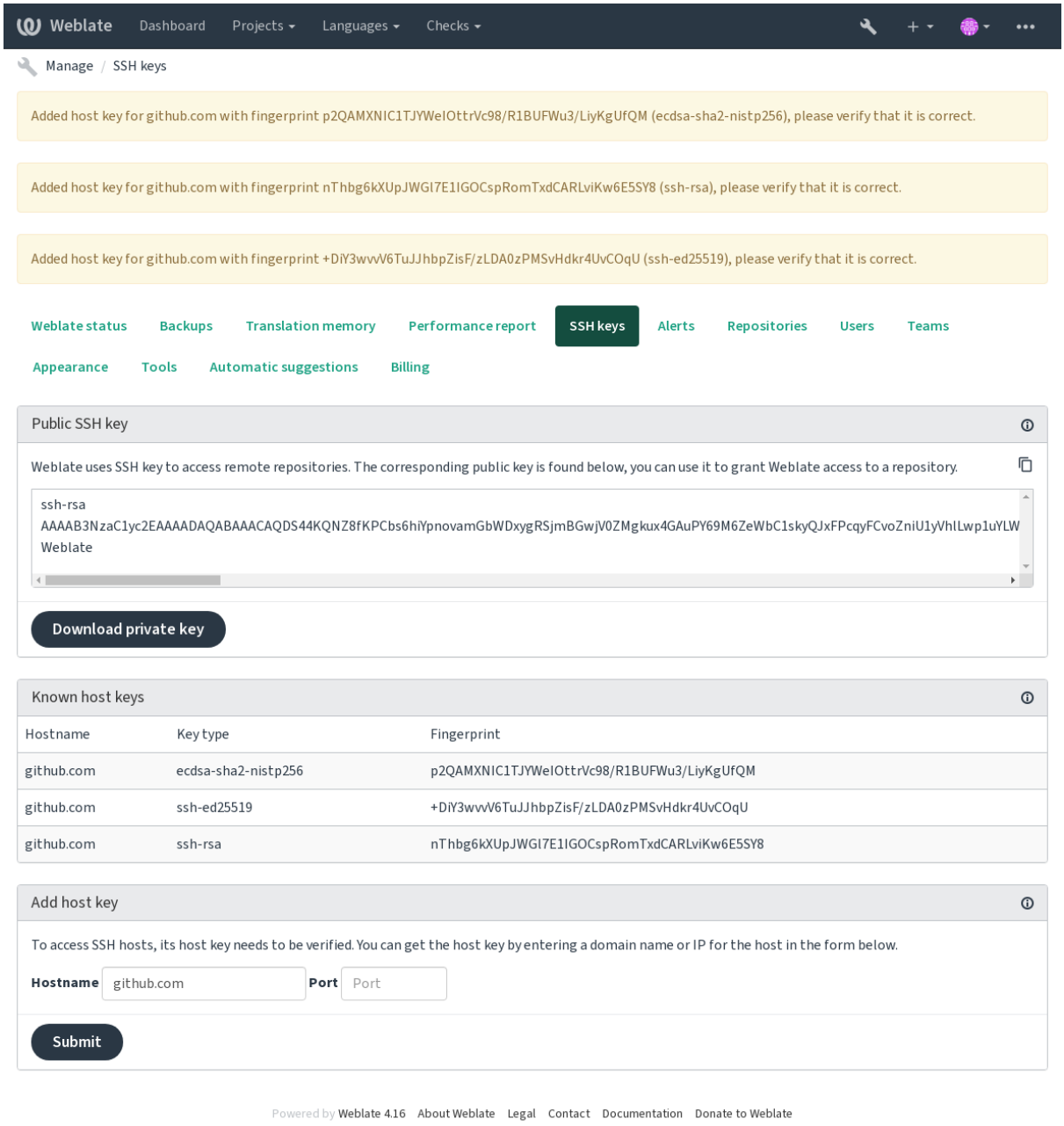
提示： 对生成的私有 Weblate SSH 密钥进行备份。

验证 SSH 主机密钥

Weblate 在第一次访问时自动存储 SSH 主机密钥，并记住它们以备将来使用。

如果你想在连接到仓库之前验证密钥指纹，在管理界面的同一区域，在 添加主机密钥中添加你要访问的服务器的 SSH 主机密钥。输入你要访问的主机名（例如：`gitlab.com`），然后点击 提交。确认其指纹与你添加的服务器相匹配。

添加的带指纹的密钥显示在确认消息中：



GitHub 仓库

可以通过 SSH 访问（请参阅*SSH 仓库*），但如果您需要访问多个仓库，您将遇到 GitHub 对允许使用的 SSH 密钥的限制（因为每个密钥只能使用一次）.

如果推送分支 未设置，则项目将被分叉并通过分叉推送更改。如果已设置，更改将推送到上游仓库和选择的分支。

对于较小的部署，使用带有个人访问令牌和您的 GitHub 账户的 HTTPS 身份验证，请参阅 创建用于命令行使用的访问令牌 。

对于更大的设置，通常最好为 Weblate 创建一个专用用户，为其分配在 Weblate 中生成的公共 SSH 密钥（请参阅*Weblate SSH 密钥*）并授予它访问您要翻译的所有仓库的权限。这种方法也用于 Hosted Weblate，有专门的 *weblate* 用户。

参见:

从 Hosted Weblate 访问仓库

Weblate 内部网址

通过在其他（链接）组件中将其位置称为 “weblate://project/component”，在不同组件之间共享一个仓库设置。这样，链接的组件使用主（引用）组件的版本控制系统（VCS）仓库配置。

警告： 删除主部件也会删除链接的部件。

如果 Weblate 找到具有匹配的仓库设置的部件，则在创建组件时会自动调整仓库 URL。您可以在部件配置的最后一步中覆盖它。

使用这个的原因：

- 为了节省服务器上的磁盘空间，仓库仅存储一次。
- 为了使更新更快，只更新一个仓库。
- 只有一个带有 Weblate 翻译的导出仓库（参见*Git 导出器* ）。
- 一些附加组件可以在共享一个仓库的多个组件上运行，例如挤压 *Git 提交* 。

HTTPS 仓库

要访问受保护的 HTTPS 仓库，请在 URL 中包含用户名和密码。不用担心，当 URL 显示给用户时，Weblate 会删除此信息（如果甚至允许查看仓库 URL）。

例如，添加了身份验证的 GitHub URL 可能如下所示：`https://user:your_access_token@github.com/WeblateOrg/weblate.git`。

备注： 如果你的用户名和密码包含特殊字符，需要对 URL 进行 encode 编码，例如 `https://user%40example.com:%24password%23@bitbucket.org/…`。

使用代理

如果需要使用代理服务器访问 HTTP/HTTPS VCS 仓库，请将 VCS 配置为使用它。

这可以使用 `http_proxy` 、`https_proxy` 和 `all_proxy` 环境变量来完成（如 cURL 文档 中所述）或通过在版本控制系统（VCS）配置中强制执行，例如：

```
git config --global http.proxy http://user:password@proxy.example.com:80
```

备注： 代理配置需要在运行 Weblate 的用户下完成（参见文件系统权限 ）并使用 `HOME=$DATA_DIR/home` （参见*DATA_DIR* ），否则由 Weblate 执行 Git 不会使用它。

参见：

cURL 手册页 、Git 配置文档

1.11.2 Git

提示： Weblate 需要 Git 2.12 或更新版本。

参见：

有关如何访问不同类型的仓库的信息，请参阅访问仓库 。

Git 强制推送

这与 Git 本身的行为完全一样，唯一的区别是它总是强制推送。这仅适用于使用单独的翻译仓库的情况。

警告： 请谨慎使用，因为这很容易导致上游仓库中的提交丢失。

自定义 Git 配置

Weblate 使用 `HOME=$DATA_DIR/home` 调用所有 VCS 命令（请参阅*DATA_DIR*），因此需要在 `DATA_DIR/home/.git` 中完成编辑用户配置。

Git 远程帮助程序

您还可以使用 Git 远程帮助程序 来额外支持其他版本控制系统，但要准备好调试这可能导致的问题。

目前，Bazaar 和 Mercurial 的助手可在 GitHub 上的单独仓库中使用：git-remote-hg 和 git-remote-bzr 。手动下载它们并将它们放在搜索路径中的某个位置（例如 `~/bin` ）。确保您安装了相应的版本控制系统。

一旦你安装了这些，这些远程可以用来在 Weblate 中指定一个仓库。

使用 Bazaar 从 Launchpad 克隆 `gnuhello` 项目：

```
bzr::lp:gnuhello
```

对于来自 selenic.com 的 `hello` 仓库使用 Mercurial：

```
hg::http://selenic.com/repo/hello
```

警告： 使用 Git 远程助手的不便之处在于 Mercurial，远程帮助程序有时会在推送更改时创建新的提示。

1.11.3 GitHub 拉取请求

在 2.3 版本加入.

这在*Git* 顶部添加了一个薄层，使用 GitHub API 允许将翻译更改作为拉取请求推送，而不是直接推送到仓库。

Git 将更改直接推送到仓库，而*GitHub 拉取请求* 创建拉取请求。仅访问 Git 仓库不需要后者。

您需要在 Weblate 设置中配置 API 凭据 (*GITHUB_CREDENTIALS*) 才能使其工作。配置完成后，您将在选择*版本控制系统（VCS）* 时看到 *GitHub* 选项。

参见：

推送 Weblate 的更改, *GITHUB_CREDENTIALS*

1.11.4 GitLab 合并请求

在 3.9 版本加入.

这只是使用 GitLab API 在*Git* 上添加了一个薄层，以允许将翻译更改作为合并请求推送，而不是直接推送到仓库。

不需要使用它来访问 Git 仓库，普通的*Git* 工作方式相同，唯一的区别是如何处理推送到仓库。使用*Git* 将更改直接推送到仓库，而*GitLab 合并请求* 创建合并请求。

您需要在 Weblate 设置中配置 API 凭据 (`GITLAB_CREDENTIALS`) 才能使其工作。配置完成后，您将在选择*版本控制系统（VCS）* 时看到 *GitLab* 选项。

参见:

推送 Weblate 的更改, `GITLAB_CREDENTIALS`

1.11.5 Gitea 拉取请求

在 4.12 版本加入.

这只是使用 Gitea API 在*Git* 上添加了一个薄层，以允许将翻译更改作为拉取请求推送，而不是直接推送到仓库。

不需要使用它来访问 Git 仓库，普通的*Git* 工作方式相同，唯一的区别是如何处理推送到仓库。使用*Git* 将更改直接推送到仓库，而*Gitea 拉取请求* 创建拉取请求。

您需要在 Weblate 设置中配置 API 凭据 (`GITEA_CREDENTIALS`) 才能使其工作。配置完成后，您将在选择*版本控制系统（VCS）* 时看到 *Gitea* 选项。

参见:

推送 Weblate 的更改, `GITEA_CREDENTIALS`

1.11.6 Bitbucket 服务器拉取请求

在 4.16 版本加入.

这只是使用 Bitbucket Server API 在*Git* 上添加了一个薄层，以允许将翻译更改作为拉取请求推送，而不是直接推送到仓库。

> **警告:** 这不支持 Bitbucket Cloud API。

不需要使用它来访问 Git 仓库，普通的*Git* 工作方式相同，唯一的区别是如何处理推送到仓库。使用*Git* 将更改直接推送到仓库，而*Bitbucket 服务器拉取请求* 创建拉取请求。

您需要在 Weblate 设置中配置 API 凭据 (`BITBUCKETSERVER_CREDENTIALS`) 才能使其工作。配置完成后，您将在选择*版本控制系统（VCS）* 时看到 *Bitbucket Server* 选项。

参见:

推送 Weblate 的更改, `BITBUCKETSERVER_CREDENTIALS`

1.11.7 Pagure 合并请求

在 4.3.2 版本加入.

这只是使用 Pagure API 在*Git* 上添加了一个薄层，以允许将翻译更改作为合并请求推送，而不是直接推送到仓库。

不需要使用它来访问 Git 仓库，普通的*Git* 工作方式相同，唯一的区别是如何处理推送到仓库。使用*Git* 将更改直接推送到仓库，而*Pagure 合并请求* 创建合并请求。

您需要在 Weblate 设置中配置 API 凭据 (*PAGURE_CREDENTIALS*) 才能使其工作。配置完成后，您将在选择*版本控制系统（VCS）* 时看到 *Pagure* 选项。

参见:

推送 Weblate 的更改, *PAGURE_CREDENTIALS*

1.11.8 Gerrit

在 2.2 版本加入.

使用 git-review 工具在*Git* 上添加一个薄层，以允许将翻译更改作为 Gerrit 审查请求推送，而不是将它们直接推送到仓库。

Gerrit 文档详细介绍了设置此类仓库所需的配置。

1.11.9 Mercurial

在 2.1 版本加入.

Mercurial 是另一个可以在 Weblate 中直接使用的版本控制系统（VCS）。

备注: 它应该适用于任何 Mercurial 版本，但有时对命令行界面的不兼容更改会破坏 Weblate 集成。

参见:

有关如何访问不同类型的仓库的信息，请参阅*访问仓库* 。

1.11.10 Subversion

在 2.8 版本加入.

Weblate 使用 git-svn 与 subversion 仓库交互。它是一个 Perl 脚本，允许 Git 客户端使用 subversion，使用户能够维护内部仓库的完整克隆并在本地提交。

备注: Weblate 尝试自动检测 Subversion 仓库布局 - 它支持分支的直接 URL 或具有标准布局的仓库（分支/、标签/和主干/）。有关这方面的更多信息，请参阅 git-svn 文档 。如果您的仓库没有标准布局并且遇到错误，请尝试在仓库 URL 中包含分支名称并将分支留空。

在 2.19 版本发生变更: 在此之前，仅支持使用标准布局的仓库。

Subversion 凭据

Weblate 希望您预先接受证书（如果需要，还需要您的凭据）。它会将它们插入到*DATA_DIR* 目录中。在 *$HOME* 环境变量设置为*DATA_DIR* 的情况下使用 *svn* 接受证书：

```
# Use DATA_DIR as configured in Weblate settings.py, it is /app/data in the Docker
HOME=${DATA_DIR}/home svn co https://svn.example.com/example
```

参见：

DATA_DIR

1.11.11 本地文件

1.11.12 Git

提示： 下面，它使用*Git*。它需要安装 Git，并允许您切换到本机使用 Git，并提供完整的翻译历史记录。

在 3.8 版本加入.

Weblate 也可以在没有远程 VCS 的情况下运行。初始翻译是通过上传来导入的。稍后您可以通过文件上传替换单个文件，或直接从 Weblate 添加翻译字符串（目前仅适用于单语翻译）。

在后台 Weblate 为您创建一个 Git 仓库并跟踪所有更改。如果您以后决定使用版本控制系统（VCS）来存储翻译，您已经在 Weblate 中有一个仓库可以作为您的集成的基础。

1.12 Weblate 的 REST API

在 2.6 版本加入: 从 Weblate 2.6 开始可以使用 REST API。

API 可以在 `/api/` URL 上访问，并且它基于 Django REST 框架 。你可以直接使用或参考*Weblate 客户端*。

1.12.1 身份验证和通用参数

公共项目的 API 无需身份验证即可使用，但未经身份验证的请求会受到严格的限制（默认为每天 100 个请求），所以建议使用身份验证。身份验证使用令牌，你可以在你的个人资料中获取该令牌。在 `Authorization` 标头中使用它：

ANY /

API 的通用请求行为、标头、状态码和此处的参数也适用于所有端点。

查询参数

- **format** –响应格式（覆盖了 Accept）。可能的值取决于 REST 框架设置，默认支持 `json` 和 `api`。后者为 API 提供了 web 浏览器接口。
- **page** –返回给定页面的分页结果 (使用 *next* 和 *previous* 字段来响应的自动导航).

请求标头

- Accept –相应内容的类型取决于 Accept 标头
- Authorization –验证为 `Authorization: Token YOUR-TOKEN` 的可选令牌

响应标头

- Content-Type –这取决于请求的标头 Accept
- Allow –对象允许的 HTTP 方法的列表

响应 JSON 对象

- **detail** (*string*) –结果的详细说明（对于 200 OK 以外的 HTTP 状态码）
- **count** (*int*) –对象列表的总项目计数
- **next** (*string*) –对象列表的下一页 URL
- **previous** (*string*) –对象列表的上一页 URL
- **results** (*array*) –对象列表的结果
- **url** (*string*) –使用 API 访问这个资源的 URL
- **web_url** (*string*) –使用浏览器访问这个资源的 URL

状态码

- 200 OK –当请求被正确地处理时
- 201 Created –当成功创建新对象时
- 204 No Content –当成功删除对象时
- 400 Bad Request –当表单参数丢失时
- 403 Forbidden –当访问被拒绝时
- 429 Too Many Requests –当出现瓶颈时

身份验证令牌

在 4.10 版本发生变更: 在 4.10 版本中引入了项目范围的令牌。

每个用户都有自己的个人访问令牌, 可以在用户档案中获得。新生成的用户访问令牌带有 `wlu_` 前缀。

可以创建项目范围的访问令牌，只用于访问指定项目的 API。这些访问令牌带有 `wlp_` 前缀。

身份验证的示例

示例请求：

```
GET /api/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Authorization: Token YOUR-TOKEN
```

示例响应：

```
HTTP/1.0 200 OK
Date: Fri, 25 Mar 2016 09:46:12 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, HEAD, OPTIONS

{
    "projects":"http://example.com/api/projects/",
    "components":"http://example.com/api/components/",
    "translations":"http://example.com/api/translations/",
    "languages":"http://example.com/api/languages/"
}
```

CURL 示例：

```
curl \
    -H "Authorization: Token TOKEN" \
    https://example.com/api/
```

传递参数的示例

对于 POST 方法，参数可以指定为表单提交（*application/x-www-form-urlencoded*）或 JSON（*application/json*）。

表单请求示例：

```
POST /api/projects/hello/repository/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Token TOKEN

operation=pull
```

JSON 请求的示例：

```
POST /api/projects/hello/repository/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{"operation":"pull"}
```

CURL 示例：

```
curl \
    -d operation=pull \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/components/hello/weblate/repository/
```

CURL JSON 示例：

```
curl \
    --data-binary '{"operation":"pull"}' \
    -H "Content-Type: application/json" \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/components/hello/weblate/repository/
```

API 频次限制

这个 API 请求限制了速率；对于匿名用户默认配置限制为每天 100 个请求，对于身份验证的用户限制为每小时 5000 个请求。

速率限制可以在 `settings.py` 中调整；如何配置它的更多细节请参见 Throttling in Django REST framework documentation。

在 Docker 容器中可使用 *WEBLATE_API_RATELIMIT_ANON* 和 *WEBLATE_API_RATELIMIT_USER* 对此进行配置。

速率限制在后面的标头中报告：

X-RateLimit-Limit	要执行的对速率限制进行限制的请求
X-RateLimit-Remaining	保持限制的请求
X-RateLimit-Remaining	直到速率限制窗口重置时的秒数

在 4.1 版本发生变更: 添加速率限制状态的标头。

参见:

频次限制, 频次限制, *WEBLATE_API_RATELIMIT_ANON*, *WEBLATE_API_RATELIMIT_USER*

1.12.2 API 入口点

GET /api/

API 根入口点。

示例请求:

```
GET /api/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Authorization: Token YOUR-TOKEN
```

示例响应:

```
HTTP/1.0 200 OK
Date: Fri, 25 Mar 2016 09:46:12 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, HEAD, OPTIONS

{
    "projects":"http://example.com/api/projects/",
    "components":"http://example.com/api/components/",
    "translations":"http://example.com/api/translations/",
    "languages":"http://example.com/api/languages/"
}
```

1.12.3 用户

在 4.0 版本加入.

GET /api/users/

返回用户列表，如果有权限查看管理用户的话。如果没有，那么会只看到自己的具体信息。

参见:

用户对象属性记录在*GET /api/users/(str:username)/*。

POST /api/users/

创建新用户。

参数

- **username** (*string*) –用户名
- **full_name** (*string*) –用户全名
- **email** (*string*) –用户电子邮箱

- **is_superuser** (*boolean*) –用户是超级用户吗?（可选的）
- **is_active** (*boolean*) –用户是活动用户吗?（可选的）
- **is_bot** (*boolean*) –用户是机器人吗?（可选）（用于项目范围内的访问令牌）

GET /api/users/(str: *username*)/

返回用户的信息。

参数

- **username** (*string*) –用户的用户名

响应 JSON 对象

- **username** (*string*) –用户的用户名
- **full_name** (*string*) –用户的全名
- **email** (*string*) –用户的电子邮箱
- **is_superuser** (*boolean*) –用户是否是超级用户
- **is_active** (*boolean*) –用户是否是活动用户
- **is_bot** (*boolean*) –用户是否是机器人（用于项目范围的令牌）
- **date_joined** (*string*) –创建用户的日期
- **groups** (*array*) –连接到关联的组；请参见*GET /api/groups/(int:id)/*

示例 JSON 数据:

```
{
    "email": "user@example.com",
    "full_name": "Example User",
    "username": "exampleusername",
    "groups": [
        "http://example.com/api/groups/2/",
        "http://example.com/api/groups/3/"
    ],
    "is_superuser": true,
    "is_active": true,
    "is_bot": false,
    "date_joined": "2020-03-29T18:42:42.617681Z",
    "url": "http://example.com/api/users/exampleusername/",
    "statistics_url": "http://example.com/api/users/exampleusername/statistics/
↪"
}
```

PUT /api/users/(str: *username*)/

更改用户参数。

参数

- **username** (*string*) –用户的用户名

响应 JSON 对象

- **username** (*string*) –用户的用户名
- **full_name** (*string*) –用户的全名
- **email** (*string*) –用户的电子邮箱
- **is_superuser** (*boolean*) –用户是否是超级用户
- **is_active** (*boolean*) –用户是否是活动用户
- **is_bot** (*boolean*) –用户是否是机器人（用于项目范围的令牌）
- **date_joined** (*string*) –创建用户的日期

PATCH /api/users/(**str:** *username*)**/**

更改用户参数。

参数

- **username** (*string*) –用户的用户名

响应 JSON 对象

- **username** (*string*) –用户的用户名
- **full_name** (*string*) –用户的全名
- **email** (*string*) –用户的电子邮箱
- **is_superuser** (*boolean*) –用户是否是超级用户
- **is_active** (*boolean*) –用户是否是活动用户
- **is_bot** (*boolean*) –用户是否是机器人（用于项目范围的令牌）
- **date_joined** (*string*) –创建用户的日期

DELETE /api/users/(**str:** *username*)**/**

删除所有的用户信息并将用户标记为不活动用户。

参数

- **username** (*string*) –用户的用户名

POST /api/users/(**str:** *username*)**/groups/**

将群组与用户关联。

参数

- **username** (*string*) –用户的用户名

表单参数

- **string group_id** –唯一的群组 ID

DELETE /api/users/(**str:** *username*)**/groups/**

在 4.13.1 版本加入.

从组中删除用户。

参数

- **username** (*string*) –用户的用户名

表单参数

- **string group_id** –唯一的群组 ID

GET /api/users/(**str:** *username*)**/statistics/**

用户的统计数据列表。

参数

- **username** (*string*) –用户的用户名

响应 JSON 对象

- **translated** (*int*) –用户翻译的数量
- **suggested** (*int*) –用户提交建议的数量
- **uploaded** (*int*) –用户上传次数
- **commented** (*int*) –用户评论的数量
- **languages** (*int*) –用户能够翻译的语言数量

GET /api/users/(**str:** *username*)**/notifications/**

用户的订阅列表。

参数

- **username** (*string*) –用户的用户名

POST /api/users/(**str:** *username*)**/notifications/**

将订阅与用户关联。

参数

- **username** (*string*) –用户的用户名

请求 JSON 对象

- **notification** (*string*) –注册通知的名称
- **scope** (*int*) –可用选项的通知范围
- **frequency** (*int*) –通知的频率选项

GET /api/users/(**str:** *username*)**/notifications/**
int: *subscription_id***/**

获得与用户关联的订阅。

参数

- **username** (*string*) –用户的用户名
- **subscription_id** (*int*) –已注册通知 ID

PUT /api/users/(**str:** *username*)**/notifications/**
int: *subscription_id***/**

编辑与用户关联的订阅。

参数

- **username** (*string*) –用户的用户名
- **subscription_id** (*int*) –已注册通知 ID

请求 JSON 对象

- **notification** (*string*) –注册通知的名称
- **scope** (*int*) –可用选项的通知范围
- **frequency** (*int*) –通知的频率选项

PATCH /api/users/(**str:** *username*)**/notifications/**
int: *subscription_id***/**

编辑与用户关联的订阅。

参数

- **username** (*string*) –用户的用户名
- **subscription_id** (*int*) –已注册通知 ID

请求 JSON 对象

- **notification** (*string*) –注册通知的名称
- **scope** (*int*) –可用选项的通知范围
- **frequency** (*int*) –通知的频率选项

DELETE /api/users/(**str:** *username*)**/notifications/**
int: *subscription_id***/**

删除与用户关联的订阅。

参数

- **username** (*string*) –用户的用户名
- **subscription_id** –注册通知的名称
- **subscription_id** –int

1.12.4 群组

在 4.0 版本加入.

GET /api/groups/

返回群组列表，如果有权限看到管理群组的话，如果没有，那么会只看到用户所在的群组。

参见:

群组对象属性记录在*GET /api/groups/(int:id)/*。

POST /api/groups/

创建新的群组。

参数

- **name** (*string*) –组名
- **project_selection** (*int*) –给定选项的项目选择的群组
- **language_selection** (*int*) –给定选项的语言选择的群组
- **defining_project** (*str*) –链接到定义项目，用于管理每个项目的访问控制；请参见*GET /api/projects/(string:project)/*

GET /api/groups/(int: *id*)/

返回群组的信息。

参数

- **id** (*int*) –群组的 ID

响应 JSON 对象

- **name** (*string*) –群组的名称
- **project_selection** (*int*) –相应于对象群组的整数
- **language_selection** (*int*) –相应于语言群组的整数
- **roles** (*array*) –相关联角色的连接；请参见*GET /api/roles/(int:id)/*
- **projects** (*array*) –相关联项目的连接；请参见*GET /api/projects/(string:project)/*
- **components** (*array*) –相关联部件的连接；请参见*GET /api/components/(string:project)/(string:component)/*
- **componentlists** (*array*) –相关联部件列表的连接；请参见*GET /api/component-lists/(str:slug)/*
- **defining_project** (*str*) –链接到定义项目，用于管理每个项目的访问控制；请参见*GET /api/projects/(string:project)/*

示例 JSON 数据:

```
{
    "name": "Guests",
    "defining_project": null,
    "project_selection": 3,
    "language_selection": 1,
```

(续下页)

(接上页)

```
    "url": "http://example.com/api/groups/1/",
    "roles": [
        "http://example.com/api/roles/1/",
        "http://example.com/api/roles/2/"
    ],
    "languages": [
        "http://example.com/api/languages/en/",
        "http://example.com/api/languages/cs/",
    ],
    "projects": [
        "http://example.com/api/projects/demo1/",
        "http://example.com/api/projects/demo/"
    ],
    "componentlist": "http://example.com/api/component-lists/new/",
    "components": [
        "http://example.com/api/components/demo/weblate/"
    ]
}
```

PUT /api/groups/(**int:** *id*)**/**

更改群组参数。

参数

- **id** (*int*) – 群组的 ID

响应 JSON 对象

- **name** (*string*) – 群组的名称
- **project_selection** (*int*) – 相应于对象群组的整数
- **language_selection** (*int*) – 相应于语言群组的整数

PATCH /api/groups/(**int:** *id*)**/**

更改群组参数。

参数

- **id** (*int*) – 群组的 ID

响应 JSON 对象

- **name** (*string*) – 群组的名称
- **project_selection** (*int*) – 相应于对象群组的整数
- **language_selection** (*int*) – 相应于语言群组的整数

DELETE /api/groups/(**int:** *id*)**/**

删除群组。

参数

- **id** (*int*) – 群组的 ID

POST /api/groups/(**int:** *id*)**/roles/**

将角色与群组关联。

参数

- **id** (*int*) – 群组的 ID

表单参数

- **string role_id** – 唯一的角色 ID

POST /api/groups/(**int:** *id*)**/components/**

将部件与群组关联。

参数

- **id**(*int*) –群组的 ID

表单参数

- **string component_id** –唯一的部件 ID

DELETE /api/groups/(**int:** *id*)**/components/**
int: *component_id*

从群组删除部件。

参数

- **id**(*int*) –群组的 ID
- **component_id**(*int*) –唯一的部件 ID

POST /api/groups/(**int:** *id*)**/projects/**

将项目与群组关联。

参数

- **id**(*int*) –群组的 ID

表单参数

- **string project_id** –唯一的项目 ID

DELETE /api/groups/(**int:** *id*)**/projects/**
int: *project_id*

从群组删除项目。

参数

- **id**(*int*) –群组的 ID
- **project_id**(*int*) –唯一的项目 ID

POST /api/groups/(**int:** *id*)**/languages/**

将语言与群组关联。

参数

- **id**(*int*) –群组的 ID

表单参数

- **string language_code** –唯一的语言代码

DELETE /api/groups/(**int:** *id*)**/languages/**
string: *language_code*

从群组删除语言。

参数

- **id**(*int*) –群组的 ID
- **language_code**(*string*) –唯一的语言代码

POST /api/groups/(**int:** *id*)**/componentlists/**

将部件列表与群组关联。

参数

- **id**(*int*) –群组的 ID

表单参数

- **string component_list_id** –唯一的部件列表 ID

DELETE /api/groups/(**int:** *id*)**/componentlists/**
int: *component_list_id*

从群组删除部件列表。

参数

- **id** (*int*) –群组的 ID
- **component_list_id** (*int*) –唯一的部件列表 ID

1.12.5 角色

GET /api/roles/

返回与用户关联的所有角色列表。如果用户是超级用户，那么返回所有现有角色的列表。

参见:

角色对象属性记录在*GET /api/roles/(int:id)/*。

POST /api/roles/

创建新角色。

参数

- **name** (*string*) –角色名称
- **permissions** (*array*) –权限代号列表

GET /api/roles/(**int:** *id*)**/**

返回角色的信息。

参数

- **id** (*int*) –角色 ID

响应 JSON 对象

- **name** (*string*) –角色名称
- **permissions** (*array*) –权限代号列表

示例 JSON 数据:

```
{
    "name": "Access repository",
    "permissions": [
        "vcs.access",
        "vcs.view"
    ],
    "url": "http://example.com/api/roles/1/",
}
```

PUT /api/roles/(**int:** *id*)**/**

更改角色参数。

参数

- **id** (*int*) –角色的 ID

响应 JSON 对象

- **name** (*string*) –角色名称
- **permissions** (*array*) –权限代号列表

PATCH /api/roles/(**int:** *id*)**/**

更改角色参数。

参数

- **id** (*int*) –角色的 ID

响应 JSON 对象

- **name** (*string*) –角色名称
- **permissions** (*array*) –权限代号列表

DELETE /api/roles/(**int:** *id*)**/**

删除角色。

参数

- **id** (*int*) –角色的 ID

1.12.6 语言

GET /api/languages/

返回所有语言的列表。

参见:

语言对象属性记录在*GET /api/languages/(string:language)/*。

POST /api/languages/

创建新的语言。

参数

- **code** (*string*) –语言名称
- **name** (*string*) –语言名称
- **direction** (*string*) –文字方向
- **population** (*int*) –语言使用者数量
- **plural** (*object*) –语言复数公式和数量

GET /api/languages/(**string:** *language*)**/**

返回语言的信息。

参数

- **language** (*string*) –语言代码

响应 JSON 对象

- **code** (*string*) –语言代码
- **direction** (*string*) –文字方向
- **plural** (*object*) –语言复数信息的对象
- **aliases** (*array*) –语言别名的数组

请求 JSON 对象

- **population** (*int*) –语言使用者数量

示例 JSON 数据:

```
{
    "code": "en",
    "direction": "ltr",
    "name": "English",
    "population": 159034349015,
    "plural": {
        "id": 75,
        "source": 0,
        "number": 2,
        "formula": "n != 1",
        "type": 1
    },
    "aliases": [
        "english",
        "en_en",
        "base",
        "source",
        "eng"
    ],
    "url": "http://example.com/api/languages/en/",
    "web_url": "http://example.com/languages/en/",
    "statistics_url": "http://example.com/api/languages/en/statistics/"
}
```


PUT /api/languages/(**string:** *language*)**/**

更改语言参数。

参数

- **language** (*string*) –语言的代码

请求 JSON 对象

- **name** (*string*) –语言名称
- **direction** (*string*) –文字方向
- **population** (*int*) –语言使用者数量
- **plural** (*object*) –语言复数的细节

PATCH /api/languages/(**string:** *language*)**/**

更改语言参数。

参数

- **language** (*string*) –语言的代码

请求 JSON 对象

- **name** (*string*) –语言名称
- **direction** (*string*) –文字方向
- **population** (*int*) –语言使用者数量
- **plural** (*object*) –语言复数的细节

DELETE /api/languages/(**string:** *language*)**/**

删除该语言。

参数

- **language** (*string*) –语言的代码

GET /api/languages/(**string:** *language*)**/statistics/**

返回语言的统计数据。

参数

- **language** (*string*) –语言代码

响应 JSON 对象

- **total** (*int*) –字符串的总数
- **total_words** (*int*) –词的总数
- **last_change** (*timestamp*) –语言的最后更改
- **recent_changes** (*int*) –更改的总数
- **translated** (*int*) –已翻译字符串的数量
- **translated_percent** (*float*) –已翻译字符串的百分比
- **translated_words** (*int*) –已翻译词的数量
- **translated_words_percent** (*int*) –已翻译词的百分比
- **translated_chars** (*int*) –已翻译字符的数量
- **translated_chars_percent** (*int*) –已翻译字符的百分比
- **total_chars** (*int*) –总字符的数量
- **fuzzy** (*int*) –模糊（标记为需要编辑）字符串的数量
- **fuzzy_percent** (*int*) –模糊字符串（标记为需要编辑）的百分比
- **failing** (*int*) –失败字符串的数量
- **failing** –失败字符串的百分比

1.12.7 项目

GET /api/projects/

返回所有项目的列表。

参见:

项目对象属性记录在*GET /api/projects/(string:project)/*。

POST /api/projects/

在 3.9 版本加入.

创建新项目。

参数

- **name** (*string*) –项目名称
- **slug** (*string*) –项目标识串
- **web** (*string*) –项目网站

GET /api/projects/(string: *project*)/

返回项目的信息。

参数

- **project** (*string*) –项目 URL 标识符

响应 JSON 对象

- **name** (*string*) –项目名称
- **slug** (*string*) –项目标识串
- **web** (*string*) –项目网站

- **components_list_url** (*string*) –部件列表的 URL；请参见*GET /api/projects/(string:project)/components/*
- **repository_url** (*string*) – 仓库状态的 URL；请参见*GET /api/projects/(string:project)/repository/*
- **changes_list_url** (*string*) – 更改列表的 URL；请参见*GET /api/projects/(string:project)/repository/*
- **translation_review** (*boolean*) –启用审校
- **source_review** (*boolean*) –启用原文审校
- **set_language_team** (*boolean*) –设置“*Language-Team*”标头
- **enable_hooks** (*boolean*) –启用钩子
- **instructions** (*string*) –翻译说明
- **language_aliases** (*string*) –语言别名

示例 JSON 数据：

```
{
    "name": "Hello",
    "slug": "hello",
    "url": "http://example.com/api/projects/hello/",
    "web": "https://weblate.org/",
    "web_url": "http://example.com/projects/hello/"
}
```

PATCH /api/projects/(string: *project*)/

在 4.3 版本加入.

通过 PATCH 请求来编辑一个项目。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

PUT /api/projects/(string: *project*)/

在 4.3 版本加入.

通过 PUT 请求来编辑一个项目。

参数

- **project** (*string*) –项目 URL 标识符

DELETE /api/projects/(string: *project*)/

在 3.9 版本加入.

删除项目。

参数

- **project** (*string*) –项目 URL 标识符

GET /api/projects/(string: *project*)/changes/

返回项目更改的列表。这本质上是一个项目范围的*GET /api/changes/*接受相同的参数。

参数

- **project** (*string*) –项目 URL 标识符

响应 JSON 对象

- **results** (*array*) – 部件对象的数组；请参见*GET /api/changes/(int:id)/*

GET /api/projects/(string: *project*)/repository/

返回版本控制系统（VCS）仓库状态的信息。这个端点只包含项目所有仓库的整体概况。为了得到更多细节，请使用*GET /api/components/(string:project)/(string:component)/repository/*。

参数

- **project** (*string*) –项目 URL 标识符

响应 JSON 对象

- **needs_commit** (*boolean*) –是否有任何要提交的待处理更改
- **needs_merge** (*boolean*) –是否有上游更改要合并
- **needs_push** (*boolean*) –是否有本地更改要推送

示例 JSON 数据：

```
{
    "needs_commit": true,
    "needs_merge": false,
    "needs_push": true
}
```

POST /api/projects/(string: *project*)/repository/

在版本控制系统（VCS）仓库上执行给定的操作。

参数

- **project** (*string*) –项目 URL 标识符

请求 JSON 对象

- **operation** (*string*) – 要执行的操作：`push`、`pull`、`commit`、`reset`、`cleanup`、`file-sync` 中的一个

响应 JSON 对象

- **result** (*boolean*) –操作的结果

CURL 示例：

```
curl \
    -d operation=pull \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/projects/hello/repository/
```

JSON 请求的示例：

```
POST /api/projects/hello/repository/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{"operation":"pull"}
```

JSON 响应的示例：

```
HTTP/1.0 200 OK
Date: Tue, 12 Apr 2016 09:32:50 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
```

(续下页)

(接上页)

```
Content-Type: application/json
Content-Language: en
Allow: GET, POST, HEAD, OPTIONS

{"result":true}
```

GET /api/projects/(string: *project*)/components/

返回给定项目的翻译部件列表。

参数

- **project** (*string*) –项目 URL 标识符

响应 JSON 对象

- **results** (*array*) – 部件对象的数组；请参见*GET /api/components/(string:project)/(string:component)/*

POST /api/projects/(string: *project*)/components/

在 3.9 版本加入.

在 4.3 版本发生变更: `zipfile` 和 `docfile` 参数现在可用于无 VCS 的部件，参见本地文件。

在 4.6 版本发生变更: 克隆的仓库现在可以使用*Weblate 内部网址*. 使用 `disable_autoshare` 关闭此功能。

在给定的项目中新建翻译部件。

提示: 从单个版本控制系统（VCS）仓库创建多个部件时，请使用*Weblate 内部网址*。

备注: 多数部件的新建发生在后台。检查新建部件的 `task_url` 属性，并按照那里的步骤进行。

参数

- **project** (*string*) –项目 URL 标识符

表单参数

- **file zipfile** –上传到 Weblate 用于翻译初始化的 ZIP 文件
- **file docfile** –要翻译的文档
- **boolean disable_autoshare** –禁用通过:ref:‘internal-urls‘自动共享仓库。

请求 JSON 对象

- **object** – 部件参数，参见*GET /api/components/(string:project)/(string:component)/*

响应 JSON 对象

- **result** (*object*) – 新建部件对象；请参见*GET /api/components/(string:project)/(string:component)/*

使用 “zipfile“ 上传文件时无法使用 JSON 和 “docfile“参数数据必须以:mimetype:‘multipart/form-data‘的形式上传。

CURL 表单请求示例:

```
curl \
    --form docfile=@strings.html \
    --form name=Weblate \
```

(续下页)

(接上页)

```
    --form slug=weblate \
    --form file_format=html \
    --form new_lang=add \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/projects/hello/components/
```

CURL JSON 请求的示例：

```
curl \
    --data-binary '{
        "branch": "main",
        "file_format": "po",
        "filemask": "po/*.po",
        "name": "Weblate",
        "slug": "weblate",
        "repo": "https://github.com/WeblateOrg/hello.git",
        "template": "",
        "new_base": "po/hello.pot",
        "vcs": "git"
    }' \
    -H "Content-Type: application/json" \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/projects/hello/components/
```

从 Git 创建新部件的 JSON 请求：

```
POST /api/projects/hello/components/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{
    "branch": "main",
    "file_format": "po",
    "filemask": "po/*.po",
    "name": "Weblate",
    "slug": "weblate",
    "repo": "https://github.com/WeblateOrg/hello.git",
    "template": "",
    "new_base": "po/hello.pot",
    "vcs": "git"
}
```

从另一个部件创建新部件的 JSON 请求：

```
POST /api/projects/hello/components/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{
    "file_format": "po",
    "filemask": "po/*.po",
    "name": "Weblate",
    "slug": "weblate",
    "repo": "weblate://weblate/hello",
    "template": "",
```

(续下页)

(接上页)

```
    "new_base": "po/hello.pot",
    "vcs": "git"
}
```

JSON 响应的示例：

```
HTTP/1.0 200 OK
Date: Tue, 12 Apr 2016 09:32:50 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, POST, HEAD, OPTIONS

{
    "branch": "main",
    "file_format": "po",
    "filemask": "po/*.po",
    "git_export": "",
    "license": "",
    "license_url": "",
    "name": "Weblate",
    "slug": "weblate",
    "project": {
        "name": "Hello",
        "slug": "hello",
        "source_language": {
            "code": "en",
            "direction": "ltr",
             "population": 159034349015,
            "name": "English",
            "url": "http://example.com/api/languages/en/",
            "web_url": "http://example.com/languages/en/"
        },
        "url": "http://example.com/api/projects/hello/",
        "web": "https://weblate.org/",
        "web_url": "http://example.com/projects/hello/"
    },
    "repo": "file:///home/nijel/work/weblate-hello",
    "template": "",
    "new_base": "",
    "url": "http://example.com/api/components/hello/weblate/",
    "vcs": "git",
    "web_url": "http://example.com/projects/hello/weblate/"
}
```

GET /api/projects/(string: *project*)/languages/

对项目内的所有语言返回编页的统计数据。

在 3.8 版本加入.

参数

- **project** (*string*) –项目 URL 标识符

响应 JSON 对象

- **results** (*array*) –翻译统计数据对象的数组
- **language** (*string*) –语言名称
- **code** (*string*) –语言代码

- **total** (*int*) –字符串的总数
- **translated** (*int*) –已翻译字符串的数量
- **translated_percent** (*float*) –已翻译字符串的百分比
- **total_words** (*int*) –词的总数
- **translated_words** (*int*) –已翻译词的数量
- **words_percent** (*float*) –已翻译词的百分比

GET /api/projects/(**string:** *project*)**/statistics/**

返回项目的统计数据。

在 3.8 版本加入.

参数

- **project** (*string*) –项目 URL 标识符

响应 JSON 对象

- **total** (*int*) –字符串的总数
- **translated** (*int*) –已翻译字符串的数量
- **translated_percent** (*float*) –已翻译字符串的百分比
- **total_words** (*int*) –词的总数
- **translated_words** (*int*) –已翻译词的数量
- **words_percent** (*float*) –已翻译词的百分比

1.12.8 部件

提示: 使用:http:post:*/api/projects/(string:project)/components/* 创建新部件。

GET /api/components/

返回一个翻译部件的列表。

参见:

部件对象属性记录在*GET /api/components/(string:project)/(string:component)/*。

GET /api/components/(**string:** *project*)**/**
string: *component***/**

返回翻译部件的信息。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

响应 JSON 对象

- **project** (*object*) – 翻译项目; 请参见*GET /api/projects/(string:project)/*
- **name** (*string*) –部件名称
- **slug** (*string*) –部件标识串
- **vcs** (*string*) –版本控制系统（*VCS*）
- **repo** (*string*) –源代码仓库

- **git_export** (*string*) –已导出仓库 *URL*
- **branch** (*string*) –仓库分支
- **push_branch** (*string*) –推送分支
- **filemask** (*string*) –文件掩码
- **template** (*string*) –单语言译文模版语言文件
- **edit_template** (*string*) –编辑译文模版文件
- **intermediate** (*string*) –中间语言文件
- **new_base** (*string*) –新翻译的翻译模版
- **file_format** (*string*) –文件格式
- **license** (*string*) –翻译许可证
- **agreement** (*string*) –贡献者协议
- **new_lang** (*string*) –添加新译文
- **language_code_style** (*string*) –语言代码风格
- **source_language** (*object*) –源语言对象；请参见*GET /api/languages/(string:language)/*
- **push** (*string*) –仓库推送 *URL*
- **check_flags** (*string*) –翻译标记
- **priority** (*string*) –优先级
- **enforced_checks** (*string*) –强制检查
- **restricted** (*string*) –受限制的访问
- **repoweb** (*string*) –仓库浏览器
- **report_source_bugs** (*string*) –源字符串缺陷报告地址
- **merge_style** (*string*) –合并方式
- **commit_message** (*string*) –提交、添加、删除、合并、附加组件及合并请求说明
- **add_message** (*string*) –提交、添加、删除、合并、附加组件及合并请求说明
- **delete_message** (*string*) –提交、添加、删除、合并、附加组件及合并请求说明
- **merge_message** (*string*) –提交、添加、删除、合并、附加组件及合并请求说明
- **addon_message** (*string*) –提交、添加、删除、合并、附加组件及合并请求说明
- **pull_message** (*string*) –提交、添加、删除、合并、附加组件及合并请求说明
- **allow_translation_propagation** (*string*) –允许同步翻译
- **enable_suggestions** (*string*) –启用建议
- **suggestion_voting** (*string*) –建议投票
- **suggestion_autoaccept** (*string*) –自动接受建议
- **push_on_commit** (*string*) –提交时推送
- **commit_pending_age** (*string*) –对更改进行提交的延时时间
- **auto_lock_error** (*string*) –出错时锁定
- **language_regex** (*string*) –语言筛选

- **variant_regex** (*string*) –正则表达式变体
- **repository_url** (*string*) – 仓库状态的 URL；请参见*GET /api/components/(string:project)/(string:component)/repository/*
- **translations_url** (*string*) – 翻译列表的 URL；请参见*GET /api/components/(string:project)/(string:component)/translations/*
- **lock_url** (*string*) –锁定状态的 URL；请参见*GET /api/components/(string:project)/(string:component)/lock/*
- **changes_list_url** (*string*) –更改的列表的 URL；请参见*GET /api/components/(string:project)/(string:component)/changes/*
- **task_url** (*string*) –后台任务 URL（如果有的话）；请参见*GET /api/tasks/(str:uuid)/*

示例 JSON 数据：

```
{
    "branch": "main",
    "file_format": "po",
    "filemask": "po/*.po",
    "git_export": "",
    "license": "",
    "license_url": "",
    "name": "Weblate",
    "slug": "weblate",
    "project": {
        "name": "Hello",
        "slug": "hello",
        "source_language": {
            "code": "en",
            "direction": "ltr",
             "population": 159034349015,
            "name": "English",
            "url": "http://example.com/api/languages/en/",
            "web_url": "http://example.com/languages/en/"
        },
        "url": "http://example.com/api/projects/hello/",
        "web": "https://weblate.org/",
        "web_url": "http://example.com/projects/hello/"
    },
    "source_language": {
        "code": "en",
        "direction": "ltr",
        "population": 159034349015,
        "name": "English",
        "url": "http://example.com/api/languages/en/",
        "web_url": "http://example.com/languages/en/"
    },
    "repo": "file:///home/nijel/work/weblate-hello",
    "template": "",
    "new_base": "",
    "url": "http://example.com/api/components/hello/weblate/",
    "vcs": "git",
    "web_url": "http://example.com/projects/hello/weblate/"
}
```

PATCH /api/components/(string: *project*)/
string: *component*/

通过 PATCH 请求编辑一个部件。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **source_language** (*string*) –项目源语言代码（可选）

请求 JSON 对象

- **name** (*string*) –部件名称
- **slug** (*string*) –部件的标识串
- **repo** (*string*) –版本控制系统（VCS）仓库的 URL

CURL 示例：

```
curl \
    --data-binary '{"name": "new name"}' \
    -H "Content-Type: application/json" \
    -H "Authorization: Token TOKEN" \
    PATCH http://example.com/api/projects/hello/components/
```

JSON 请求的示例：

```
PATCH /api/projects/hello/components/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{
    "name": "new name"
}
```

JSON 响应的示例：

```
HTTP/1.0 200 OK
Date: Tue, 12 Apr 2016 09:32:50 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, POST, HEAD, OPTIONS

{
    "branch": "main",
    "file_format": "po",
    "filemask": "po/*.po",
    "git_export": "",
    "license": "",
    "license_url": "",
    "name": "new name",
    "slug": "weblate",
    "project": {
        "name": "Hello",
        "slug": "hello",
        "source_language": {
            "code": "en",
            "direction": "ltr",
            "population": 159034349015,
            "name": "English",
            "url": "http://example.com/api/languages/en/",
            "web_url": "http://example.com/languages/en/"
```

(续下页)

(接上页)


```
        },
        "url": "http://example.com/api/projects/hello/",
        "web": "https://weblate.org/",
        "web_url": "http://example.com/projects/hello/"
    },
    "repo": "file:///home/nijel/work/weblate-hello",
    "template": "",
    "new_base": "",
    "url": "http://example.com/api/components/hello/weblate/",
    "vcs": "git",
    "web_url": "http://example.com/projects/hello/weblate/"
}
```


PUT /api/components/(string: *project*)/
string: *component*/

通过 PUT 请求编辑一个部件。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

请求 JSON 对象

- **branch** (*string*) –版本控制系统（VCS）仓库分支
- **file_format** (*string*) –翻译的文件格式
- **filemask** (*string*) –仓库中翻译的文件掩码
- **name** (*string*) –部件名称
- **slug** (*string*) –部件的标识串
- **repo** (*string*) –版本控制系统（VCS）仓库的 URL
- **template** (*string*) –但语言翻译的翻译模板文件
- **new_base** (*string*) –用于添加新翻译的翻译模板文件
- **vcs** (*string*) –版本控制系统

DELETE /api/components/(string: *project*)/
string: *component*/

在 3.9 版本加入.

删除部件。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

GET /api/components/(string: *project*)/
string: *component*/changes/

返回部件更改的列表。这本质是一个部件范围的*GET /api/changes/* 接受相同的参数。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

响应 JSON 对象

- **results** (*array*) – 部件对象的数组；请参见*GET /api/changes/(int:id)/*

GET /api/components/(string: *project*)/(string: *component*)/file/

在 4.9 版本加入.

使用请求的格式将与部件关联的所有可用翻译作为存档文件下载。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

查询参数

- **format** (*string*) –要使用的存档格式；如果未指定，默认为 `zip`; 支持的格式: `zip`

GET /api/components/(string: *project*)/(string: *component*)/screenshots/

返回部件屏幕截图的列表。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

响应 JSON 对象

- **results** (*array*) –部件屏幕截图的数组；请参见*GET /api/screenshots/(int:id)/*

GET /api/components/(string: *project*)/(string: *component*)/lock/

返回部件锁定状态。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

响应 JSON 对象

- **locked** (*boolean*) –部件是否因更新而锁定

示例 JSON 数据:

```
{
    "locked": false
}
```

POST /api/components/(string: *project*)/(string: *component*)/lock/

设置部件锁定状态。

响应时间与:http:get:‘/api/components/(string:project)/(string:component)/lock/‘相同。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

请求 JSON 对象

- **lock** –是否锁定的布尔值。

CURL 示例:

```
curl \
    -d lock=true \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/components/hello/weblate/repository/
```

JSON 请求的示例：

```
POST /api/components/hello/weblate/repository/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{"lock": true}
```

JSON 响应的示例：

```
HTTP/1.0 200 OK
Date: Tue, 12 Apr 2016 09:32:50 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, POST, HEAD, OPTIONS

{"locked":true}
```

GET /api/components/(string: *project*)/(string: *component*)/repository/

返回版本控制系统（VCS）仓库状态的信息。

响应与*GET /api/projects/(string:project)/repository/* 相同。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

响应 JSON 对象

- **needs_commit** (*boolean*) –是否有任何要提交的待处理更改
- **needs_merge** (*boolean*) –是否有上游更改要合并
- **needs_push** (*boolean*) –是否有本地更改要推送
- **remote_commit** (*string*) –远程提交说明
- **status** (*string*) –由版本控制系统（VCS）报告的 VCS 状态
- **merge_failure** –描述合并失败的文本，没有的话为空

POST /api/components/(string: *project*)/(string: *component*)/repository/

在版本控制系统（VCS）仓库执行给定的操作。

文档请参见*POST /api/projects/(string:project)/repository/*。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

请求 JSON 对象

- **operation** (*string*) –执行的操作：push, pull, commit, reset, cleanup 之一

响应 JSON 对象

- **result** (*boolean*) –操作的结果

CURL 示例：

```
curl \
    -d operation=pull \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/components/hello/weblate/repository/
```

JSON 请求的示例：

```
POST /api/components/hello/weblate/repository/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{"operation":"pull"}
```

JSON 响应的示例：

```
HTTP/1.0 200 OK
Date: Tue, 12 Apr 2016 09:32:50 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, POST, HEAD, OPTIONS

{"result":true}
```

GET /api/components/(string: *project*)/string: *component*/monolingual_base/

为单语言翻译下载翻译模板文件。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

GET /api/components/(string: *project*)/string: *component*/new_template/

为新的翻译下载模板文件。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

GET /api/components/(string: *project*)/string: *component*/translations/

返回给定部件中翻译对象的列表。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

响应 JSON 对象

- **results** (*array*) –翻译对象的数组；请参见*GET /api/translations/(string:project)/(string:component)/(string:language)/*

POST /api/components/(string: *project*)/string: *component*/translations/

在给定部件中新建新的翻译。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

请求 JSON 对象

- **language_code** (*string*) –翻译语言代码；请参见*GET /api/languages/(string:language)/*

响应 JSON 对象

- **result** (*object*) –新建的新翻译对象

CURL 示例：

```
curl \
    -d language_code=cs \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/projects/hello/components/
```

JSON 请求的示例：

```
POST /api/projects/hello/components/ HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Token TOKEN
Content-Length: 20

{"language_code": "cs"}
```

JSON 响应的示例：

```
HTTP/1.0 200 OK
Date: Tue, 12 Apr 2016 09:32:50 GMT
Server: WSGIServer/0.1 Python/2.7.11+
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Language: en
Allow: GET, POST, HEAD, OPTIONS

{
    "failing_checks": 0,
    "failing_checks_percent": 0,
    "failing_checks_words": 0,
    "filename": "po/cs.po",
    "fuzzy": 0,
    "fuzzy_percent": 0.0,
    "fuzzy_words": 0,
    "have_comment": 0,
    "have_suggestion": 0,
    "is_template": false,
    "is_source": false,
```

(续下页)

(接上页)


```
    "language": {
        "code": "cs",
        "direction": "ltr",
        "population": 1303174280
        "name": "Czech",
        "url": "http://example.com/api/languages/cs/",
        "web_url": "http://example.com/languages/cs/"
    },
    "language_code": "cs",
    "id": 125,
    "last_author": null,
    "last_change": null,
    "share_url": "http://example.com/engage/hello/cs/",
    "total": 4,
    "total_words": 15,
    "translate_url": "http://example.com/translate/hello/weblate/cs/",
    "translated": 0,
    "translated_percent": 0.0,
    "translated_words": 0,
    "url": "http://example.com/api/translations/hello/weblate/cs/",
    "web_url": "http://example.com/projects/hello/weblate/cs/"
}
```


GET /api/components/(string: *project*)/(string: *component*)/statistics/

对部件内所有的翻译返回分页的统计数据。

在 2.7 版本加入.

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

响应 JSON 对象

- **results** (*array*) – 翻译统计数据对象的数组; 请参见*GET /api/translations/(string:project)/(string:component)/(string:language)/statistics/*

GET /api/components/(string: *project*)/(string: *component*)/links/

返回与部件相连的项目。

在 4.5 版本加入.

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

响应 JSON 对象

- **projects** (*array*) – 相关的项目, 请参见*GET /api/projects/(string:project)/*

POST /api/components/(string: *project*)/(string: *component*)/links/

将项目与一个部件相关联。

在 4.5 版本加入.

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符

表单参数

- **string project_slug** –项目标识串

DELETE /api/components/(**string:** *project*)**/**
string: *component***/links/string:** *project_slug***/**

删除项目与部件的关联性。

在 4.5 版本加入.

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **project_slug** (*string*) –要移除的项目的地址

1.12.9 翻译

GET /api/translations/

返回翻译的列表。

参见:

翻译对象属性记录在*GET /api/translations/(string:project)/(string:component)/(string:language)/*。

GET /api/translations/(**string:** *project*)**/**
string: *component***/string:** *language***/**

返回翻译的信息。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

响应 JSON 对象

- **component** (*object*) – 部件对象；请参见*GET /api/components/(string:project)/(string:component)/*
- **failing_checks** (*int*) –未通过检查的字符串数量
- **failing_checks_percent** (*float*) –未通过检查的字符串百分比
- **failing_checks_words** (*int*) –带有未通过检查的单词数量
- **filename** (*string*) –翻译文件名
- **fuzzy** (*int*) –模糊（标记为需要编辑）字符串的数量
- **fuzzy_percent** (*float*) –模糊字符串（标记为需要编辑）的百分比
- **fuzzy_words** (*int*) –模糊（标记为编辑)）字符串中的单词数
- **have_comment** (*int*) –带有评论的字符串数量
- **have_suggestion** (*int*) –带有建议的字符串数量
- **is_template** (*boolean*) –译文是否有单语基础
- **language** (*object*) – 源语言对象；请参见*GET /api/languages/(string:language)/*

- **language_code** (*string*) –仓库中使用的语言代码；这可以不同于语言对象中的语言代码
- **last_author** (*string*) –最后一位作者的姓名
- **last_change** (*timestamp*) –最后更改的时间戳
- **revision** (*string*) –文件的修订哈希值
- **share_url** (*string*) –用于分享的 URL，指向参与页面
- **total** (*int*) –字符串的总数
- **total_words** (*int*) –词的总数
- **translate_url** (*string*) –翻译的 URL
- **translated** (*int*) –已翻译字符串的数量
- **translated_percent** (*float*) –已翻译字符串的百分比
- **translated_words** (*int*) –已翻译词的数量
- **repository_url** (*string*) – 仓库状态的 URL；请参见*GET /api/translations/(string:project)/(string:component)/(string:language)/repository/*
- **file_url** (*string*) –文件对象的 URL；请参见*GET /api/translations/(string:project)/(string:component)/(string:language)/file/*
- **changes_list_url** (*string*) – 更改的列表的 URL；请参见*GET /api/translations/(string:project)/(string:component)/(string:language)/changes/*
- **units_list_url** (*string*) – 字符串列表的 URL；请参见*GET /api/translations/(string:project)/(string:component)/(string:language)/units/*

示例 JSON 数据：


```
{
    "component": {
        "branch": "main",
        "file_format": "po",
        "filemask": "po/*.po",
        "git_export": "",
        "license": "",
        "license_url": "",
        "name": "Weblate",
        "new_base": "",
        "project": {
            "name": "Hello",
            "slug": "hello",
            "source_language": {
                "code": "en",
                "direction": "ltr",
                "population": 159034349015,
                "name": "English",
                "url": "http://example.com/api/languages/en/",
                "web_url": "http://example.com/languages/en/"
            },
            "url": "http://example.com/api/projects/hello/",
            "web": "https://weblate.org/",
            "web_url": "http://example.com/projects/hello/"
        },
        "repo": "file:///home/nijel/work/weblate-hello",
```


(续下页)

(接上页)


```
        "slug": "weblate",
        "template": "",
        "url": "http://example.com/api/components/hello/weblate/",
        "vcs": "git",
        "web_url": "http://example.com/projects/hello/weblate/"
    },
    "failing_checks": 3,
    "failing_checks_percent": 75.0,
    "failing_checks_words": 11,
    "filename": "po/cs.po",
    "fuzzy": 0,
    "fuzzy_percent": 0.0,
    "fuzzy_words": 0,
    "have_comment": 0,
    "have_suggestion": 0,
    "is_template": false,
    "language": {
        "code": "cs",
        "direction": "ltr",
        "population": 1303174280
        "name": "Czech",
        "url": "http://example.com/api/languages/cs/",
        "web_url": "http://example.com/languages/cs/"
    },
    "language_code": "cs",
    "last_author": "Weblate Admin",
    "last_change": "2016-03-07T10:20:05.499",
    "revision": "7ddfafe6daaf57fc8654cc852ea6be212b015792",
    "share_url": "http://example.com/engage/hello/cs/",
    "total": 4,
    "total_words": 15,
    "translate_url": "http://example.com/translate/hello/weblate/cs/",
    "translated": 4,
    "translated_percent": 100.0,
    "translated_words": 15,
    "url": "http://example.com/api/translations/hello/weblate/cs/",
    "web_url": "http://example.com/projects/hello/weblate/cs/"
}
```


DELETE /api/translations/(**string**: *project*)**/**
string: *component***/string**: *language***/**

在 3.9 版本加入.

删除翻译。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

GET /api/translations/(**string**: *project*)**/**
string: *component***/string**: *language***/changes/**

返回翻译更改的列表。这本质上是一个翻译范围的*GET /api/changes/*接受相同的参数。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

响应 JSON 对象

- **results** (*array*) – 部件对象的数组；请参见*GET /api/changes/(int:id)/*

GET /api/translations/(string: *project*)/string: *component*/string: *language*/units/

返回翻译单元的列表。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码
- **q** (*string*) –搜索查询字符串搜索 （可选）

响应 JSON 对象

- **results** (*array*) –部件对象的数组；请参见*GET /api/units/(int:id)/*

POST /api/translations/(string: *project*)/string: *component*/string: *language*/units/

添加新单元。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

请求 JSON 对象

- **key** (*string*) –翻译单元的名称（用作键或上下文）
- **value** (*array*) –源字符串（如果不创建复数则使用单条字符串）
- **state** (*int*) –字符串状态；见*GET /api/units/(int:id)/*

响应 JSON 对象

- **unit** (*object*) –新创建的单元；请参见*GET /api/units/(int:id)/*

参见:

管理字符串, adding-new-strings

POST /api/translations/(string: *project*)/string: *component*/string: *language*/autotranslate/

触发自动翻译。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

请求 JSON 对象

- **mode** (*string*) –自动翻译模式
- **filter_type** (*string*) –自动翻译筛选类型
- **auto_source** (*string*) –自动翻译的来源 - `mt` 或 `others`
- **component** (*string*) –为项目打开对共享翻译记忆库的贡献，以访问其他部件。
- **engines** (*array*) –机器翻译引擎

- **threshold** (*string*) –匹配分数阈值

GET /api/translations/(**string:** *project*)**/**
string: *component***/string:** *language***/file/**

下载存储在版本控制系统（VCS）中的当前翻译文件（不带 format 参数）或将其转换为另一格式（见下载译文）。

备注: 这个 API 端点使用了不同于 API 其余的逻辑来输出，它在整个文件而不是在数据上操作。接受的 format 参数组不同，没有这个参数会将翻译文件存储在版本控制系统（VCS）中。

查询参数

- **format** –使用的文件格式；如果不指定，则不会进行格式转换；支持的文件格式有：po, mo, xliff, xliff11, tbx, tmx, csv, xlsx, json, aresource, strings
- **q** (*string*) –过滤已下载的字符串，见 search，仅在对话就绪时适用（指定了 format）。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

POST /api/translations/(**string:** *project*)**/**
string: *component***/string:** *language***/file/**

上传带有翻译的新文件。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

表单参数

- **string conflicts** –如何处理冲突（ignore、replace-translated 或 replace-approved）
- **file file** –上传文件
- **string email** –作者电子邮箱
- **string author** –作者姓名
- **string method** –上传方法（translate、approve、suggest、fuzzy、replace、source、add），请参见导入方式
- **string fuzzy** –模糊（标记为需要编辑）的字符串处理（*empty*, process, approve）

CURL 示例:

```
curl -X POST \
    -F file=@strings.xml \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/translations/hello/android/cs/file/
```

GET /api/translations/(**string:** *project*)**/**
string: *component***/string:** *language***/repository/**

返回版本控制系统（VCS）仓库状态的信息。

响应与*GET /api/components/(string:project)/(string:component)/repository/*相同。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

POST /api/translations/(string: *project*)/string: *component*/string: *language*/repository/

在版本控制系统（VCS）仓库上执行给定的操作。

文档请参见*POST /api/projects/(string:project)/repository/*。

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

请求 JSON 对象

- **operation** (*string*) –执行的操作：`push`, `pull`, `commit`, `reset`, `cleanup` 之一

响应 JSON 对象

- **result** (*boolean*) –操作的结果

GET /api/translations/(string: *project*)/string: *component*/string: *language*/statistics/

返回具体的翻译统计数据。

在 2.7 版本加入.

参数

- **project** (*string*) –项目 URL 标识符
- **component** (*string*) –部件 URL 标识符
- **language** (*string*) –翻译语言代码

响应 JSON 对象

- **code** (*string*) –语言代码
- **failing** (*int*) –未通过检查的数量
- **failing_percent** (*float*) –未通过检查的百分比
- **fuzzy** (*int*) –模糊（标记为需要编辑）字符串的数量
- **fuzzy_percent** (*float*) –模糊字符串（标记为需要编辑）的百分比
- **total_words** (*int*) –词的总数
- **translated_words** (*int*) –已翻译词的数量
- **last_author** (*string*) –最后一位作者的姓名
- **last_change** (*timestamp*) –最后一次更改的日期
- **name** (*string*) –语言名称
- **total** (*int*) –字符串的总数

- **translated** (*int*) –已翻译字符串的数量
- **translated_percent** (*float*) –已翻译字符串的百分比
- **url** (*string*) –访问翻译的 URL（参与 URL）
- **url_translate** (*string*) –访问翻译的 URL （真实翻译的 URL）

1.12.10 记忆存储

在 4.14 版本加入.

GET /api/memory/

返回记忆结果列表。

DELETE /api/memory/(**int:** *memory_object_id*)**/**

删除记忆项目

参数

- **memory_object_id** –内存对象 ID

1.12.11 单元

unit'是翻译的一个单件它将一个源字符串与一个相应的翻译字符串配对，还包含一些相关的元数据。该术语源自 'Translate Toolkit 和 XLIFF。

在 2.10 版本加入.

GET /api/units/

返回翻译单元的列表。

参数

- **q** (*string*) –搜索查询字符串搜索 （可选）

参见:

单元对象属性记录在*GET /api/units/(int:id)/*。

GET /api/units/(**int:** *id*)**/**

在 4.3 版本发生变更: `target` 和 `source` 现在是数组了，可以正确处理复数字符串。

返回翻译单元的信息。

参数

- **id** (*int*) –单元 ID

响应 JSON 对象

- **translation** (*string*) –相关翻译对象的 URL
- **source** (*array*) –源字符串
- **previous_source** (*string*) –用于模糊匹配的之前的源字符串
- **target** (*array*) –目标字符串
- **id_hash** (*string*) –单元的唯一标识符
- **content_hash** (*string*) –源字符串的唯一标识符
- **location** (*string*) –源代码中单元的位置
- **context** (*string*) –翻译单元的上下文
- **note** (*string*) –翻译单元的注解

- **flags** (*string*) –翻译单元的标记
- **labels** (*array*) –译文单元标签，在原文单元上可用
- **state** (*int*) –单元状态，0——未翻译、10——需要编辑、20——已翻译、30——已核准、100——只读
- **fuzzy** (*boolean*) –该单元是模糊的还是标记为需要复查
- **translated** (*boolean*) –单元是否被翻译
- **approved** (*boolean*) –译文是否已核准
- **position** (*int*) –单元在翻译文件中的位次
- **has_suggestion** (*boolean*) –单元是否有翻译建议
- **has_comment** (*boolean*) –单元是否有评论
- **has_failing_check** (*boolean*) –单元是否有未通过的检查
- **num_words** (*int*) –原文单词数
- **priority** (*int*) –翻译优先级；100 为默认值
- **id** (*int*) –单元的标识符
- **explanation** (*string*) –字符串的解释，可在源单元获得，请参见源字符串另外的信息
- **extra_flags** (*string*) –另外的字符串标记，可在源单元获得，请参见使用标记定制行为
- **web_url** (*string*) –单元可以被编辑的 URL
- **source_unit** (*string*) –源单元链接；请参见*GET /api/units/(int:id)/*
- **pending** (*boolean*) –该单元是否待写入
- **timestamp** (*timestamp*) –字符串添加时间

PATCH /api/units/(**int:** *id*)**/**

在 4.3 版本加入.

对翻译单元执行部分更新。

参数

- **id** (*int*) –单元 ID

请求 JSON 对象

- **state** (*int*) –单元状态，0——未翻译、10——需要编辑、20——已翻译、30——已核准（需启用审校流程，请参见专门的审校员）
- **target** (*array*) –目标字符串
- **explanation** (*string*) –字符串的解释，可在源单元获得，请参见源字符串另外的信息
- **extra_flags** (*string*) –另外的字符串标记，可在源单元获得，请参见使用标记定制行为

响应 JSON 对象

- **labels** (*array*) –标签，在源单元上可用

PUT /api/units/(**int:** *id*)**/**

在 4.3 版本加入.

对翻译单元执行完整翻译。

参数

- **id** (*int*) –单元 ID

请求 JSON 对象

- **state** (*int*) –单元状态，0——未翻译、10——需要编辑、20——已翻译、30——已核准（需启用审校流程，请参见专门的审校员）
- **target** (*array*) –目标字符串
- **explanation** (*string*) –字符串的解释，可在源单元获得，请参见源字符串另外的信息
- **extra_flags** (*string*) –另外的字符串标记，可在源单元获得，请参见使用标记定制行为

响应 JSON 对象

- **labels** (*array*) –标签，在源单元上可用

DELETE /api/units/(int: *id*)/

在 4.3 版本加入.

删除一个翻译单元。

参数

- **id** (*int*) –单元 ID

1.12.12 变化

在 2.10 版本加入.

GET /api/changes/

在 4.1 版本发生变更: 更改的筛选在 4.1 版本引入。

返回翻译更改的列表。

参见:

更改对象属性记录在*GET /api/changes/(int:id)/*。

查询参数

- **user** (*string*) –筛选用户的用户名
- **action** (*int*) –操作筛选，可以多次使用
- **timestamp_after** (*timestamp*) –ISO 8601 格式的时间戳，列出此时间之后的更改
- **timestamp_before** (*timestamp*) –ISO 8601 格式的时间戳，列出此时间之前的更改

GET /api/changes/(int: *id*)/

返回有关翻译更改的信息。

参数

- **id** (*int*) –更改的 ID

响应 JSON 对象

- **unit** (*string*) –相关单元对象的 URL
- **translation** (*string*) –相关翻译对象的 URL
- **component** (*string*) –相关部件对象的 URL
- **user** (*string*) –相关用户对象的 URL

- **author** (*string*) –相关作者对象的 URL
- **timestamp** (*timestamp*) –事件时间戳
- **action** (*int*) –操作的数字标识
- **action_name** (*string*) –操作的文本描述
- **target** (*string*) –更改的事件的文本或细节
- **id** (*int*) –变更的标识符

1.12.13 截屏

在 2.14 版本加入.

GET /api/screenshots/

返回屏幕截图字符串信息的列表。

参见:

屏幕截图对象属性记录在*GET /api/screenshots/(int:id)/*。

GET /api/screenshots/(int: *id*)/

返回与屏幕截图信息有关的信息。

参数

- **id** (*int*) –截图 ID

响应 JSON 对象

- **name** (*string*) –屏幕截图的名称
- **component** (*string*) –相关部件对象的 URL
- **file_url** (*string*) –下载文件的 URL；请参见*GET /api/screenshots/(int:id)/file/*
- **units** (*array*) –与源字符串信息相关的链接；请参见*GET /api/units/(int:id)/*

GET /api/screenshots/(int: *id*)/file/

下载屏幕截图的图片。

参数

- **id** (*int*) –截图 ID

POST /api/screenshots/(int: *id*)/file/

替换屏幕截图。

参数

- **id** (*int*) –截图 ID

表单参数

- **file image** –上传文件

CURL 示例:

```
curl -X POST \
    -F image=@image.png \
    -H "Authorization: Token TOKEN" \
    http://example.com/api/screenshots/1/file/
```

POST /api/screenshots/(int: *id*)/units/

与屏幕截图相关的源字符串。

参数

- **id** (*int*) – 截图 ID

表单参数

- **string unit_id** – 单元 ID

响应 JSON 对象

- **name** (*string*) – 屏幕截图的名称
- **translation** (*string*) – 相关翻译对象的 URL
- **file_url** (*string*) – 下载文件的 URL；请参见*GET /api/screenshots/(int:id)/file/*
- **units** (*array*) – 与源字符串信息相关的链接；请参见*GET /api/units/(int:id)/*

DELETE /api/screenshots/(int: *id*)/units/
int: *unit_id*

删除与截图关联的源字符串。

参数

- **id** (*int*) – 截图 ID
- **unit_id** – 源字符串单元 ID

POST /api/screenshots/

新建新的屏幕截图。

表单参数

- **file image** – 上传文件
- **string name** – 截图名称
- **string project_slug** – 项目标识串
- **string component_slug** – 部件标识串
- **string language_code** – 语言代码

响应 JSON 对象

- **name** (*string*) – 屏幕截图的名称
- **component** (*string*) – 相关部件对象的 URL
- **file_url** (*string*) – 下载文件的 URL；请参见*GET /api/screenshots/(int:id)/file/*
- **units** (*array*) – 与源字符串信息相关的链接；请参见*GET /api/units/(int:id)/*

PATCH /api/screenshots/(int: *id*)/

编辑截屏的部分信息。

参数

- **id** (*int*) – 截图 ID

响应 JSON 对象

- **name** (*string*) – 屏幕截图的名称
- **component** (*string*) – 相关部件对象的 URL

- **file_url** (*string*) –下载文件的 URL；请参见*GET /api/screenshots/(int:id)/file/*
- **units** (*array*) –与源字符串信息相关的链接；请参见*GET /api/units/(int:id)/*

PUT /api/screenshots/(int: *id*)/

编辑截屏的完整信息。

参数

- **id** (*int*) –截图 ID

响应 JSON 对象

- **name** (*string*) –屏幕截图的名称
- **component** (*string*) –相关部件对象的 URL
- **file_url** (*string*) –下载文件的 URL；请参见*GET /api/screenshots/(int:id)/file/*
- **units** (*array*) –与源字符串信息相关的链接；请参见*GET /api/units/(int:id)/*

DELETE /api/screenshots/(int: *id*)/

删除截图。

参数

- **id** (*int*) –截图 ID

1.12.14 附加组件

在 4.4.1 版本加入.

GET /api/addons/

返回附加组件的列表。

参见:

附加组件对象属性记录在*GET /api/units/(int:id)/*。

GET /api/addons/(int: *id*)/

返回有关附加组件的信息。

参数

- **id** (*int*) –附加组件 ID

响应 JSON 对象

- **name** (*string*) –附加组件名称
- **component** (*string*) –相关部件对象的 URL
- **configuration** (*object*) –可选的附加组件配置

参见:

附加组件

POST /api/components/(string: *project*)/
string: *component*/addons/

创建新附加组件。

参数

- **project_slug** (*string*) –项目标识串

- **component_slug** (*string*) –部件标识串

请求 JSON 对象

- **name** (*string*) –附加组件名称
- **configuration** (*object*) –可选的附加组件配置

PATCH /api/addons/(**int:** *id*)**/**

编辑附加组件的部分信息。

参数

- **id** (*int*) –附加组件 ID

响应 JSON 对象

- **configuration** (*object*) –可选的附加组件配置

PUT /api/addons/(**int:** *id*)**/**

编辑附加组件的完整信息。

参数

- **id** (*int*) –附加组件 ID

响应 JSON 对象

- **configuration** (*object*) –可选的附加组件配置

DELETE /api/addons/(**int:** *id*)**/**

删除附加组件。

参数

- **id** (*int*) –附加组件 ID

1.12.15 部件列表

在 4.0 版本加入.

GET /api/component-lists/

返回一个部件列表的列表。

参见:

部件列表对象属性记录在*GET /api/component-lists/(str:slug)/*。

GET /api/component-lists/(**str:** *slug*)**/**

返回部件列表的信息。

参数

- **slug** (*string*) –部件列表的标识串

响应 JSON 对象

- **name** (*string*) –部件列表的名称
- **slug** (*string*) –部件列表的表示串
- **show_dashboard** (*boolean*) –是否在操作面板上显示
- **components** (*array*) –相关联部件的连接；请参见*GET /api/components/(string:project)/(string:component)/*
- **auto_assign** (*array*) –自动分配规则

PUT /api/component-lists/(str: *slug*)/

更改部件列表参数。

参数

- **slug** (*string*) –部件列表的标识串

请求 JSON 对象

- **name** (*string*) –部件列表的名称
- **slug** (*string*) –部件列表的表示串
- **show_dashboard** (*boolean*) –是否在操作面板上显示

PATCH /api/component-lists/(str: *slug*)/

更改部件列表参数。

参数

- **slug** (*string*) –部件列表的标识串

请求 JSON 对象

- **name** (*string*) –部件列表的名称
- **slug** (*string*) –部件列表的表示串
- **show_dashboard** (*boolean*) –是否在操作面板上显示

DELETE /api/component-lists/(str: *slug*)/

删除部件列表。

参数

- **slug** (*string*) –部件列表的标识串

POST /api/component-lists/(str: *slug*)/components/

使部件与部件列表相关。

参数

- **slug** (*string*) –部件列表的标识串

表单参数

- **string component_id** –部件 ID

DELETE /api/component-lists/(str: *slug*)/components/
str: *component_slug*

取消部件与部件列表的关联。

参数

- **slug** (*string*) –部件列表的标识串
- **component_slug** (*string*) –部件标识串

1.12.16 术语表

在 4.5 版本发生变更: 术语表现在存储为常规部件、翻译和字符串，请改用相应的 API。

1.12.17 任务

在 4.4 版本加入.

GET /api/tasks/
任务列表当前不可用。

GET /api/tasks/(str: *uuid*)/
返回任务信息

参数

- **uuid** (*string*) –任务 UUID

响应 JSON 对象

- **completed** (*boolean*) –任务是否已完成
- **progress** (*int*) –任务进度百分比
- **result** (*object*) –任务结果或过程细节
- **log** (*string*) –任务日志

1.12.18 指标

GET /api/metrics/
返回服务器指标。

响应 JSON 对象

- **units** (*int*) –单元数量
- **units_translated** (*int*) –已翻译单元数量
- **users** (*int*) –用户数量
- **changes** (*int*) –更改次数
- **projects** (*int*) –项目数量
- **components** (*int*) –部件数量
- **translations** (*int*) –翻译数量
- **languages** (*int*) –所用语言数量
- **checks** (*int*) –触发的质量检查数
- **configuration_errors** (*int*) –配置错误的数量
- **suggestions** (*int*) –待处理建议数量
- **celery_queues** (*object*) –Celery 队列长度，见使用 *Celery* 的后台任务
- **name** (*string*) –配置的服务器名

1.12.19 通知钩子

通知钩子允许外部应用来通知 Weblate 版本控制系统（VCS）仓库已经更新。

可以为项目、部件和翻译使用仓库端点来更新各自的仓库；文档请参见*POST /api/projects/(string:project)/repository/*。

GET /hooks/update/(string: *project*)/
string: *component*/

自 2.6 版本弃用: 请使用*POST /api/components/(string:project)/(string:component)/repository/* 来替代，它使用 ACL 限制的身份验证而工作正常。

触发部件的更新（从版本控制系统 VCS 拉取并扫描翻译的更改）。

GET /hooks/update/(string: *project*)/

自 2.6 版本弃用: 请使用*POST /api/projects/(string:project)/repository/* 来替代，它使用 ACL 限制的身份验证而工作正常。

触发项目中所有部件的更新（从版本控制系统 VCS 拉取并扫描翻译的更改）。

POST /hooks/github/

处理 Github 通知与自动更新匹配部件的特殊钩子。

备注: Github 包括了对通知 Weblate 的直接支持：在仓库设置中启动 Weblate 服务钩子，并将 URL 设置为你的 Weblate 安装的 URL。

参见:

从 *GitHub* 自动接收更改
关于设置 Github 集成的指令

https://docs.github.com/en/get-started/customizing-your-github-workflow/exploring-integrations/about-webhooks
GitHub Webhooks 的一般信息

ENABLE_HOOKS
关于对整个 Weblate 启动钩子

POST /hooks/gitlab/

处理 GitLab 通知并自动更新匹配部件的特殊钩子。

参见:

从 *GitLab* 自动接收更改
关于设置 GitLab 集成的指示

https://docs.gitlab.com/ee/user/project/integrations/webhooks.html
关于 GitLab Webhooks 的一般信息

ENABLE_HOOKS
关于对整个 Weblate 启动钩子

POST /hooks/bitbucket/

处理 Bitbucket 通知并自动更新匹配的部件的特殊钩子。

参见:

从 *Bitbucket* 自动接收更改
关于设置 Bitbucket 集成的指示

https://support.atlassian.com/bitbucket-cloud/docs/manage-webhooks/
关于 Bitbucket Webhooks 的一般信息

ENABLE_HOOKS
关于对整个 Weblate 启动钩子

POST /hooks/pagure/

在 3.3 版本加入.

处理 Pagure 通知并自动更新匹配的部件的特殊钩子。

参见:

从 *Pagure* 自动接受更改
关于设置 Pagure 集成的指示

https://docs.pagure.org/pagure/usage/using_webhooks.html
关于 Pagure Webhooks 的一般信息

ENABLE_HOOKS
关于对整个 Weblate 启动钩子

POST /hooks/azure/

在 3.8 版本加入.

处理 Azure DevOps 通知并自动更新匹配的部件的特殊钩子。

备注: 请确保:guilabel:‘Resource details to send‘设置为 *All*，否则 Weblate 将无法匹配你的 Azure 仓库。

参见:

从 *Azure Repos* 自动接收更改
关于设置 Azure 集成的指示

https://learn.microsoft.com/en-us/azure/devops/service-hooks/services/webhooks?view=azure-devops
关于 Azure DevOps Web Hooks 的一般信息

ENABLE_HOOKS
关于对整个 Weblate 启动钩子

POST /hooks/gitea/

在 3.9 版本加入.

处理 Gitea Webhook 通知并自动更新匹配的部件的特殊钩子。

参见:

从 *Gitea Repos* 自动接收更改
关于设置 Gitea 集成的指示

https://docs.gitea.io/en-us/webhooks/
关于 Gitea Webhooks 的一般信息

ENABLE_HOOKS
关于对整个 Weblate 启动钩子

POST /hooks/gitee/

在 3.9 版本加入.

处理 Gitee Webhook 通知并自动更新匹配的部件的特殊钩子。

参见:

从 Gitee Repos 自动接收更改
关于设置 Gitee 集成的指示

https://gitee.com/help/categories/40
关于 Gitee Webhooks 的一般信息

ENABLE_HOOKS
关于对整个 Weblate 启动钩子

1.12.20 导出

Weblate 提供各种导出，允许进一步处理数据。

GET /exports/stats/(string: *project*)/
string: *component*/

查询参数

- **format** (*string*) –输出格式：json 或 csv

自 2.6 版本弃用: 请替代使用*GET /api/components/(string:project)/(string:component)/statistics/* 和*GET /api/translations/(string:project)/(string:component)/(string:language)/statistics/*；它也允许访问 ACL 控制的项目。

为给定的部件以给定的格式检索统计数据。

示例请求:

```
GET /exports/stats/weblate/main/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

示例响应:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

[
    {
        "code": "cs",
        "failing": 0,
        "failing_percent": 0.0,
        "fuzzy": 0,
        "fuzzy_percent": 0.0,
        "last_author": "Michal Čihař",
        "last_change": "2012-03-28T15:07:38+00:00",
        "name": "Czech",
        "total": 436,
        "total_words": 15271,
        "translated": 436,
        "translated_percent": 100.0,
        "translated_words": 3201,
        "url": "http://hosted.weblate.org/engage/weblate/cs/",
        "url_translate": "http://hosted.weblate.org/projects/weblate/main/cs/"
```

(续下页)

(接上页)


```
    },
    {
        "code": "nl",
        "failing": 21,
        "failing_percent": 4.8,
        "fuzzy": 11,
        "fuzzy_percent": 2.5,
        "last_author": null,
        "last_change": null,
        "name": "Dutch",
        "total": 436,
        "total_words": 15271,
        "translated": 319,
        "translated_percent": 73.2,
        "translated_words": 3201,
        "url": "http://hosted.weblate.org/engage/weblate/nl/",
        "url_translate": "http://hosted.weblate.org/projects/weblate/main/nl/"
    },
    {
        "code": "el",
        "failing": 11,
        "failing_percent": 2.5,
        "fuzzy": 21,
        "fuzzy_percent": 4.8,
        "last_author": null,
        "last_change": null,
        "name": "Greek",
        "total": 436,
        "total_words": 15271,
        "translated": 312,
        "translated_percent": 71.6,
        "translated_words": 3201,
        "url": "http://hosted.weblate.org/engage/weblate/el/",
        "url_translate": "http://hosted.weblate.org/projects/weblate/main/el/"
    }
]
```


1.12.21 RSS 订阅源

翻译的变更会导出到 RSS 订阅源中。

GET /exports/rss/(**string:** *project*)**/**
string: *component***/string:** *language***/**
返回带有翻译近期变更的 RSS 订阅源。

GET /exports/rss/(**string:** *project*)**/**
string: *component***/**
返回带有部件近期变更的 RSS 订阅源。

GET /exports/rss/(**string:** *project*)**/**
返回带有项目近期变更的 RSS 订阅源。

GET /exports/rss/language/(**string:** *language*)**/**
返回带有语言近期变更的 RSS 订阅源。

GET /exports/rss/
返回带有 Weblate 实例近期变更的 RSS 订阅源。

参见:

维基百科上的 RSS（英文）

1.13 Weblate 客户端

在 2.7 版本加入: 自从 Weblate 2.7 以来，已经有完整的 wlc 实用程序支持。如果您使用的是旧版本，则可能会与 API 发生某些不兼容。

1.13.1 安装

Weblate 客户端是单独提供的，包括 Python 模块。要使用下面的命令，您需要安装 *wlc*：

```
pip install wlc
```

1.13.2 Docker 用法

Weblate 客户端也可以用作 Docker 映像。

该映像发布在 Docker Hub：https://hub.docker.com/r/weblate/wlc

正在安装：

```
docker pull weblate/wlc
```

Docker 容器使用 Weblate 的默认设置并连接到部署在 localhost 中的 API。可以通过 Weblate 接受的参数进行配置 API URL 和 API_KEY。

启动容器的命令使用以下语法：

```
docker run --rm weblate/wlc [WLC_ARGS]
```

示例：

```
docker run --rm weblate/wlc --url https://hosted.weblate.org/api/ list-projects
```

您可能希望将配置文件 传递给 Docker 容器，最简单的方法是将当前目录添加为 `/home/weblate` 卷：

```
docker run --volume $PWD:/home/weblate --rm weblate/wlc show
```

1.13.3 入门

wlc 配置存储在 `~/.config/weblate` 中（其他位置请参见配置文件），请根据你的环境创建它：

```
[weblate]
url = https://hosted.weblate.org/api/

[keys]
https://hosted.weblate.org/api/ = APIKEY
```

然后，您可以在默认服务器上调用命令：

```
wlc ls
wlc commit sandbox/hello-world
```

参见:

配置文件

1.13.4 概要

```
wlc [arguments] <command> [options]
```

命令实际上指示应该执行哪个操作。

1.13.5 说明

Weblate 客户端是一个 Python 库和命令行实用程序，可使用*Weblate 的 REST API* 远程管理 Weblate。命令行实用程序可以作为 **wlc** 调用，并且内置在*wlc* 上。

参数

程序接受以下参数来定义输出格式或使用哪个 Weblate 实例。这些参数必须位于任何命令之前。

--format {csv,json,text,html}
: 指定输出格式。

--url URL
: 指定 API URL。覆盖在配置文件中找到的任何值，请参阅配置文件。该网址应以 /api/ 结尾，例如 https://hosted.weblate.org/api/。

--key KEY
: 指定要使用的 API 用户密钥。覆盖在配置文件中找到的任何值，请参阅配置文件。您可以在 Weblate 的个人资料中找到密钥。

--config PATH
: 覆盖配置文件路径，请参阅配置文件。

--config-section SECTION
: 覆盖正在使用的配置文件部分，请参阅配置文件。

命令

以下命令可用：

version
: 打印当前版本。

list-languages
: 列出 Weblate 中使用的语言。

list-projects
: 列出 Weblate 中的项目。

list-components
: 列出 Weblate 中的部件。

list-translations
: 列出 Weblate 中的翻译。

show
: 显示 Weblate 对象（翻译，部件或项目）。

ls
: 列出 Weblate 对象（翻译，部件或项目）。

commit
提交在 Weblate 对象（翻译，部件或项目）中所做的更改。

pull
拉取远程仓库的更改到 Weblate 对象中（翻译，部件或项目）。

push
将 Weblate 对象更改推送到远程仓库（翻译，部件或项目）。

reset
在 0.7 版本加入: 自 wlc 0.7 起受支持。

重置 Weblate 对象中的更改以匹配远程仓库（翻译，部件或项目）。

cleanup
在 0.9 版本加入: 从 wlc 0.9 开始支持。

删除 Weblate 对象中所有未跟踪的更改以匹配远程仓库（翻译，部件或项目）。

repo
显示给定 Weblate 对象（翻译，部件或项目）的仓库状态。

stats
显示给定 Weblate 对象（翻译，部件或项目）的详细统计数据。

lock-status
在 0.5 版本加入: 自 wlc 0.5 起受支持。

显示锁定状态。

lock
在 0.5 版本加入: 自 wlc 0.5 起受支持。

锁定部件以防止在 Weblate 中进一步翻译。

unlock
在 0.5 版本加入: 自 wlc 0.5 起受支持。

解锁 Weblate 部件的翻译。

changes
在 0.7 版本加入: 从 wlc 0.7 和 Weblate 2.10 开始支持。

显示给定对象的更改。

download
在 0.7 版本加入: 自 wlc 0.7 起受支持。

下载翻译文件。

--convert
转换文件格式，如果未指定，则在服务器上不进行任何转换，并且将文件原样下载到仓库中。

--output
指定要保存输出的文件，如果未指定，则将其打印到 stdout。

upload
在 0.9 版本加入: 从 wlc 0.9 开始支持。

上传翻译文件。

--overwrite
上传时覆盖现有翻译。

--input
从中读取内容的文件，如果未指定，则从 stdin 中读取。

--method
要使用的上传方法，参见导入方式。

--fuzzy
模糊（标记为需要编辑）的字符串处理（*empty*, `process`, `approve`）

--author-name
作者姓名，以覆盖当前经过身份验证的用户

--author-email
作者电子邮件，以覆盖当前经过身份验证的用户

提示： 您可以通过传递 `--help` 获得有关调用单个命令的更多详细信息，例如：`wlc ls --help` 。

1.13.6 配置文件

.weblate, .weblate.ini, weblate.ini
在 1.6 版本发生变更: 同时支持扩展名为 '.ini'的文件。

每个项目的配置文件

C:\Users\NAME\AppData\weblate.ini
在 1.6 版本加入.

Windows 上的用户配置文件。

~/.config/weblate
用户配置文件

/etc/xdg/weblate
系统范围的配置文件

该程序遵循 XDG 规范，因此您可以通过环境变量 `XDG_CONFIG_HOME` 或 `XDG_CONFIG_DIRS` 来调整配置文件的位置。在 Windows 系统上'APPDATA"目录是配置文件的首选位置。

可以在 `[weblate]` 部分中配置以下设置（您可以通过*`--config-section`* 进行自定义）：

key
用于访问 Weblate 的 API KEY。

url
API 服务器网址，默认为 `http://127.0.0.1:8000/api/`。

translation
默认翻译的路径——部件或项目。

配置文件是一个 INI 文件，例如：

```
[weblate]
url = https://hosted.weblate.org/api/
key = APIKEY
translation = weblate/application
```

另外，API 密钥可以存储在 `[keys]` 部分中：

```
[keys]
https://hosted.weblate.org/api/ = APIKEY
```

这样，您就可以在版本控制系统（VCS）仓库中使用 `.weblate` 配置时，将密钥存储在个人设置中，以便 wlc 知道它应该与哪个服务器通信。

1.13.7 示例

打印当前程序版本：

```
$ wlc version
version: 0.1
```

列出所有项目：

```
$ wlc list-projects
name: Hello
slug: hello
url: http://example.com/api/projects/hello/
web: https://weblate.org/
web_url: http://example.com/projects/hello/
```

上传翻译文件：

```
$ wlc upload project/component/language --input /tmp/hello.po
```

您还可以指定 wlc 应该从事的项目：

```
$ cat .weblate
[weblate]
url = https://hosted.weblate.org/api/
translation = weblate/application

$ wlc show
branch: main
file_format: po
source_language: en
filemask: weblate/locale/*/LC_MESSAGES/django.po
git_export: https://hosted.weblate.org/git/weblate/application/
license: GPL-3.0+
license_url: https://spdx.org/licenses/GPL-3.0+
name: Application
new_base: weblate/locale/django.pot
project: weblate
repo: git://github.com/WeblateOrg/weblate.git
slug: application
template:
url: https://hosted.weblate.org/api/components/weblate/application/
vcs: git
web_url: https://hosted.weblate.org/projects/weblate/application/
```

通过此设置，可以轻松地提交当前项目中待处理的更改：

```
$ wlc commit
```

1.14 Weblate 的 Python API

1.14.1 安装

Python API 是单独发布的，你需要安装 *Weblate 客户端* (wlc) 才能拥有它。

```
pip install wlc
```

1.14.2 wlc

WeblateException

exception wlc.**WeblateException**
所有异常的基类。

Weblate

class wlc.**Weblate**(*key=''*, *url=None*, *config=None*)

参数

- **key** (*str*) –用户密钥
- **url** (*str*) –API 服务器 URL，如果未指定则使用默认值
- **config** (wlc.config.WeblateConfig) –配置对象，覆盖任何其他参数。

API 的访问类，定义 API 密钥和可选的 API URL。

get(*path*)

参数
path (*str*) –请求路径

返回类型
object

执行单个 API GET 调用。

post(*path*, ***kwargs*)

参数
path (*str*) –请求路径

返回类型
object

执行单个 API GET 调用。

1.14.3 wlc.config

WeblateConfig

class wlc.config.**WeblateConfig**(*section='wlc'*)

参数
section (*str*) –要使用的配置部分

遵循 XDG 规范的配置文件解析器。

load(*path=None*)

参数
path (*str*) –从中加载配置的路径。

从文件中加载配置，如果未指定配置文件，，则从 XDG 路径 (/etc/xdg/wlc) 中的”wlc”配置文件 (~/.config/wlc) 加载配置。

1.14.4 wlc.main

wlc.main.**main**(*settings=None*, *stdout=None*, *args=None*)

参数

- **settings** (*list*) –设置为元组列表覆盖
- **stdout** (*object*) –用于打印输出的 stdout 文件对象，使用 sys.stdout 作为默认值
- **args** (*list*) –要处理的命令行参数，使用 sys.args 作为默认值

命令行界面的主要入口点。

@wlc.main.**register_command**(*command*)

在*main()* 所用的主解析器中注册*Command* 类的装饰器。

Command

class wlc.main.**Command**(*args*, *config*, *stdout=None*)

调用命令的主类。

CHAPTER 2

管理员文档

2.1 配置说明

2.1.1 安装 Weblate

使用 Docker 安装

通过 dockerized Weblate 部署，您可以在几秒钟内启动并运行您的个人 Weblate 实例。Weblate 的所有依赖项已包含在内。PostgreSQL 被新建为默认数据库。

硬件要求

Weblate 应该可以在任何现代硬件上正常运行，以下是在单个主机（Weblate、数据库和 Web 服务器）上运行 Weblate 所需的最低配置：

- 3 GB 的内存
- 2 个 CPU 核心
- 1 GB 的存储空间

内存越多越好——用于所有级别的缓存（文件系统，数据库和 Weblate ）。

许多并发用户会增加所需的 CPU 内核数量。对于数百个翻译部件，推荐至少有 4 GB 的内存。

典型的数据库存储用量大约为每 1 百万单词 300 MB。克隆仓库所需的存储空间会变化，但 Weblate 试图通过浅克隆将其大小最小化。

备注： 安装 Weblate 的实际要求会因其中管理的翻译规模而大不相同。

安装

以下示例假设您拥有一个工作正常的 Docker 环境，并安装了 `docker-compose`。请查看 Docker 文档以获取说明。

1. 克隆 weblate-docker 仓库：

```
git clone https://github.com/WeblateOrg/docker-compose.git weblate-docker
cd weblate-docker
```

2. 使用您的设置创建一个 `docker-compose.override.yml` 文件。请参阅*Docker 环境变量* 以获取环境变量的完整列表。

```
version: '3'
services:
  weblate:
    ports:
      - 80:8080
    environment:
      WEBLATE_EMAIL_HOST: smtp.example.com
      WEBLATE_EMAIL_HOST_USER: user
      WEBLATE_EMAIL_HOST_PASSWORD: pass
      WEBLATE_SERVER_EMAIL: weblate@example.com
      WEBLATE_DEFAULT_FROM_EMAIL: weblate@example.com
      WEBLATE_SITE_DOMAIN: weblate.example.com
      WEBLATE_ADMIN_PASSWORD: password for the admin user
      WEBLATE_ADMIN_EMAIL: weblate.admin@example.com
```

备注： 如果未设置 *WEBLATE_ADMIN_PASSWORD*，则使用首次启动时显示的随机密码创建管理员用户。

提供的示例使 Weblate 侦听端口 80，在 `docker-compose.override.yml` 文件中编辑端口映射来更改。

---

3. 启动 Weblate 容器：

```
docker-compose up
```

享受您的 Weblate 部署，可以在 `weblate` 容器的端口 80 上进行访问。

在 2.15-2 版本发生变更: 最近更改了设置，以前有单独的 web 服务器容器，因为 2.15-2 开始，web 服务器已嵌入 Weblate 容器中。

在 3.7.1-6 版本发生变更: 在 2019 年 7 月（从 3.7.1-6 tag 开始）中，容器未以 root 用户身份运行。这已将裸露端口从 80 更改为 8080。

参见：

调用管理命令

选择 Docker hub 标签

你可以在 Docker hub 上使用以下标签，请参阅 https://hub.docker.com/r/weblate/weblate/tags/，以获得完整的可用标签列表。

标签名称	说明	用例
`latest`	与最新的标签版本相匹配的 Weblate 稳定发行版	在生产环境中滚动更新
`<VERSION>-<PATC`	Weblate 稳定发行版	生产环境下定义良好的部署
`edge`	Weblate 稳定发行版在 Docker 容器中的开发变化（例如依赖的更新）	在准生产环境中进行滚动更新
`edge-<DATE>-<SH`	Weblate 稳定发行版在 Docker 容器中的开发变化（例如依赖的更新）	在准生产环境中定义完善部署
`bleeding`	来自 Git 的 Weblate 开发版	滚动更新以测试即将推出的 Weblate 功能
`bleeding-<DATE>`	来自 Git 的 Weblate 开发版	定义明确的部署以测试即将推出的 Weblate 功能

每张图片在发布之前都经过我们的 CI 测试，所以即使是 *bleeding* 版本也应该可以安全使用。

具有 HTTPS 支持的 Docker 容器

请参阅安装 以获取常规部署说明，本节仅提及与之相比的差异。

使用自己的 SSL 证书

在 3.8-3 版本加入.

如果您要使用自己的 SSL 证书，只需将文件放入 Weblate 数据卷中（请参阅*Docker 容器卷*）：

- `ssl/fullchain.pem` 包含证书，包括任何需要的 CA 证书
- `ssl/privkey.pem` 包含有私钥

拥有这两个文件的用户必须与启动 docker 容器并将文件掩码设置为 `600`（仅拥有用户可读可写）的用户为同一用户。

此外，Weblate 容器现在将在端口 4443 上接受 SSL 连接，您将要在 docker compose override 中包括 HTTPS 的端口转发：

```
version: '3'
services:
  weblate:
    ports:
      - 80:8080
      - 443:4443
```

如果您已经在同一服务器上托管其他站点，则反向代理（例如 NGINX ）可能会使用端口 `80` 和 `443`。要将 HTTPS 连接从 NGINX 传递到 docker 容器，可以使用以下配置：

```
server {
    listen 443;
    listen [::]:443;

    server_name <SITE_URL>;
    ssl_certificate /etc/letsencrypt/live/<SITE>/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/<SITE>/privkey.pem;
```

(续下页)

(接上页)

```
    location / {
            proxy_set_header HOST $host;
            proxy_set_header X-Forwarded-Proto https;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Host $server_name;
            proxy_pass https://127.0.0.1:<EXPOSED_DOCKER_PORT>;
    }
}
```

将 <SITE_URL>，<SITE> 和 <EXPOSED_DOCKER_PORT> 替换为您环境中的实际值。

使用 Let's Encrypt 自动生成 SSL 证书

如果要在公共安装中使用 Let's Encrypt 自动生成的 SSL 证书，则需要在其他 Docker 容器中添加反向 HTTPS 代理，这将使用 https-portal。这是在 docker-compose-https.yml 文件中使用的。然后使用您的设置创建一个 docker-compose-https.override.yml 文件：

```
version: '3'
services:
  weblate:
    environment:
      WEBLATE_EMAIL_HOST: smtp.example.com
      WEBLATE_EMAIL_HOST_USER: user
      WEBLATE_EMAIL_HOST_PASSWORD: pass
      WEBLATE_SITE_DOMAIN: weblate.example.com
      WEBLATE_ADMIN_PASSWORD: password for admin user
  https-portal:
    environment:
      DOMAINS: 'weblate.example.com -> http://weblate:8080'
```

每当调用 **docker-compose** 时，您都需要将两个文件都传递给它，然后执行以下操作：

```
docker-compose -f docker-compose-https.yml -f docker-compose-https.override.yml␣
↪build
docker-compose -f docker-compose-https.yml -f docker-compose-https.override.yml up
```

升级 Docker 容器

通常，只更新 Weblate 容器并保持 PostgreSQL 容器为您的版本是一个好主意，因为升级 PostgreSQL 会很痛苦，并且在大多数情况下不会带来很多好处。

在 4.10-1 版本发生变更: 从 Weblate 4.10-1 开始，Docker 容器使用 Django 4.0 需要 PostgreSQL 10 或更高版本，请在升级 Weblate 之前对其进行升级。参见从 *4.9 升级到 4.10* 和升级 *PostgreSQL* 容器。

您可以通过坚持使用现有的 docker-compose，并且只是拉取最新镜像，然后重新启动，来执行此操作：

```
# Fetch latest versions of the images
docker-compose pull
# Stop and destroy the containers
docker-compose down
# Spawn new containers in the background
docker-compose up -d
# Follow the logs during upgrade
docker-compose logs -f
```

Weblate 数据库应在首次启动时自动迁移，并且不需要其他手动操作。

备注： Weblate 不支持跨大版本升级。比如，如果您使用的是 3.x 系列，要升级到 4.x，请首先升级到最新的 4.0.x-y 镜像（在撰写本文时为 `4.0.4-5`），它将进行迁移和然后继续升级到较新版本。

您可能还想更新 `docker-compose` 仓库，尽管在大多数情况下并不需要。在这种情况下，升级 PostgreSQL 服务器的信息，见*升级 PostgreSQL 容器*。

升级 PostgreSQL 容器

PostgreSQL 容器不支持版本间自动升级，需要手动升级。以下步骤显示了升级选项之一。

参见：

https://github.com/docker-library/postgres/issues/37

1. 停止 Weblate 容器：

```
docker-compose stop weblate cache
```

2. 备份数据库：

```
docker-compose exec database pg_dumpall --clean --username weblate > backup.sql
```

3. 停止数据库容器：

```
docker-compose stop database
```

4. 删除 PostgreSQL 卷：

```
docker-compose rm -v database
docker volume remove weblate-docker_postgres-data
```

5. 调整:file:‘docker-compose.yml‘以使用新的 PostgreSQL 版本。
6. 启动数据库容器：

```
docker-compose up -d database
```

7. 从备份恢复数据库：

```
cat backup.sql | docker-compose exec -T database psql --username weblate --
↪dbname postgres
```

8. （可选）更新 Weblate 用户的密码。迁移到 PostgreSQL 14 或 15 时可能需要这么做，因为密码存储方式发生了改变：

```
docker-compose exec -T database psql --username weblate --dbname postgres -c
↪"ALTER USER weblate WITH PASSWORD 'weblate'"
```

9. 启动所有剩余的容器：

```
docker-compose up -d
```

管理员登录

设置容器之后，您可以使用 *WEBLATE_ADMIN_PASSWORD* 中提供的密码以 管理员 用户身份登录，或者如果未设置该密码，则在首次启动时生成随机密码。

要重置 管理员 密码，请在 *WEBLATE_ADMIN_PASSWORD* 设置为新密码的情况下重启容器。

参见:

WEBLATE_ADMIN_PASSWORD, *WEBLATE_ADMIN_NAME*, *WEBLATE_ADMIN_EMAIL*

进程数量和内存消耗

uWSGI 和 Celery 的工作进程数是根据 CPU 的数量自动确定的。这适用于大多数云虚拟机，因为这些虚拟机通常具有很少的 CPU 和大量的内存。

在你有很多 CPU 核心并且碰到内存用尽问题情况下，尝试减少 worker 的数量：

```
environment:
  WEBLATE_WORKERS: 2
```

你还可以微调单个 worker 类别：

```
environment:
  WEB_WORKERS: 4
  CELERY_MAIN_OPTIONS: --concurrency 2
  CELERY_NOTIFY_OPTIONS: --concurrency 1
  CELERY_TRANSLATE_OPTIONS: --concurrency 1
```

参见:

WEBLATE_WORKERS, *CELERY_MAIN_OPTIONS*, *CELERY_NOTIFY_OPTIONS*, *CELERY_MEMORY_OPTIONS*, *CELERY_TRANSLATE_OPTIONS*, *CELERY_BACKUP_OPTIONS*, *CELERY_BEAT_OPTIONS*, *WEB_WORKERS*

横向扩展

在 4.6 版本加入.

您可以运行多个 Weblate 容器来水平扩展服务。`/app/data` 卷必须由所有容器共享，建议使用集群文件系统，如 GlusterFS。对于每个容器，:file:‘/app/cache‘卷应该是分开的。

每个 Weblate 容器都使用 *WEBLATE_SERVICE* 环境变量定义了角色。请仔细阅读文档，因为某些服务应该在集群中只运行一次，并且服务的顺序也很重要。

您可以在 `docker-compose` 仓库中找到示例设置，如 docker-compose-split.yml。

Docker 环境变量

可以使用下述环境变量在 Docker 容器中设置 Weblate 的许多配置。

如果您需要定义不通过 Docker 环境变量公开的设置，请参考修改未暴露为 *Docker* 环境变量的设置项。

通用设置

WEBLATE_DEBUG

使用*DEBUG*配置 Django 调试模式。

示例：

```
environment:
  WEBLATE_DEBUG: 1
```

参见：

禁止调试模式

WEBLATE_LOGLEVEL

配置日志记录的详细程度。

WEBLATE_LOGLEVEL_DATABASE

配置数据库查询详细程度的日志记录。

WEBLATE_SITE_TITLE

更改所有页面页眉上显示的站点标题。

WEBLATE_SITE_DOMAIN

配置站点域名。此参数为必填项。

参见：

设置正确的网站域名, *SITE_DOMAIN*

WEBLATE_ADMIN_NAME

WEBLATE_ADMIN_EMAIL

配置站点管理员的姓名和电子邮件。它用于*ADMINS*设置和创建 管理员用户（有关此信息，请参阅*WEBLATE_ADMIN_PASSWORD*）。

示例：

```
environment:
  WEBLATE_ADMIN_NAME: Weblate admin
  WEBLATE_ADMIN_EMAIL: noreply@example.com
```

参见：

管理员登录, *是当地配置管理设置*, *ADMINS*

WEBLATE_ADMIN_PASSWORD

设置 管理员用户的密码。

- 如果未设置并且 管理员用户不存在，则会使用首次启动容器时显示的随机密码来创建它。
- 如果未设置并且 管理员用户存在，则不执行任何操作。
- 如果设置，则在每次容器启动时都会对 管理员用户进行调整，以匹配*WEBLATE_ADMIN_PASSWORD*，*WEBLATE_ADMIN_NAME*和*WEBLATE_ADMIN_EMAIL*。

警告： 将密码存储在配置文件中可能会带来安全风险。考虑仅将此变量用于初始设置（或让 Weblate 在初始启动时生成随机密码）或用于密码恢复。

参见：

管理员登录, *WEBLATE_ADMIN_PASSWORD*, *WEBLATE_ADMIN_PASSWORD_FILE*, *WEBLATE_ADMIN_NAME*, *WEBLATE_ADMIN_EMAIL*

WEBLATE_ADMIN_PASSWORD_FILE

设置指向包含 管理员用户密码的一个文件。

参见:

WEBLATE_ADMIN_PASSWORD

WEBLATE_SERVER_EMAIL

发送错误消息的电子邮箱地址。

参见:

SERVER_EMAIL, 配置电子邮件发送的设置

WEBLATE_DEFAULT_FROM_EMAIL

配置外发电子邮件的地址。

参见:

DEFAULT_FROM_EMAIL, 配置电子邮件发送的设置

WEBLATE_CONTACT_FORM

配置联系表单行为，请参阅*CONTACT_FORM*。

WEBLATE_ALLOWED_HOSTS

使用*ALLOWED_HOSTS* 配置允许的 HTTP 主机名。

默认为 * 来允许所有的主机名称。

示例:

```
environment:
  WEBLATE_ALLOWED_HOSTS: weblate.example.com,example.com
```

参见:

ALLOWED_HOSTS, 允许主机设置, 设置正确的网站域名

WEBLATE_REGISTRATION_OPEN

通过切换*REGISTRATION_OPEN* 配置是否打开注册。

示例:

```
environment:
  WEBLATE_REGISTRATION_OPEN: 0
```

WEBLATE_REGISTRATION_ALLOW_BACKENDS

配置可用于通过*REGISTRATION_ALLOW_BACKENDS* 创建新账户的身份验证方法。

示例:

```
environment:
  WEBLATE_REGISTRATION_OPEN: 0
  WEBLATE_REGISTRATION_ALLOW_BACKENDS: azuread-oauth2,azuread-tenant-
↪oauth2
```

WEBLATE_REGISTRATION_REBIND

在 4.16 版本加入.

配置*REGISTRATION_REBIND*.

WEBLATE_TIME_ZONE

在 Weblate 中配置使用的时区，请参阅 TIME_ZONE。

备注: 为了更改 Docker 自己的时区，使用 TZ 环境变量。

示例：

```
environment:
  WEBLATE_TIME_ZONE: Europe/Prague
```

WEBLATE_ENABLE_HTTPS

让 Weblate 假定在反向 HTTPS 代理后面操作，这使 Weblate 在电子邮件和 API 链接中使用 HTTPS，或者在 cookies 上设置安全标记。

提示： 可能的警告请参见*ENABLE_HTTPS* 文档。

备注： 这不会使 Weblate 容器接受 HTTPS 连接，您同样需要配置它，示例请参见*具有 HTTPS 支持的 Docker 容器*。

示例：

```
environment:
  WEBLATE_ENABLE_HTTPS: 1
```

参见：

*ENABLE_HTTPS*设置正确的网站域名，*WEBLATE_SECURE_PROXY_SSL_HEADER*

WEBLATE_INTERLEDGER_PAYMENT_POINTERS

在 4.12.1 版本加入.

让 Weblate 在文档的头部设置 *meta[name=monetization]* 字段。如果指定了多个，则随机选择一个。

参见：

INTERLEDGER_PAYMENT_POINTERS

WEBLATE_IP_PROXY_HEADER

让 Weblate 从任何给定的 HTTP 标头中取回 IP 地址。在使用 Weblate 容器之前的反向代理时使用它。

允许*IP_BEHIND_REVERSE_PROXY* 并设置*IP_PROXY_HEADER*。

备注： 格式必须符合 Django 的要求。Django transforms 原始 HTTP 标头如下命名：

- 将所有字符装换为大写
- 用下划线替换任何连字符
- 预置 `HTTP_` 前缀

所以 `X-Forwarded-For` 将被映射到 `HTTP_X_FORWARDED_FOR`。

示例：

```
environment:
  WEBLATE_IP_PROXY_HEADER: HTTP_X_FORWARDED_FOR
```

WEBLATE_SECURE_PROXY_SSL_HEADER

代表 HTTP 标头/值的组合的元组，用于表达请求，这样的元组是安全的。当 Weblate 在进行终止 SSL 的反向代理之后运行时，这是需要的，终止 SSL 不通过标准 HTTPS 标头。

示例：

```
environment:
  WEBLATE_SECURE_PROXY_SSL_HEADER: HTTP_X_FORWARDED_PROTO,https
```

参见：

SECURE_PROXY_SSL_HEADER

WEBLATE_REQUIRE_LOGIN

启用*REQUIRE_LOGIN* 而在整个 Weblate 上强制认证。

示例：

```
environment:
  WEBLATE_REQUIRE_LOGIN: 1
```

WEBLATE_LOGIN_REQUIRED_URLS_EXCEPTIONS

WEBLATE_ADD_LOGIN_REQUIRED_URLS_EXCEPTIONS

WEBLATE_REMOVE_LOGIN_REQUIRED_URLS_EXCEPTIONS

使用*LOGIN_REQUIRED_URLS_EXCEPTIONS* 来为整个 Weblate 安装所需的身份验证添加 URL 例外。

可以替换整个设置，或者使用 ADD 和 REMOVE 变量修改默认值。

WEBLATE_GOOGLE_ANALYTICS_ID

通过更改*GOOGLE_ANALYTICS_ID* 来配置 Google Analytics（分析）的 ID。

WEBLATE_GITHUB_USERNAME

WEBLATE_GITHUB_TOKEN

WEBLATE_GITHUB_HOST

配置 GitHub 拉取请求集成，方法是更改*GITHUB_CREDENTIALS*。

参见：

GitHub 拉取请求

WEBLATE_GITLAB_USERNAME

WEBLATE_GITLAB_TOKEN

WEBLATE_GITLAB_HOST

配置 GitLab merge-requests 集成，方法是更改*GITLAB_CREDENTIALS*。

参见：

GitLab 合并请求

WEBLATE_GITEA_USERNAME

WEBLATE_GITEA_TOKEN

WEBLATE_GITEA_HOST

配置 Gitea 拉取请求集成，方法是更改*GITEA_CREDENTIALS*。

参见：

Gitea 拉取请求

WEBLATE_PAGURE_USERNAME

WEBLATE_PAGURE_TOKEN

WEBLATE_PAGURE_HOST

配置 Pagure merge-requests 集成，方法是更改*PAGURE_CREDENTIALS*。

参见：

Pagure 合并请求

WEBLATE_BITBUCKETSERVER_USERNAME

WEBLATE_BITBUCKETSERVER_TOKEN

WEBLATE_BITBUCKETSERVER_HOST

配置 Bitbucket 服务器拉取请求集成，方法是更改*BITBUCKETSERVER_CREDENTIALS*。

参见:

Bitbucket 服务器拉取请求

WEBLATE_DEFAULT_PULL_MESSAGE

通过更改*DEFAULT_PULL_MESSAGE* 来配置使用 API 发起的拉取请求的默认标题和说明

参见:

DEFAULT_PULL_MESSAGE

WEBLATE_SIMPLIFY_LANGUAGES

配置语言简化策略，请参见*SIMPLIFY_LANGUAGES*。

WEBLATE_DEFAULT_ACCESS_CONTROL

为新项目配置默认的访问控制，请参见*DEFAULT_ACCESS_CONTROL*。

WEBLATE_DEFAULT_RESTRICTED_COMPONENT

为新部件的受限制的访问 配置默认值，请参见*DEFAULT_RESTRICTED_COMPONENT*。

WEBLATE_DEFAULT_TRANSLATION_PROPAGATION

为新部件的允许同步翻译 配置默认值，请参见*DEFAULT_TRANSLATION_PROPAGATION*。

WEBLATE_DEFAULT_COMMITER_EMAIL

配置*DEFAULT_COMMITER_EMAIL*。

WEBLATE_DEFAULT_COMMITER_NAME

配置*DEFAULT_COMMITER_NAME*。

WEBLATE_DEFAULT_SHARED_TM

配置*DEFAULT_SHARED_TM*.

WEBLATE_AKISMET_API_KEY

配置 Akismet API 密钥，请参见*AKISMET_API_KEY*。

WEBLATE_GPG_IDENTITY

配置提交的 GPG 签名，请参见*WEBLATE_GPG_IDENTITY*。

参见:

使用 GnuPG 为 Git 提交签名

WEBLATE_URL_PREFIX

配置 Weblate 运行的 URL 前缀，请参见*URL_PREFIX*。

WEBLATE_SILENCED_SYSTEM_CHECKS

配置您不想要显示的检查，请参见 SILENCED_SYSTEM_CHECKS。

WEBLATE_CSP_SCRIPT_SRC

WEBLATE_CSP_IMG_SRC

WEBLATE_CSP_CONNECT_SRC

WEBLATE_CSP_STYLE_SRC

WEBLATE_CSP_FONT_SRC

允许定制 `Content-Security-Policy` HTTP 标头。

参见:

内容安全政策, *CSP_SCRIPT_SRC*, *CSP_IMG_SRC*, *CSP_CONNECT_SRC*, *CSP_STYLE_SRC*, *CSP_FONT_SRC*

WEBLATE_LICENSE_FILTER

配置*LICENSE_FILTER*.

WEBLATE_LICENSE_REQUIRED

配置*LICENSE_REQUIRED*

WEBLATE_WEBSITE_REQUIRED

配置*WEBSITE_REQUIRED*

WEBLATE_HIDE_VERSION

配置*HIDE_VERSION*。

WEBLATE_BASIC_LANGUAGES

配置*BASIC_LANGUAGES*.

WEBLATE_DEFAULT_AUTO_WATCH

配置*DEFAULT_AUTO_WATCH*.

WEBLATE_RATELIMIT_ATTEMPTS

WEBLATE_RATELIMIT_LOCKOUT

WEBLATE_RATELIMIT_WINDOW

在 4.6 版本加入.

配置速率限制器。

提示: 你可以为任何速率限制器的范围设置配置。要做到这一点，请在:ref:‘rate-limit’ 中描述的任何设置中添加 ‘WEBLATE_‘前缀。

参见:

频次限制, *RATELIMIT_ATTEMPTS*, *RATELIMIT_WINDOW*, *RATELIMIT_LOCKOUT*

WEBLATE_API_RATELIMIT_ANON

WEBLATE_API_RATELIMIT_USER

在 4.11 版本加入.

配置 API 速率限制。默认值如下：匿名用户 `100/day`，已验证身份用户‘‘5000/hour‘。

参见:

API 频次限制

WEBLATE_ENABLE_HOOKS

在 4.13 版本加入.

配置*ENABLE_HOOKS*。

WEBLATE_ENABLE_AVATARS

在 4.6.1 版本加入.

配置*ENABLE_AVATARS*.

WEBLATE_AVATAR_URL_PREFIX

在 4.15 版本加入.

配置*AVATAR_URL_PREFIX*。

WEBLATE_LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH

在 4.9 版本加入.

配置*LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH*.

WEBLATE_SSH_EXTRA_ARGS

在 4.9 版本加入.

配置*SSH_EXTRA_ARGS*。

WEBLATE_BORG_EXTRA_ARGS

在 4.9 版本加入.

配置*BORG_EXTRA_ARGS*。

WEBLATE_ENABLE_SHARING

在 4.14.1 版本加入.

配置*ENABLE_SHARING*.

WEBLATE_EXTRA_HTML_HEAD

在 4.15 版本加入.

配置*EXTRA_HTML_HEAD*.

WEBLATE_PRIVATE_COMMIT_EMAIL_TEMPLATE

在 4.15 版本加入.

配置*PRIVATE_COMMIT_EMAIL_TEMPLATE*.

WEBLATE_PRIVATE_COMMIT_EMAIL_OPT_IN

在 4.15 版本加入.

配置*PRIVATE_COMMIT_EMAIL_OPT_IN*.

WEBLATE_CORS_ALLOWED_ORIGINS

在 4.16 版本加入.

允许来自给定源的 CORS 请求。

示例:

```
environment:
  WEBLATE_CORS_ALLOWED_ORIGINS: https://example.com,https://weblate.org
```

自动建议设置

在 4.13 版本发生变更: 自动建议服务现在在用户界面进行配置，见配置自动建议。

现有的环境变量在迁移到 Weblate 4.13 时被导入，但改变它们不会有任何进一步的影响。

身份验证设置

LDAP

WEBLATE_AUTH_LDAP_SERVER_URI

WEBLATE_AUTH_LDAP_USER_DN_TEMPLATE

WEBLATE_AUTH_LDAP_USER_ATTR_MAP

WEBLATE_AUTH_LDAP_BIND_DN

WEBLATE_AUTH_LDAP_BIND_PASSWORD

WEBLATE_AUTH_LDAP_BIND_PASSWORD_FILE

包含 LDAP 服务器绑定密码的文件的路径。

参见:

WEBLATE_AUTH_LDAP_BIND_PASSWORD

WEBLATE_AUTH_LDAP_CONNECTION_OPTION_REFERRALS

WEBLATE_AUTH_LDAP_USER_SEARCH

WEBLATE_AUTH_LDAP_USER_SEARCH_FILTER

WEBLATE_AUTH_LDAP_USER_SEARCH_UNION

WEBLATE_AUTH_LDAP_USER_SEARCH_UNION_DELIMITER

LDAP 身份验证配置。

直接绑定的示例:

```
environment:
  WEBLATE_AUTH_LDAP_SERVER_URI: ldap://ldap.example.org
  WEBLATE_AUTH_LDAP_USER_DN_TEMPLATE: uid=%(user)s,ou=People,dc=example,dc=net
  # map weblate 'full_name' to ldap 'name' and weblate 'email' attribute to
↪'mail' ldap attribute.
  # another example that can be used with OpenLDAP: 'full_name:cn,email:mail'
  WEBLATE_AUTH_LDAP_USER_ATTR_MAP: full_name:name,email:mail
```

搜索与绑定的示例:

```
environment:
  WEBLATE_AUTH_LDAP_SERVER_URI: ldap://ldap.example.org
  WEBLATE_AUTH_LDAP_BIND_DN: CN=ldap,CN=Users,DC=example,DC=com
  WEBLATE_AUTH_LDAP_BIND_PASSWORD: password
  WEBLATE_AUTH_LDAP_USER_ATTR_MAP: full_name:name,email:mail
  WEBLATE_AUTH_LDAP_USER_SEARCH: CN=Users,DC=example,DC=com
```

联合搜索与绑定的示例:

```
environment:
  WEBLATE_AUTH_LDAP_SERVER_URI: ldap://ldap.example.org
  WEBLATE_AUTH_LDAP_BIND_DN: CN=ldap,CN=Users,DC=example,DC=com
  WEBLATE_AUTH_LDAP_BIND_PASSWORD: password
  WEBLATE_AUTH_LDAP_USER_ATTR_MAP: full_name:name,email:mail
  WEBLATE_AUTH_LDAP_USER_SEARCH_UNION: ou=users,dc=example,
↪dc=com|ou=otherusers,dc=example,dc=com
```

针对 Active Directory 的搜索与绑定的示例:

```
environment:
  WEBLATE_AUTH_LDAP_BIND_DN: CN=ldap,CN=Users,DC=example,DC=com
  WEBLATE_AUTH_LDAP_BIND_PASSWORD: password
  WEBLATE_AUTH_LDAP_SERVER_URI: ldap://ldap.example.org
  WEBLATE_AUTH_LDAP_CONNECTION_OPTION_REFERRALS: 0
  WEBLATE_AUTH_LDAP_USER_ATTR_MAP: full_name:name,email:mail
  WEBLATE_AUTH_LDAP_USER_SEARCH: CN=Users,DC=example,DC=com
  WEBLATE_AUTH_LDAP_USER_SEARCH_FILTER: (sAMAccountName=%(user)s)
```

参见:

LDAP 身份验证

GitHub

WEBLATE_SOCIAL_AUTH_GITHUB_KEY

WEBLATE_SOCIAL_AUTH_GITHUB_SECRET

WEBLATE_SOCIAL_AUTH_GITHUB_ORG_KEY

WEBLATE_SOCIAL_AUTH_GITHUB_ORG_SECRET

WEBLATE_SOCIAL_AUTH_GITHUB_ORG_NAME

WEBLATE_SOCIAL_AUTH_GITHUB_TEAM_KEY

WEBLATE_SOCIAL_AUTH_GITHUB_TEAM_SECRET

WEBLATE_SOCIAL_AUTH_GITHUB_TEAM_ID

允许*GitHub 身份验证*。

Bitbucket

WEBLATE_SOCIAL_AUTH_BITBUCKET_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_BITBUCKET_OAUTH2_SECRET

WEBLATE_SOCIAL_AUTH_BITBUCKET_KEY

WEBLATE_SOCIAL_AUTH_BITBUCKET_SECRET

允许*Bitbucket 身份验证*。

Facebook

WEBLATE_SOCIAL_AUTH_FACEBOOK_KEY

WEBLATE_SOCIAL_AUTH_FACEBOOK_SECRET

允许*Facebook OAuth 2*.

Google

WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET

WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_WHITELISTED_DOMAINS

WEBLATE_SOCIAL_AUTH_GOOGLE_OAUTH2_WHITELISTED_EMAILS
允许*Google OAuth 2*。

GitLab

WEBLATE_SOCIAL_AUTH_GITLAB_KEY

WEBLATE_SOCIAL_AUTH_GITLAB_SECRET

WEBLATE_SOCIAL_AUTH_GITLAB_API_URL
允许*GitLab OAuth 2*。

Gitea

WEBLATE_SOCIAL_AUTH_GITEA_API_URL

WEBLATE_SOCIAL_AUTH_GITEA_KEY

WEBLATE_SOCIAL_AUTH_GITEA_SECRET
启用 Gitea 身份验证。

Azure Active Directory

WEBLATE_SOCIAL_AUTH_AZUREAD_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_AZUREAD_OAUTH2_SECRET
启用 Azure Active Directory 身份验证，请参见*Microsoft Azure Active Directory*。

支持 Azure Active Directory 租户

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_SECRET

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_TENANT_ID
启用支持 Azure Active Directory 租户的身份验证，请参见*Microsoft Azure Active Directory*。

Keycloak

WEBLATE_SOCIAL_AUTH_KEYCLOAK_KEY

WEBLATE_SOCIAL_AUTH_KEYCLOAK_SECRET

WEBLATE_SOCIAL_AUTH_KEYCLOAK_PUBLIC_KEY

WEBLATE_SOCIAL_AUTH_KEYCLOAK_ALGORITHM

WEBLATE_SOCIAL_AUTH_KEYCLOAK_AUTHORIZATION_URL

WEBLATE_SOCIAL_AUTH_KEYCLOAK_ACCESS_TOKEN_URL

WEBLATE_SOCIAL_AUTH_KEYCLOAK_TITLE

WEBLATE_SOCIAL_AUTH_KEYCLOAK_IMAGE

允许 Keycloak 身份验证，请参见 documentation。

Linux 销售商

您可以通过将后面的变量设置为任何值，使用 Linux 销售商身份验证服务来允许身份验证。

WEBLATE_SOCIAL_AUTH_FEDORA

WEBLATE_SOCIAL_AUTH_OPENSUSE

WEBLATE_SOCIAL_AUTH_UBUNTU

Slack

WEBLATE_SOCIAL_AUTH_SLACK_KEY

SOCIAL_AUTH_SLACK_SECRET

允许 Slack 身份验证，请参见*Slack*。

OpenID 连接

在 4.13-1 版本加入.

WEBLATE_SOCIAL_AUTH_OIDC_OIDC_ENDPOINT

WEBLATE_SOCIAL_AUTH_OIDC_KEY

WEBLATE_SOCIAL_AUTH_OIDC_SECRET

WEBLATE_SOCIAL_AUTH_OIDC_USERNAME_KEY

配置通用 OpenID Connect 集成。

参见:

OIDC (OpenID Connect)

SAML

在第一次启动容器时会自动生成自签名的 SAML 密钥。如果您想要使用自己的密钥，请将证书和私钥分别放在 `/app/data/ssl/saml.crt` 和 `/app/data/ssl/saml.key` 中。

WEBLATE_SAML_IDP_ENTITY_ID

WEBLATE_SAML_IDP_URL

WEBLATE_SAML_IDP_X509CERT

WEBLATE_SAML_IDP_IMAGE

WEBLATE_SAML_IDP_TITLE

SAML 身份提供者设置，请参见*SAML 身份验证*。

其他身份认证设置

WEBLATE_NO_EMAIL_AUTH

当设置为任何值时禁止电子邮箱身份认证。请参见*关闭密码身份验证*。

PostgreSQL 数据库设置

数据库由 `docker-compose.yml` 建立，所以这些设置影响 Weblate 和 PostgreSQL 容器。

参见:

Weblate 的数据库设置

POSTGRES_PASSWORD

PostgreSQL 密码。

POSTGRES_PASSWORD_FILE

包含 PostgreSQL 密码的文件的路径。用作 POSTGRES_PASSWORD 的替代。

POSTGRES_USER

PostgreSQL 用户名。

POSTGRES_DATABASE

PostgreSQL 数据库名。

POSTGRES_HOST

PostgreSQL 服务器主机名或 IP 地址。默认为 `database`。

POSTGRES_PORT

PostgreSQL 服务器端口。默认为无（使用默认值）。

POSTGRES_SSL_MODE

配置 PostgreSQL 如何处理 SSL 连接到服务器，可能的选项请参见 SSL Mode Descriptions

POSTGRES_ALTER_ROLE

在迁移过程中配置要改变的角色名称，请参见*配置 Weblate 来使用 PostgreSQL*。

POSTGRES_CONN_MAX_AGE

在 4.8.1 版本加入.

数据库连接的寿命，以秒为单位的整数。使用 0 可以在每次请求结束时关闭数据库连接（这是默认行为）。

启用连接持久性通常会导致更多的数据库开放连接。请在启用前调整你的数据库配置。

配置的示例：

```
environment:
    POSTGRES_CONN_MAX_AGE: 3600
```

参见:

CONN_MAX_AGE, 持久连接

POSTGRES_DISABLE_SERVER_SIDE_CURSORS

在 4.9.1 版本加入.

禁用数据库中的服务器端游标。这在一些:command:‘pgbouncer‘设置中是必要的。

配置的示例:

```
environment:
    POSTGRES_DISABLE_SERVER_SIDE_CURSORS: 1
```

参见:

DISABLE_SERVER_SIDE_CURSORS, 事务池和服务器端游标

数据库备份设置

参见:

下载的数据用于备份

WEBLATE_DATABASE_BACKUP

使用*DATABASE_BACKUP* 配置每日数据库转储。默认为 plain。

缓存服务器设置

Weblate 强烈推荐使用 Redis，在 Docker 中运行 Weblate 时您必须提供 Redis 实例。

参见:

允许缓存

REDIS_HOST

Redis 服务器主机名称或 IP 地址。默认为 cache。

REDIS_PORT

Redis 服务器端口。默认为 6379。

REDIS_DB

Redis 数据库编号，默认为 1。

REDIS_PASSWORD

Redis 服务器密码，默认不使用。

REDIS_PASSWORD_FILE

包含 Redis 服务器密码的文件的路径。

参见:

REDIS_PASSWORD

REDIS_TLS

允许使用 SSL 进行 Redis 连接。

REDIS_VERIFY_SSL

可以用于禁止 Redis 连接的 SSL 身份认证。

电子邮件服务器设置

要使外发电子邮件正常工作，您需要提供一个电子邮件服务器。

TLS 配置示例：

```
environment:
    WEBLATE_EMAIL_HOST: smtp.example.com
    WEBLATE_EMAIL_HOST_USER: user
    WEBLATE_EMAIL_HOST_PASSWORD: pass
```

SSL 配置的示例：

```
environment:
    WEBLATE_EMAIL_HOST: smtp.example.com
    WEBLATE_EMAIL_PORT: 465
    WEBLATE_EMAIL_HOST_USER: user
    WEBLATE_EMAIL_HOST_PASSWORD: pass
    WEBLATE_EMAIL_USE_TLS: 0
    WEBLATE_EMAIL_USE_SSL: 1
```

参见：

配置电子邮件发件箱

WEBLATE_EMAIL_HOST

电子邮件服务器主机名或 IP 地址。

参见：

WEBLATE_EMAIL_PORT, *WEBLATE_EMAIL_USE_SSL*, *WEBLATE_EMAIL_USE_TLS*, EMAIL_HOST

WEBLATE_EMAIL_PORT

电子邮件服务器端口，默认为 25。

参见：

EMAIL_PORT

WEBLATE_EMAIL_HOST_USER

电子邮件身份验证用户。

参见：

EMAIL_HOST_USER

WEBLATE_EMAIL_HOST_PASSWORD

电子邮件验证密码。

参见：

EMAIL_HOST_PASSWORD

WEBLATE_EMAIL_HOST_PASSWORD_FILE

到包含电子邮件验证密码的文件的路径。

参见：

WEBLATE_EMAIL_HOST_PASSWORD

WEBLATE_EMAIL_USE_SSL

与 SMTP 服务器通信时是否使用隐式 TLS（安全）连接。在大多数电子邮件文档中，这种 TLS 连接类型称为 SSL。通常在端口 465 上使用。如果遇到问题，请参阅显式 TLS 设置 *WEBLATE_EMAIL_USE_TLS*。

在 4.11 版本发生变更: SSL/TLS 支持是根据:envvar:‘WEBLATE_EMAIL_PORT‘自动启用的。

参见:

WEBLATE_EMAIL_PORT, *WEBLATE_EMAIL_USE_TLS*, EMAIL_USE_SSL

WEBLATE_EMAIL_USE_TLS

与 SMTP 服务器通讯时是否使用 TLS（安全）连接。这用于显式 TLS 连接，通常在端口 587 或 25 上。如果您遇到挂起的连接，请参见隐式 TLS 设置 *WEBLATE_EMAIL_USE_SSL*。

在 4.11 版本发生变更: SSL/TLS 支持是根据:envvar:‘WEBLATE_EMAIL_PORT‘自动启用的。

参见:

WEBLATE_EMAIL_PORT, *WEBLATE_EMAIL_USE_SSL*, EMAIL_USE_TLS

WEBLATE_EMAIL_BACKEND

将 Django 后端配置为用于发送电子邮件。

参见:

配置电子邮件发送的设置, EMAIL_BACKEND

WEBLATE_AUTO_UPDATE

配置 Weblate 是否更新仓库以及如何更新。

参见:

AUTO_UPDATE

备注: 这是一个布尔值设定（使用 `"true"` 或者 `"false"`）。

站点集成

WEBLATE_GET_HELP_URL

配置*GET_HELP_URL*.

WEBLATE_STATUS_URL

配置*STATUS_URL*.

WEBLATE_LEGAL_URL

配置: setting:*LEGAL_URL*.

WEBLATE_PRIVACY_URL

配置*PRIVACY_URL*.

错误报告

推荐从安装中系统地收集错误，请参见收集错误报告。

要启用对 Rollbar 的支持，请进行以下设置:

ROLLBAR_KEY

您的 Rollbar 发布服务器访问令牌。

ROLLBAR_ENVIRONMENT

您的 Rollbar 环境，默认为 `production`。

要启用对 Sentry 的支持，请进行以下设置:

SENTRY_DSN

您的 Sentry DSN。

SENTRY_ENVIRONMENT

您的 Sentry 环境（可选）。

语言本地化内容分发网络

WEBLATE_LOCALIZE_CDN_URL

WEBLATE_LOCALIZE_CDN_PATH

在 4.2.1 版本加入.

:ref:‘addon-weblate.cdn.cdnjs‘的配置。

WEBLATE_LOCALIZE_CDN_PATH 是容器内的路径。它应该存储在持久卷上，而不能存储在瞬态存储器中。

一种可能性是存储在 Weblate 数据目录中：

```
environment:
  WEBLATE_LOCALIZE_CDN_URL: https://cdn.example.com/
  WEBLATE_LOCALIZE_CDN_PATH: /app/data/l10n-cdn
```

备注： 您负责设置 Weblate 生成的文件的服务，它只在配置的位置存储文件。

参见：

weblate-cdn, *LOCALIZE_CDN_URL*, *LOCALIZE_CDN_PATH*

更改启用的 app、检查、附加组件或自动修复

在 3.8-5 版本加入.

可以通过后面的变量来调整允许的检查、附加组件或自动修复的内建配置：

WEBLATE_ADD_APPS

WEBLATE_REMOVE_APPS

WEBLATE_ADD_CHECK

WEBLATE_REMOVE_CHECK

WEBLATE_ADD_AUTOFIX

WEBLATE_REMOVE_AUTOFIX

WEBLATE_ADD_ADDONS

WEBLATE_REMOVE_ADDONS

示例：

```
environment:
  WEBLATE_REMOVE_AUTOFIX: weblate.trans.autofixes.whitespace.
↪SameBookendingWhitespace
  WEBLATE_ADD_ADDONS: customize.addons.MyAddon,customize.addons.OtherAddon
```

参见：

CHECK_LIST, *AUTOFIX_LIST*, *WEBLATE_ADDONS*, INSTALLED_APPS

容器设置

WEBLATE_WORKERS

在 4.6.1 版本加入.

容器中运行的工作者进程的基本数量。如果没有设置，它将在容器启动时根据可用的 CPU 核数自动确定。

它被用来确定 *CELERY_MAIN_OPTIONS*, *CELERY_NOTIFY_OPTIONS*, *CELERY_MEMORY_OPTIONS*, : envvar:*CELERY_TRANSLATE_OPTIONS*, *CELERY_BACKUP_OPTIONS*, *CELERY_BEAT_OPTIONS*, 和 *WEB_WORKERS*。你可以使用这些设置来微调。

CELERY_MAIN_OPTIONS

CELERY_NOTIFY_OPTIONS

CELERY_MEMORY_OPTIONS

CELERY_TRANSLATE_OPTIONS

CELERY_BACKUP_OPTIONS

CELERY_BEAT_OPTIONS

这些变量允许您调整 Celery worker 选项。它可以用于调整并发性（`--concurrency 16`），或使用不同的池实现（`--pool=gevent`）。

默认情况下，并发工作者的数量是基于 *WEBLATE_WORKERS*。

示例：

```
environment:
  CELERY_MAIN_OPTIONS: --concurrency 16
```

参见：

Celery worker 选项, 使用 *Celery* 的后台任务

WEB_WORKERS

配置应该执行多少个 uWSGI worker。

它默认为 *WEBLATE_WORKERS*。

示例：

```
environment:
  WEB_WORKERS: 32
```

WEBLATE_SERVICE

定义应在容器内执行哪些服务，使用对象横向扩展。

定义了以下服务:

celery-beat
: Celery 任务调度器，应该只有一个实例在运行。这个容器也负责数据库结构的迁移，它应该在其他容器之前启动。

celery-backup
: 用于备份的 Celery worker，应该只有一个实例在运行。

celery-celery
: 普通的 Celery worker.

celery-memory
: 翻译记忆库 Celery worker。

celery-notify
通知 Celery worker。

celery-translate
自动翻译 Celery worker。

web
Web 服务器。

Docker 容器卷

Weblate 容器导出了两个卷（数据和缓存）。其他服务容器（PostgreSQL 或 Redis）也具有其数据卷，但本文档未涵盖这些数据卷。

数据卷用于存储 Weblate 持久数据（例如克隆的仓库）或自定义 Weblate 安装。

Docker 卷在主机系统上的位置取决于您的 Docker 配置，但通常存储在 `/var/lib/docker/volumes/weblate-docker_weblate-data/_data/` 中。(该路径由你的 docker-compose 目录的名称、容器和卷的名称组成)。在容器中，它挂载为 `/app/data`。

缓存卷被挂载为 `/app/cache`，用于存储静态文件和*CACHE_DIR*。它的内容在容器启动时被重新创建，卷可以使用短暂的文件系统（如 *tmpfs*）挂载。

当手动创建卷时，目录应该由 UID 1000 拥有，因为那是容器内使用的用户。

参见:

Docker 卷文档

修改未暴露为 Docker 环境变量的设置项

Docker 环境变量。

如果您发现没有作为环境变量公开的设置，并且您认为它应该公开，请随时*要求在未来版本的 Weblate* 中公开它。

如果您需要修改未公开为 Docker 环境变量的设置，您仍然可以这样做*来自数据卷* 或者*扩展 Docker 镜像*。

参见:

定制 Weblate

覆盖数据卷的设置

您可以在 `/app/data/settings-override.py` 创建一个文件，即在*数据卷* 的根目录，以扩展或覆盖通过环境变量定义的设置。

通过扩展 Docker 镜像覆盖设置

要在 Docker 镜像级别而不是从数据卷覆盖设置:

1. *创建自定义 Python 包*。
2. 将一个模块添加到您的包中，该模块从 `weblate.settings_docker` 导入所有设置。

   例如，在*建立 Python 模块*，`weblate_customization/weblate_customization/settings.py` 定义的示例包结构中，您可以使用以下初始代码创建一个文件:

   ```
   from weblate.settings_docker import *
   ```

3. 创建一个继承自官方 Weblate Docker 镜像的自定义 `Dockerfile`，然后安装你的包并将 `DJANGO_SETTINGS_MODULE` 环境变量指向你的设置模块:

```
FROM weblate/weblate

USER root

COPY weblate_customization /usr/src/weblate_customization
RUN pip install --no-cache-dir /usr/src/weblate_customization
ENV DJANGO_SETTINGS_MODULE=weblate_customization.settings

USER 1000
```

4. 不要使用官方的 Weblate Docker 镜像，而是从这个 `Dockerfile` 文件构建一个自定义镜像。

   使用 `docker-compose.override.yml` 则 无法干净地 做到这一点。您 可以将 `build: .` 添加到该文件中的 `weblate` 节点，但随后您的自定义镜像将在系统中被打上 `weblate/weblate` 的标签，这可能会出问题。

   因此，比起原封不动直接套用来自 官方仓库 的 `docker-compose.yml`，并通过 `docker-compose.override.yml` 进行扩展，你或许想要复制一份 `docker-compose.yml` 文件对其进行编辑，然后借助 `build: .` 用编辑后的文件替换 `image: weblate/weblate`.

   有关使用 `docker-compose` 时从源代码构建镜像的详情请见 Compose file build reference.

5. 扩展你的自定义设置模块来定义或重新定义设置。

   你可以定义上方 import 语句之前或之后的设置，以确定哪些设置优先。import 语句之前定义的设置可以被环境变量和数据卷中定义的设置覆盖所覆盖。不能覆盖 import 语句后定义的设置。

   你也可以更进一步。比如，你可以重现一些 `weblate.docker_settings` 所做的事 <https://github.com/WeblateOrg/weblate/blob/main/weblate/settings_docker.py>‘__，如将设置暴露为环境变量，或允许在数据卷中覆盖来自 Python 文件的设置。

替换标志和其它静态文件

在 3.8-5 版本加入.

Weblate 附带的静态文件可以通过放置到 `/app/data/python/customize/static` 中来覆盖（请参阅*Docker 容器卷*）。例如，创建 `/app/data/python/customize/static/favicon.ico` 将替换 favicon。

提示： 在容器启动时，这些文件被复制到相应的位置，因此需要在更改卷的内容后重新启动 Weblate。

这种方法也可以用来覆盖 Weblate 的模板。例如 legal‘文件可以放在:file:/app/data/python/customize/templates/legal/documents‘里。

或者，您也可以包括自己的模块（请参阅定制 *Weblate*），并将其作为单独的卷添加到 Docker 容器中，例如：

```
weblate:
  volumes:
    - weblate-data:/app/data
    - ./weblate_customization/weblate_customization:/app/data/python/weblate_
↪customization
  environment:
    WEBLATE_ADD_APPS: weblate_customization
```

配置 PostgreSQL 服务器

PostgtreSQL 容器使用默认的 PostgreSQL 配置，它不会有效地利用你的 CPU 核心或内存。建议自定义配置以提高性能。

配置可以按照 https://hub.docker.com/_/postgres “数据库配置“中的描述来调整。匹配你的环境的配置可以使用 https://pgtune.leopard.in.ua/。

容器内部构件

容器在使用 **supervisor`来启动单独服务。遇到:ref:`docker-scaling**，它只在容器中启动一个服务。

要检查服务状态，请使用：

```
docker-compose exec --user weblate weblate supervisorctl status
```

每个 Celery 队列有单独服务（详见*使用 Celery 的后台任务*）。你可以停止相应的 worker 来停止处理某些任务：

```
docker-compose exec --user weblate weblate supervisorctl stop celery-translate
```

在 Debian 和 Ubantu 上安装

硬件要求

Weblate 应该可以在任何现代硬件上正常运行，以下是在单个主机（Weblate、数据库和 Web 服务器）上运行 Weblate 所需的最低配置：

- 3 GB 的内存
- 2 个 CPU 核心
- 1 GB 的存储空间

内存越多越好——用于所有级别的缓存（文件系统，数据库和 Weblate ）。

许多并发用户会增加所需的 CPU 内核数量。对于数百个翻译部件，推荐至少有 4 GB 的内存。

典型的数据库存储用量大约为每 1 百万单词 300 MB。克隆仓库所需的存储空间会变化，但 Weblate 试图通过浅克隆将其大小最小化。

备注： 安装 Weblate 的实际要求会因其中管理的翻译规模而大不相同。

安装

系统要求

安装构建 Python 模块所需的依赖项（请参见*软件要求*）：

```
apt install -y \
   libxml2-dev libxslt-dev libfreetype6-dev libjpeg-dev libz-dev libyaml-dev \
   libffi-dev libcairo-dev gir1.2-pango-1.0 libgirepository1.0-dev \
   libacl1-dev libssl-dev libpq-dev libjpeg-dev build-essential \
   python3-gdbm python3-dev python3-pip python3-virtualenv virtualenv git
```

根据您打算使用的功能来安装所需的可选依赖项（参见*可选依赖项*）：

```
apt install -y \
   tesseract-ocr libtesseract-dev libleptonica-dev \
   libldap2-dev libldap-common libsasl2-dev \
   libxmlsec1-dev
```

可选地安装生产服务器运行所需要的软件，参见运行服务器 、*Weblate* 的数据库设置 、使用 *Celery* 的后台任务。根据于您的安装所占的空间，您会想要在特定的服务器上运行这些部件。

本地安装的使用说明：

```
# Web server option 1: NGINX and uWSGI
apt install -y nginx uwsgi uwsgi-plugin-python3

# Web server option 2: Apache with ``mod_wsgi``
apt install -y apache2 libapache2-mod-wsgi-py3

# Caching backend: Redis
apt install -y redis-server

# Database server: PostgreSQL
apt install -y postgresql postgresql-contrib

# SMTP server
apt install -y exim4
```

Python 模块

提示： 我们使用 virtualenv 将 Weblate 安装在一个与您的系统分离的环境中。如果您不熟悉它，查阅 virtualenv User Guide。

1. 为 Weblate 新建 virtualenv：

   ```
   virtualenv ~/weblate-env
   ```

2. 为 Weblate 激活 virtualevn ：

   ```
   . ~/weblate-env/bin/activate
   ```

3. 安装包含所有可选依赖项的 Weblate：

   ```
   # Install Weblate with all optional dependencies
   pip install "Weblate[all]"
   ```

   请查看可选依赖项 以了解对可选依赖关系的微调情况。

   备注： 在某些 Linux 发行版上，运行 Weblate 会出现 libffi 错误：

   ```
   ffi_prep_closure(): bad user_data (it seems that the version of the libffi␣
   ↪library seen at runtime is different from the 'ffi.h' file seen at compile-
   ↪time)
   ```

   这是由于通过 PyPI 发布的二进制包与该发行版不兼容造成的。为了解决这个问题，你需要在你的系统上重建该软件包：

   ```
   pip install --force-reinstall --no-binary :all: cffi
   ```

配置 Weblate

备注： 以下内容假设 Weblate 使用的 virtualenv 已经激活(可以通过 `. ~/weblate-env/bin/activate` 来实现)。如果没有，请指定 **weblate** 命令的完全路径为 `~/weblate-env/bin/weblate`。

1. 将文件 `~/weblate-env/lib/python3.9/site-packages/weblate/settings_example.py` 复制为 `~/weblate-env/lib/python3.9/site-packages/weblate/settings.py`。
2. 将新的 `settings.py` 文件中的值调整为您所需要的。您至少需要提供数据库凭据和 Django 密钥，但在生产安装中需要做更多的更改，参见：调整配置。
3. 为 Weblate 新建数据库及其结构（示例中的设置使用 PostgreSQL，已经准备好的生产设置请查看*Weblate 的数据库设置* ）：

```
weblate migrate
```

4. 为管理员用户创建一个账户并将其密码复制到剪贴板，并保存该账户以供以后使用：

```
weblate createadmin
```

5. 为你的收集 Web 服务器用的静态文件（请参见运行服务器 和为静态文件提供服务 ）：

```
weblate collectstatic
```

6. 压缩 JavaScript 和 CSS 文件（可选步骤，请参见压缩客户端资源 ）：

```
weblate compress
```

7. 启动 Celery workers。这对开发来说是不必要的，但强烈建议在其他场景这样做。更多信息请参见使用 *Celery* 的后台任务：

```
~/weblate-env/lib/python3.9/site-packages/weblate/examples/celery start
```

8. 启动开发服务器（:ref:‘server‘详细说明生产设置）：

```
weblate runserver
```

安装后

配置，您的 Weblate 服务器现在运行了，您可以使用它来启动。

- 您可以在 `http://localhost:8000/` 访问 Weblate。
- 使用安装期间获得的管理凭据登录或注册新用户。
- 现在，您可以在 Weblate virtualenv 活动时使用 **weblate** 命令来运行 Weblate 命令。
- 您可以使用 Ctrl+C 来停止测试的服务器。
- 在 `/manage/performance/` URL （参见管理界面）上或使用 **weblate check --deploy** 来复查您安装的潜在问题，请参见生产设置。

添加翻译

1. 打开管理界面（`http://localhost:8000/create/project/`），并新建您想要翻译的项目。更多细节请参见项目配置。

   这里所有需要您指定的只是项目名称及其网站。

2. 新建一个部件，该部件是要翻译的实际对象——它指向版本控制系统（VCS）仓库，并选择要翻译的文件。更多详细信息请参见部件配置。

   此处的重要字段为：部件名称、源代码仓库 及用于寻找可翻译文件的文件掩码。Weblate 支持多种格式，包括*GNU gettext*、*Android* 字符串资源、苹果 *iOS* 字符串、*Java* 属性、*Stringsdict* 格式 或*Fluent* 格式，更多细节见支持的文件格式。

3. 一旦完成上面的工作（根据您 VCS 仓库的大小，以及需要翻译的信息数量，这可能是个漫长的过程），您就可以开始翻译了。

在 SUSE 和 openSUSE 上安装

硬件要求

Weblate 应该可以在任何现代硬件上正常运行，以下是在单个主机（Weblate、数据库和 Web 服务器）上运行 Weblate 所需的最低配置：

- 3 GB 的内存
- 2 个 CPU 核心
- 1 GB 的存储空间

内存越多越好——用于所有级别的缓存（文件系统，数据库和 Weblate ）。

许多并发用户会增加所需的 CPU 内核数量。对于数百个翻译部件，推荐至少有 4 GB 的内存。

典型的数据库存储用量大约为每 1 百万单词 300 MB。克隆仓库所需的存储空间会变化，但 Weblate 试图通过浅克隆将其大小最小化。

备注： 安装 Weblate 的实际要求会因其中管理的翻译规模而大不相同。

安装

系统要求

安装构建 Python 模块所需的依赖项（请参见软件要求）：

```
zypper install \
   libxslt-devel libxml2-devel freetype-devel libjpeg-devel zlib-devel \
   libyaml-devel libffi-devel cairo-devel pango-devel \
   gobject-introspection-devel libacl-devel python3-pip python3-virtualenv \
   python3-devel git
```

根据您打算使用的功能来安装所需的可选依赖项（参见可选依赖项）：

```
zypper install tesseract-ocr tesseract-devel leptonica-devel
zypper install libldap2-devel libsasl2-devel
zypper install libxmlsec1-devel
```

可选地安装生产服务器运行所需要的软件，参见运行服务器 、*Weblate* 的数据库设置 、使用 *Celery* 的后台任务。根据于您的安装所占的空间，您会想要在特定的服务器上运行这些部件。

本地安装的使用说明：

```
# Web server option 1: NGINX and uWSGI
zypper install nginx uwsgi uwsgi-plugin-python3

# Web server option 2: Apache with ``mod_wsgi``
zypper install apache2 apache2-mod_wsgi

# Caching backend: Redis
zypper install redis-server

# Database server: PostgreSQL
zypper install postgresql postgresql-contrib

# SMTP server
zypper install postfix
```

Python 模块

提示： 我们使用 virtualenv 将 Weblate 安装在一个与您的系统分离的环境中。如果您不熟悉它，查阅 virtualenv User Guide。

1. 为 Weblate 新建 virtualenv：

   ```
   virtualenv ~/weblate-env
   ```

2. 为 Weblate 激活 virtualevn ：

   ```
   . ~/weblate-env/bin/activate
   ```

3. 安装包含所有可选依赖项的 Weblate：

   ```
   # Install Weblate with all optional dependencies
   pip install "Weblate[all]"
   ```

   请查看可选依赖项 以了解对可选依赖关系的微调情况。

   备注： 在某些 Linux 发行版上，运行 Weblate 会出现 libffi 错误：

   ```
   ffi_prep_closure(): bad user_data (it seems that the version of the libffi␣
   ↪library seen at runtime is different from the 'ffi.h' file seen at compile-
   ↪time)
   ```

   这是由于通过 PyPI 发布的二进制包与该发行版不兼容造成的。为了解决这个问题，你需要在你的系统上重建该软件包：

   ```
   pip install --force-reinstall --no-binary :all: cffi
   ```

配置 Weblate

备注： 以下内容假设 Weblate 使用的 virtualenv 已经激活(可以通过 `. ~/weblate-env/bin/activate` 来实现)。如果没有，请指定 **`weblate`** 命令的完全路径为 `~/weblate-env/bin/weblate`。

1. 将文件 `~/weblate-env/lib/python3.9/site-packages/weblate/settings_example.py` 复制为 `~/weblate-env/lib/python3.9/site-packages/weblate/settings.py`。
2. 将新的 `settings.py` 文件中的值调整为您所需要的。您至少需要提供数据库凭据和 Django 密钥，但在生产安装中需要做更多的更改，参见：调整配置。
3. 为 Weblate 新建数据库及其结构（示例中的设置使用 PostgreSQL，已经准备好的生产设置请查看*Weblate 的数据库设置*）：

```
weblate migrate
```

4. 为管理员用户创建一个账户并将其密码复制到剪贴板，并保存该账户以供以后使用：

```
weblate createadmin
```

5. 为你的收集 Web 服务器用的静态文件（请参见运行服务器 和为静态文件提供服务）：

```
weblate collectstatic
```

6. 压缩 JavaScript 和 CSS 文件（可选步骤，请参见压缩客户端资源）：

```
weblate compress
```

7. 启动 Celery workers。这对开发来说是不必要的，但强烈建议在其他场景这样做。更多信息请参见使用 *Celery* 的后台任务：

```
~/weblate-env/lib/python3.9/site-packages/weblate/examples/celery start
```

8. 启动开发服务器（:ref:‘server‘详细说明生产设置）：

```
weblate runserver
```

安装后

配置，您的 Weblate 服务器现在运行了，您可以使用它来启动。

- 您可以在 `http://localhost:8000/` 访问 Weblate。
- 使用安装期间获得的管理凭据登录或注册新用户。
- 现在，您可以在 Weblate virtualenv 活动时使用 **`weblate`** 命令来运行 Weblate 命令。
- 您可以使用 Ctrl+C 来停止测试的服务器。
- 在 `/manage/performance/` URL （参见管理界面）上或使用 **`weblate check --deploy`** 来复查您安装的潜在问题，请参见生产设置。

添加翻译

1. 打开管理界面（`http://localhost:8000/create/project/`），并新建您想要翻译的项目。更多细节请参见项目配置。

   这里所有需要您指定的只是项目名称及其网站。

2. 新建一个部件，该部件是要翻译的实际对象——它指向版本控制系统（VCS）仓库，并选择要翻译的文件。更多详细信息请参见部件配置。

   此处的重要字段为：部件名称、源代码仓库 及用于寻找可翻译文件的文件掩码。Weblate 支持多种格式，包括*GNU gettext*、*Android* 字符串资源、苹果 *iOS* 字符串、*Java* 属性、*Stringsdict* 格式 或*Fluent* 格式，更多细节见支持的文件格式。

3. 一旦完成上面的工作（根据您 VCS 仓库的大小，以及需要翻译的信息数量，这可能是个漫长的过程），您就可以开始翻译了。

在 Redhat、Fedora 和 CentOS 上安装

硬件要求

Weblate 应该可以在任何现代硬件上正常运行，以下是在单个主机（Weblate、数据库和 Web 服务器）上运行 Weblate 所需的最低配置：

- 3 GB 的内存
- 2 个 CPU 核心
- 1 GB 的存储空间

内存越多越好——用于所有级别的缓存（文件系统，数据库和 Weblate ）。

许多并发用户会增加所需的 CPU 内核数量。对于数百个翻译部件，推荐至少有 4 GB 的内存。

典型的数据库存储用量大约为每 1 百万单词 300 MB。克隆仓库所需的存储空间会变化，但 Weblate 试图通过浅克隆将其大小最小化。

备注： 安装 Weblate 的实际要求会因其中管理的翻译规模而大不相同。

安装

系统要求

安装构建 Python 模块所需的依赖项（请参见软件要求）：

```
dnf install \
   libxslt-devel libxml2-devel freetype-devel libjpeg-devel zlib-devel \
   libyaml-devel libffi-devel cairo-devel pango-devel \
   gobject-introspection-devel libacl-devel python3-pip python3-virtualenv \
   python3-devel git
```

根据您打算使用的功能来安装所需的可选依赖项（参见可选依赖项）：

```
dnf install tesseract-langpack-eng tesseract-devel leptonica-devel
dnf install libldap2-devel libsasl2-devel
dnf install libxmlsec1-devel
```

可选地安装生产服务器运行所需要的软件，参见运行服务器 、*Weblate* 的数据库设置 、使用 *Celery* 的后台任务。根据于您的安装所占的空间，您会想要在特定的服务器上运行这些部件。

本地安装的使用说明：

```
# Web server option 1: NGINX and uWSGI
dnf install nginx uwsgi uwsgi-plugin-python3

# Web server option 2: Apache with ``mod_wsgi``
dnf install apache2 apache2-mod_wsgi

# Caching backend: Redis
dnf install redis

# Database server: PostgreSQL
dnf install postgresql postgresql-contrib

# SMTP server
dnf install postfix
```

Python 模块

提示： 我们使用 virtualenv 将 Weblate 安装在一个与您的系统分离的环境中。如果您不熟悉它，查阅 virtualenv User Guide。

1. 为 Weblate 新建 virtualenv：

   ```
   virtualenv ~/weblate-env
   ```

2. 为 Weblate 激活 virtualevn ：

   ```
   . ~/weblate-env/bin/activate
   ```

3. 安装包含所有可选依赖项的 Weblate：

   ```
   # Install Weblate with all optional dependencies
   pip install "Weblate[all]"
   ```

   请查看可选依赖项 以了解对可选依赖关系的微调情况。

   备注： 在某些 Linux 发行版上，运行 Weblate 会出现 libffi 错误：

   ```
   ffi_prep_closure(): bad user_data (it seems that the version of the libffi␣
   ↪library seen at runtime is different from the 'ffi.h' file seen at compile-
   ↪time)
   ```

   这是由于通过 PyPI 发布的二进制包与该发行版不兼容造成的。为了解决这个问题，你需要在你的系统上重建该软件包：

   ```
   pip install --force-reinstall --no-binary :all: cffi
   ```

配置 Weblate

备注： 以下内容假设 Weblate 使用的 virtualenv 已经激活(可以通过 `. ~/weblate-env/bin/activate` 来实现)。如果没有，请指定 **weblate** 命令的完全路径为 `~/weblate-env/bin/weblate`。

1. 将文件 `~/weblate-env/lib/python3.9/site-packages/weblate/settings_example.py` 复制为 `~/weblate-env/lib/python3.9/site-packages/weblate/settings.py`。
2. 将新的 `settings.py` 文件中的值调整为您所需要的。您至少需要提供数据库凭据和 Django 密钥，但在生产安装中需要做更多的更改，参见：调整配置。
3. 为 Weblate 新建数据库及其结构（示例中的设置使用 PostgreSQL，已经准备好的生产设置请查看*Weblate 的数据库设置*）：

```
weblate migrate
```

4. 为管理员用户创建一个账户并将其密码复制到剪贴板，并保存该账户以供以后使用：

```
weblate createadmin
```

5. 为你的收集 Web 服务器用的静态文件（请参见运行服务器 和为静态文件提供服务）：

```
weblate collectstatic
```

6. 压缩 JavaScript 和 CSS 文件（可选步骤，请参见压缩客户端资源）：

```
weblate compress
```

7. 启动 Celery workers。这对开发来说是不必要的，但强烈建议在其他场景这样做。更多信息请参见使用 *Celery* 的后台任务：

```
~/weblate-env/lib/python3.9/site-packages/weblate/examples/celery start
```

8. 启动开发服务器（:ref:‘server‘详细说明生产设置）：

```
weblate runserver
```

安装后

配置，您的 Weblate 服务器现在运行了，您可以使用它来启动。

- 您可以在 `http://localhost:8000/` 访问 Weblate。
- 使用安装期间获得的管理凭据登录或注册新用户。
- 现在，您可以在 Weblate virtualenv 活动时使用 **weblate** 命令来运行 Weblate 命令。
- 您可以使用 Ctrl+C 来停止测试的服务器。
- 在 `/manage/performance/` URL （参见管理界面）上或使用 **weblate check --deploy** 来复查您安装的潜在问题，请参见生产设置。

添加翻译

1. 打开管理界面（`http://localhost:8000/create/project/`），并新建您想要翻译的项目。更多细节请参见项目配置。

   这里所有需要您指定的只是项目名称及其网站。

2. 新建一个部件，该部件是要翻译的实际对象——它指向版本控制系统（VCS）仓库，并选择要翻译的文件。更多详细信息请参见部件配置。

   此处的重要字段为：部件名称、源代码仓库 及用于寻找可翻译文件的文件掩码。Weblate 支持多种格式，包括*GNU gettext*、*Android* 字符串资源、苹果 *iOS* 字符串、*Java* 属性、*Stringsdict* 格式 或*Fluent* 格式，更多细节见支持的文件格式。

3. 一旦完成上面的工作（根据您 VCS 仓库的大小，以及需要翻译的信息数量，这可能是个漫长的过程），您就可以开始翻译了。

在 macOS 上安装

硬件要求

Weblate 应该可以在任何现代硬件上正常运行，以下是在单个主机（Weblate、数据库和 Web 服务器）上运行 Weblate 所需的最低配置：

- 3 GB 的内存
- 2 个 CPU 核心
- 1 GB 的存储空间

内存越多越好——用于所有级别的缓存（文件系统，数据库和 Weblate ）。

许多并发用户会增加所需的 CPU 内核数量。对于数百个翻译部件，推荐至少有 4 GB 的内存。

典型的数据库存储用量大约为每 1 百万单词 300 MB。克隆仓库所需的存储空间会变化，但 Weblate 试图通过浅克隆将其大小最小化。

备注： 安装 Weblate 的实际要求会因其中管理的翻译规模而大不相同。

安装

系统要求

安装构建 Python 模块所需的依赖项（请参见软件要求）：

```
brew install python pango cairo gobject-introspection libffi glib libyaml
pip install virtualenv
```

确认 pip 能够找到 homebrew 提供的 `libffi` 和 `openssl` 版本——这在安装构建步骤中是必需的。

```
export PKG_CONFIG_PATH="/usr/local/opt/libffi/lib/pkgconfig:/usr/local/opt/
↪openssl@3/lib/pkgconfig"
```

根据您打算使用的功能来安装所需的可选依赖项（参见可选依赖项）：

```
brew install tesseract
```

可选地安装生产服务器运行所需要的软件，参见运行服务器 、*Weblate* 的数据库设置 、使用 *Celery* 的后台任务。根据于您的安装所占的空间，您会想要在特定的服务器上运行这些部件。

本地安装的使用说明：

```
# Web server option 1: NGINX and uWSGI
brew install nginx uwsgi

# Web server option 2: Apache with ``mod_wsgi``
brew install httpd

# Caching backend: Redis
brew install redis

# Database server: PostgreSQL
brew install postgresql
```

Python 模块

提示： 我们使用 virtualenv 将 Weblate 安装在一个与您的系统分离的环境中。如果您不熟悉它，查阅 virtualenv User Guide。

1. 为 Weblate 新建 virtualenv：

   ```
   virtualenv ~/weblate-env
   ```

2. 为 Weblate 激活 virtualevn：

   ```
   . ~/weblate-env/bin/activate
   ```

3. 安装包含所有可选依赖项的 Weblate：

   ```
   # Install Weblate with all optional dependencies
   pip install "Weblate[all]"
   ```

   请查看可选依赖项 以了解对可选依赖关系的微调情况。

   备注： 在某些 Linux 发行版上，运行 Weblate 会出现 libffi 错误：

   ```
   ffi_prep_closure(): bad user_data (it seems that the version of the libffi␣
   ↪library seen at runtime is different from the 'ffi.h' file seen at compile-
   ↪time)
   ```

   这是由于通过 PyPI 发布的二进制包与该发行版不兼容造成的。为了解决这个问题，你需要在你的系统上重建该软件包：

   ```
   pip install --force-reinstall --no-binary :all: cffi
   ```

配置 Weblate

备注： 以下内容假设 Weblate 使用的 virtualenv 已经激活(可以通过 `. ~/weblate-env/bin/activate` 来实现)。如果没有，请指定 **weblate** 命令的完全路径为 `~/weblate-env/bin/weblate`。

1. 将文件 `~/weblate-env/lib/python3.9/site-packages/weblate/settings_example.py` 复制为 `~/weblate-env/lib/python3.9/site-packages/weblate/settings.py`。
2. 将新的 `settings.py` 文件中的值调整为您所需要的。您至少需要提供数据库凭据和 Django 密钥，但在生产安装中需要做更多的更改，参见：调整配置。
3. 为 Weblate 新建数据库及其结构（示例中的设置使用 PostgreSQL，已经准备好的生产设置请查看*Weblate 的数据库设置*）：

```
weblate migrate
```

4. 为管理员用户创建一个账户并将其密码复制到剪贴板，并保存该账户以供以后使用：

```
weblate createadmin
```

5. 为你的收集 Web 服务器用的静态文件（请参见运行服务器 和为静态文件提供服务）：

```
weblate collectstatic
```

6. 压缩 JavaScript 和 CSS 文件（可选步骤，请参见压缩客户端资源）：

```
weblate compress
```

7. 启动 Celery workers。这对开发来说是不必要的，但强烈建议在其他场景这样做。更多信息请参见使用 *Celery* 的后台任务：

```
~/weblate-env/lib/python3.9/site-packages/weblate/examples/celery start
```

8. 启动开发服务器（:ref:‘server‘详细说明生产设置）：

```
weblate runserver
```

安装后

配置，您的 Weblate 服务器现在运行了，您可以使用它来启动。

- 您可以在 `http://localhost:8000/` 访问 Weblate。
- 使用安装期间获得的管理凭据登录或注册新用户。
- 现在，您可以在 Weblate virtualenv 活动时使用 **weblate** 命令来运行 Weblate 命令。
- 您可以使用 Ctrl+C 来停止测试的服务器。
- 在 `/manage/performance/` URL （参见管理界面）上或使用 **weblate check --deploy** 来复查您安装的潜在问题，请参见生产设置。

添加翻译

1. 打开管理界面（`http://localhost:8000/create/project/`），并新建您想要翻译的项目。更多细节请参见项目配置。

   这里所有需要您指定的只是项目名称及其网站。

2. 新建一个部件，该部件是要翻译的实际对象——它指向版本控制系统（VCS）仓库，并选择要翻译的文件。更多详细信息请参见部件配置。

   此处的重要字段为：部件名称、源代码仓库 及用于寻找可翻译文件的文件掩码。Weblate 支持多种格式，包括*GNU gettext*、*Android* 字符串资源、苹果 *iOS* 字符串、*Java* 属性、*Stringsdict* 格式 或*Fluent* 格式，更多细节见支持的文件格式。

3. 一旦完成上面的工作（根据您 VCS 仓库的大小，以及需要翻译的信息数量，这可能是个漫长的过程），您就可以开始翻译了。

从源码安装

1. 安装 Weblate 前，请先遵照用于您系统的安装说明：

   - 在 *Debian* 和 *Ubantu* 上安装
   - 在 *SUSE* 和 *openSUSE* 上安装
   - 在 *Redhat*、*Fedora* 和 *CentOS* 上安装

2. 使用 Git 获取最新的 Weblate 源码（或下载 tarball 包并将其解压）：

   ```
   git clone https://github.com/WeblateOrg/weblate.git weblate-src
   ```

   另外，您可以使用发布的压缩包。您可以从我们的网站 <https://weblate.org/> 下载。这些下载是经过加密签名的，请参见验证发布签名。

3. 将当前的 Weblate 代码安装到 virtualenv 中：

   ```
   . ~/weblate-env/bin/activate
   pip install -e weblate-src
   ```

4. 将 `weblate/settings_example.py` 复制为 `weblate/settings.py`。

5. 将新的 `settings.py` 文件中的值调整为您所需要的。您至少需要提供数据库凭据和 Django 密钥，但在生产安装中需要做更多的更改，参见：调整配置。

6. 新建 Weblate 使用的数据库，参见*Weblate* 的数据库设置。

7. 构建 Django 表、静态文件和初始数据（参见填满数据库 和为静态文件提供服务）：

   ```
   weblate migrate
   weblate collectstatic
   weblate compress
   ```

   备注： 无论任何时候更新仓库时，都应该重复这一步骤。

在 OpenShift 上安装

使用 OpenShift Weblate 模板，您可以在几秒钟内启动并运行您的个人 Weblate 实例。Weblate 的所有依赖项都已经包含在内。PostgreSQL 被设置为默认数据库，并且使用持久化卷声明。

你可以在 <https://github.com/WeblateOrg/openshift/> 找到模板。

安装

下面的示例假设您有一个正常运作的 OpenShift v3.x 环境，它已经安装 “oc“客户端工具。请查看 OpenShift 文档中的说明。

`template.yml`适合在 OpenShift 中运行所有组件。还有:file:`template-external-postgresql.yml`，它不启动 PostgreSQL 服务器，允许你配置外部 PostgreSQL 服务器。

Web 控制台

从 template.yml 复制原始内容，并将它们导入你的项目，然后在 OpenShift web 控制台使用 `Create` 按钮来新建你的应用。web 控制台将提示你模板使用的所有参数的值。

CLI

为了将 Weblate 模板上传到你当前项目的模板库中，使用后面的命令传递 `template.yml` 文件:

```
$ oc create -f https://raw.githubusercontent.com/WeblateOrg/openshift/main/
↪template.yml \
   -n <PROJECT>
```

现在模板可以使用 CLI 的 web 控制台以供选择。

参数

模板的 parameters 部分列出了你可以覆盖的参数。你可以通过使用后面的命令并指定要使用的文件通过 CLI 列出它们:

```
$ oc process --parameters -f https://raw.githubusercontent.com/WeblateOrg/
↪openshift/main/template.yml

# If the template is already uploaded
$ oc process --parameters -n <PROJECT> weblate
```

服务开通

还可以使用 CLI 来处理模板，并使用生成的配置来立即新建对象。

```
$ oc process -f https://raw.githubusercontent.com/WeblateOrg/openshift/main/
↪template.yml \
    -p APPLICATION_NAME=weblate \
    -p WEBLATE_VERSION=4.3.1-1 \
    -p WEBLATE_SITE_DOMAIN=weblate.app-openshift.example.com \
    -p POSTGRESQL_IMAGE=docker-registry.default.svc:5000/openshift/postgresql:9.6 \
    -p REDIS_IMAGE=docker-registry.default.svc:5000/openshift/redis:3.2 \
    | oc create -f
```

在成功地迁移并部署特定的 WEBLATE_SITE_DOMAIN 参数后，Weblate 实例就应该可用了。

设置容器之后，您可以使用 WEBLATE_ADMIN_PASSWORD 中提供的密码以 管理员 用户身份登录，或者如果未设置密码，则使用首次启动时生成的随机密码。

若要重置 管理员 密码，请在容器各自的 Secret 中将 WEBLATE_ADMIN_PASSWORD 设置为新密码，然后重启容器。

消除

```
$ oc delete all -l app=<APPLICATION_NAME>
$ oc delete configmap -l app= <APPLICATION_NAME>
$ oc delete secret -l app=<APPLICATION_NAME>
# ATTTENTION! The following command is only optional and will permanently delete␣
↪all of your data.
$ oc delete pvc -l app=<APPLICATION_NAME>

$ oc delete all -l app=weblate \
    && oc delete secret -l app=weblate \
    && oc delete configmap -l app=weblate \
    && oc delete pvc -l app=weblate
```

配置

通过处理模板，将新建各自的 ConfigMap，并且可以用于定制 Weblate 镜像。ConfigMap 直接作为环境变量挂载，并且在每次更改时触发新的部署。对于更多配置选项，环境变量的完整列表请参见*Docker 环境变量*。

在 Kubernetes 上安装

备注： 本手册寻找有 Kubernetes 经验的贡献者来详细介绍安装过程。

凭借 Kubernetes Helm 图表，您可以在几秒钟内启动并运行您的个人 Weblate 实例。Weblate 的所有依赖项都已经包含在内。PostgreSQL 被设置为默认数据库，并且使用持久化卷声明。

您可以在 <https://github.com/WeblateOrg/helm/> 找到图表，并可以在 <https://artifacthub.io/packages/helm/weblate/weblate> 显示。

安装

```
helm repo add weblate https://helm.weblate.org
helm install my-release weblate/weblate
```

配置

进一步的配置选项，请参阅*Docker 环境变量* 以获取环境变量的完整列表。

根据你的设置和经验，选择一个适合你的安装方法：

- *使用 Docker 安装*，推荐用于生产设置。
- Virtualenv 安装，推荐用于生产设置：
  - *在 Debian 和 Ubantu 上安装*
  - *在 SUSE 和 openSUSE 上安装*
  - *在 Redhat、Fedora 和 CentOS 上安装*
  - *在 macOS 上安装*
- *从源码安装*，推荐用于开发。
- *在 OpenShift 上安装*
- *在 Kubernetes 上安装*

2.1.2 软件要求

操作系统

目前已知 Weblate 可运行在 Linux、FreeBSD 和 macOS 上。其他类 Unix 的系统也很可能支持运行。

Windows 不支持 Weblate。但 Weblate 可能仍然可以工作，并且很乐意接受补丁。

其他服务

Weblate 为其操作使用其他服务。至少需要后面的服务运行：

- PostgreSQL 数据库服务器，请参见*Weblate 的数据库设置*。
- Redis 服务器，用于缓存和任务队列，请参见*使用 Celery 的后台任务*。
- SMTP 服务器，用于发送电子邮件，请参见*配置电子邮件发件箱*。

Python 依赖项

Weblate 是用 Python 编写的，并且支持 Python 3.6 或更新版本。可以使用 pip 或你的分发包来安装依赖项，完整列表可在 `requirements.txt` 中找到。

最重要的依赖项：

Django
https://www.djangoproject.com/

Celery
https://docs.celeryq.dev/

Translate Toolkit（翻译工具包）
https://toolkit.translatehouse.org/

translation-finder
https://github.com/WeblateOrg/translation-finder

Python 社交认证
https://python-social-auth.readthedocs.io/

Django REST 框架
https://www.django-rest-framework.org/

可选依赖项

以下模块是某些 Weblate 功能所必需的。你可以在 `requirements-optional.txt` 中找到所有这些模块。

`Mercurial`（可选的*Mercurial* 仓库支持）
https://www.mercurial-scm.org/

`phply`（可选的*PHP* 字符串）
https://github.com/viraptor/phply

`tesserocr`（可选的字符串的可视化上下文 OCR）
https://github.com/sirfz/tesserocr

`python-akismet`（可选的针对垃圾电子邮件的保护）
https://github.com/Nekmo/python-akismet

`ruamel.yaml`（可选的*YAML* 文件）
https://pypi.org/project/ruamel.yaml/

`Zeep`（可选的微软术语服务）
https://docs.python-zeep.org/

`aeidon`（可选的字幕文件）
https://pypi.org/project/aeidon/

`fluent.syntax`（可选的*Fluent* 格式）
https://projectfluent.org/

提示： 使用 pip 进行安装时，您可以在安装时直接指定所需的功能：

```
pip install "Weblate[PHP,Fluent]"
```

或者你可以安装具有所有可选功能的 Weblate：

```
pip install "Weblate[all]"
```

或者你可以安装没有任何可选功能的 Weblate：

```
pip install Weblate
```

数据库后端依赖项

Weblate 支持 PostgreSQL、MySQL 和 MariaDB，更多细节请参见*Weblate 的数据库设置* 和后端文档。

其他系统要求

必须在系统上安装以下依赖项：

`Git`
https://git-scm.com/

Pango、Cairo 及相关的头文件和 GObject 内省数据
https://cairographics.org/，https://pango.gnome.org/，请参见*Pango 和 Cairo*

`git-review`（可选的 Gerrit 支持）
https://pypi.org/project/git-review/

`git-svn`（可选的 Subversion 支持）
https://git-scm.com/docs/git-svn

tesseract 及其数据（可选的截屏 OCR）
https://github.com/tesseract-ocr/tesseract

licensee （可选，用于创建部件时检测许可证）
https://github.com/licensee/licensee

构建时的依赖项

为了构建一些*Python 依赖项*，你可能需要安装其依赖项。这取决于你如何安装它们，因此请查阅各个包的文档。如果你使用预构建的 `Wheels` 并使用 `pip` 安装或使用分发包时，则不需要那些依赖项。

Pango 和 Cairo

在 3.7 版本发生变更.

Weblate 使用 Pango 和 Cairo 来生成位图小挂件（请参见 promotion ）并提供检查（请参见管理字型 ）。要正确地安装 Python 绑定，需要首先安装系统库——Cairo 和 Pango 都是需要的，而它们又需要 GLib。所有这些都应该与开发文件和 GObject 内省数据一起安装。

2.1.3 验证发布签名

Weblate 的发布由发布开发者通过密码签发。当前是 Michal Čihař。他的 PGP 密钥指纹是：

```
63CB 1DF1 EF12 CF2A C0EE 5A32 9C27 B313 42B7 511D
```

还可以从 <https://keybase.io/nijel> 获取更多身份信息。

您应该验证签名是否与您所下载的压缩包相符，这样可以确保使用的代码与发布的代码相同。您还应该验证签名的日期，以确保下载的是最新版本。

每个压缩包都带有 `.asc` 文件，其中包含它的 PGP 签名。将它们放在同一个文件夹中后，就可以验证签名了：

```
$ gpg --verify Weblate-3.5.tar.xz.asc
gpg: assuming signed data in 'Weblate-3.5.tar.xz'
gpg: Signature made Ne 3. března 2019, 16:43:15 CET
gpg:                using RSA key 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: Can't check signature: public key not found
```

正如你所看到的，GPG 抱怨它不知道公钥。此时应该执行以下步骤之一：

- 使用 *wkd* 来下载密钥：

```
$ gpg --auto-key-locate wkd --locate-keys michal@cihar.com
pub   rsa4096 2009-06-17 [SC]
      63CB1DF1EF12CF2AC0EE5A329C27B31342B7511D
uid           [ultimate] Michal Čihař <michal@cihar.com>
uid           [ultimate] Michal Čihař <nijel@debian.org>
uid           [ultimate] [jpeg image of size 8848]
uid           [ultimate] Michal Čihař (Braiins) <michal.cihar@braiins.cz>
sub   rsa4096 2009-06-17 [E]
sub   rsa4096 2015-09-09 [S]
```

- 从 Michal 的服务器 下载密钥环，然后将其导入：

```
$ gpg --import wmxth3chu9jfxdxywj1skpmhsj311mzm
```

- 从一个密钥服务器下载并导入密钥：

```
$ gpg --keyserver hkp://pgp.mit.edu --recv-keys␣
↪87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: key 9C27B31342B7511D: "Michal Čihař <michal@cihar.com>" imported
gpg: Total number processed: 1
gpg:               unchanged: 1
```

这会将情况改善一点——在这点上可以验证给定密钥的签名是正确的，但仍然不能相信密钥中使用的名称：

```
$ gpg --verify Weblate-3.5.tar.xz.asc
gpg: assuming signed data in 'Weblate-3.5.tar.xz'
gpg: Signature made Ne 3. března 2019, 16:43:15 CET
gpg:                using RSA key 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: Good signature from "Michal Čihař <michal@cihar.com>" [ultimate]
gpg:                 aka "Michal Čihař <nijel@debian.org>" [ultimate]
gpg:                 aka "[jpeg image of size 8848]" [ultimate]
gpg:                 aka "Michal Čihař (Braiins) <michal.cihar@braiins.cz>"␣
↪[ultimate]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 63CB 1DF1 EF12 CF2A C0EE  5A32 9C27 B313 42B7 511D
```

这里的问题是，任何人都可以使用这个名称发放密钥。您需要确保密钥确实是为提到的人所拥有。GNU 隐私手册在 验证公共密钥环上的其他密钥 一章中介绍了这个主题。最可靠的方法是与开发者面对面交换密钥指纹，不过你也可以依靠信任网络。这样您就可以信任通过与开发者见过面的其他人的签名传递的密钥。

一旦信任了密钥，警告就不会发生：

```
$ gpg --verify Weblate-3.5.tar.xz.asc
gpg: assuming signed data in 'Weblate-3.5.tar.xz'
gpg: Signature made Sun Mar  3 16:43:15 2019 CET
gpg:                using RSA key 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: Good signature from "Michal Čihař <michal@cihar.com>" [ultimate]
gpg:                 aka "Michal Čihař <nijel@debian.org>" [ultimate]
gpg:                 aka "[jpeg image of size 8848]" [ultimate]
gpg:                 aka "Michal Čihař (Braiins) <michal.cihar@braiins.cz>"␣
↪[ultimate]
```

如果签名无效（压缩包已被更改），那么无论密钥是否可信，都会得到明显的错误：

```
$ gpg --verify Weblate-3.5.tar.xz.asc
gpg: Signature made Sun Mar  3 16:43:15 2019 CET
gpg:                using RSA key 87E673AF83F6C3A0C344C8C3F4AA229D4D58C245
gpg: BAD signature from "Michal Čihař <michal@cihar.com>" [ultimate]
```

2.1.4 文件系统权限

Weblate 进程需要能够读写它保存数据的目录——*DATA_DIR*。该目录下的所有文件都应该由运行所有 Weblate 进程的用户拥有和可写入（通常是 WSGI 和 Celery，请参见运行服务器 和使用 *Celery* 的后台任务）。

默认的配置放置在 Weblate 源的相同树下，然而你会想要将这些移动到更好的位置，如：`/var/lib/weblate`。

Weblate 试图自动建立这些文件夹，但当没有权限去执行时会失败。

当运行管理命令 时应该小心，它们应该由 Weblate 自己运行的相同用户来运行，否则一些文件的权限会是错误的。

在 Docker 容器中，`/app/data` 卷中的所有文件必须由容器内的 `weblate` 用户（UID 1000）拥有。

参见:

为静态文件提供服务

2.1.5 Weblate 的数据库设置

推荐使用 PostgreSQL 数据库服务器来运行 Weblate。

参见:

使用强力的数据库引擎, 数据库, 从其它数据库迁移到 *PostgreSQL*

PostgreSQL

PostgreSQL 通常是基于 Django 的网站的最好选择。它是实现 Django 数据库层而使用的参考数据库。

备注: Weblate 使用三字母的扩展名，在某些情况下需要单独安装。查找 `postgresql-contrib` 或类似命名的包。

参见:

PostgreSQL 注意事项

创建 PostgreSQL 数据库

在另一个单独的数据库中运行 Weblate，并将用户账户分开通常是个好方法:

```
# If PostgreSQL was not installed before, set the main password
sudo -u postgres psql postgres -c "\password postgres"

# Create a database user called "weblate"
sudo -u postgres createuser --superuser --pwprompt weblate

# Create the database "weblate" owned by "weblate"
sudo -u postgres createdb -E UTF8 -O weblate weblate
```

提示: 如果不想 Weblate 在 PostgreSQL 中使用超级用户，可以省略掉。在模式中必须作为 PostgreSQL 超级用户，来手动执行一些迁移步骤的情况下，Weblate 将使用:

```
CREATE EXTENSION IF NOT EXISTS pg_trgm WITH SCHEMA weblate;
CREATE EXTENSION IF NOT EXISTS btree_gin WITH SCHEMA weblate;
```

配置 Weblate 来使用 PostgreSQL

PostgreSQL 的 `settings.py` 片段:

```
DATABASES = {
    "default": {
        # Database engine
        "ENGINE": "django.db.backends.postgresql",
        # Database name
        "NAME": "weblate",
        # Database user
        "USER": "weblate",
```

(续下页)

(接上页)

```
        # Name of role to alter to set parameters in PostgreSQL,
        # use in case role name is different than user used for authentication.
        # "ALTER_ROLE": "weblate",
        # Database password
        "PASSWORD": "password",
        # Set to empty string for localhost
        "HOST": "database.example.com",
        # Set to empty string for default
        "PORT": "",
    }
}
```

数据库的合并执行了 Weblate 使用的 ALTER ROLE 数据库角色。在多数情况下，角色的名称与用户名匹配。在更富在的设置中，角色的名称与用户名不同，而在数据库合并过程中会得到不存在的角色的错误信息（`psycopg2.errors.UndefinedObject: role "weblate@hostname" does not exist`，角色“weblate@hostname”不存在）。已知这会在 PostgreSQL 的 Azure 数据库时发生，但并不仅限于这种环境。请将 `ALTER_ROLE` 设置为数据库合并过程中 Weblate 应该更改的角色的名称。

MySQL 和 MariaDB

提示： 一些 Weblate 功能使用*PostgreSQL* 会表现得更好。这包括搜索与翻译记忆库，它们都利用了数据库中的全文特性，而且 PostgreSQL 的实现更胜一筹。

Weblate 还可以使用 MySQL 或 MariaDB，使用与两个数据库相关的 Django 而导致的警告，请参见 MySQL 注意事项 和 MariaDB 注意事项。由于这些限制，我们建议对新安装使用*PostgreSQL*。

Weblate 需要至少 5.7.8 版的或至少 10.2.7 版的 MariaDB。

推荐 Weblate 使用后面的设置：

- 使用 `utf8mb4` 字符集来允许表示更高的 Unicode 平面（例如 emojis 表情符号）。
- 用 `innodb_large_prefix` 配置服务器，以允许在文本字段上有更长的索引。
- 设置隔离级别为 `READ COMMITTED`。
- SQL 模式应该设置为 `STRICT_TRANS_TABLES`。

MySQL 8.x、MariaDB 10.5.x 或更新版本具有合理的默认配置，因此无需调整服务器，所有需要的都可以在客户端进行配置。

下面是用于 8GB 内存服务器的 `/etc/my.cnf.d/server.cnf` 的示例。这些设置应该足以用于多数安装。MySQL 和 MariaDB 具有来提高服务器性能的可调整部分，除非计划有大量并发用户访问系统，否则这些可调整部分被认为是不必要的。这些细节请查看各厂商的文档。

在运行您的 Weblate 前设置好 `innodb_file_per_table` 设置并重启 MySQL/MariaDB，这对减少安装时的问题绝对重要。

```
[mysqld]
character-set-server = utf8mb4
character-set-client = utf8mb4
collation-server = utf8mb4_unicode_ci

datadir=/var/lib/mysql

log-error=/var/log/mariadb/mariadb.log

innodb_large_prefix=1
innodb_file_format=Barracuda
innodb_file_per_table=1
```

(续下页)

(接上页)

```
innodb_buffer_pool_size=2G
sql_mode=STRICT_TRANS_TABLES
```

提示： 如遇 `#1071 - Specified key was too long; max key length is 767 bytes` 错误, 请更新你的配置以包含上方的 `innodb` 设置并重新启动你的安装。

提示： 在得到错误信息 `#2006 - MySQL server has gone away` 的形况下，配置 `CONN_MAX_AGE` 可能会有帮助。

配置 Weblate 来使用 MySQL/MariaDB

MySQL 和 MariaDB 的 `settings.py` 片段：

```
DATABASES = {
    "default": {
        # Database engine
        "ENGINE": "django.db.backends.mysql",
        # Database name
        "NAME": "weblate",
        # Database user
        "USER": "weblate",
        # Database password
        "PASSWORD": "password",
        # Set to empty string for localhost
        "HOST": "127.0.0.1",
        # Set to empty string for default
        "PORT": "3306",
        # In case you wish to use additional
        # connection options
        "OPTIONS": {},
    }
}
```

开始安装前还应该在 MySQL 或 MariaDB 中创建 `weblate` 用户账户。使用下面的命令来实现：

```
GRANT ALL ON weblate.* to 'weblate'@'localhost' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

2.1.6 其他配置

配置电子邮件发件箱

Weblate 在各种情况下会发出电子邮件——用于激活账户，以及用户配置的各种通知。对于这些需要访问 SMTP 服务器。

电子邮件服务器使用这些设置进行配置: `EMAIL_HOST`, `EMAIL_HOST_PASSWORD`, `EMAIL_USE_TLS`, `EMAIL_USE_SSL`, `EMAIL_HOST_USER` and `EMAIL_PORT`。从名称就可以大概知道它们的含义，但是你可以在 Django 文档中找到更多信息。

提示： 在得到有关不支持的认证的情况下（例如 `SMTP AUTH extension not supported by server`），这最可能因为使用不安全的链接并且服务器拒绝以这种方式认证而导致。在这种情况下尝试启动 `EMAIL_USE_TLS`。

参见:

不接收来自 Weblate 的电子邮件, *Configuring outgoing e-mail in Docker container*

在反向代理之后运行

Weblate 的几个特性依赖于能够得到客户端 IP 地址。这包括频次限制 、针对垃圾电子邮件的保护 或审计日志。

在默认设置中，Weblate 从 WSGI 句柄设置的 REMOTE_ADDR 中解析 IP 地址。

在运行反向代理的情况下，这个字段很可能包含其地址。需要配置 Weblate 来信任附加的 HTTP 标头，并从中解析 IP 地址。这不能默认允许，因为在不使用反向代理的安装时，这会允许 IP 地址欺骗。允许*IP_BEHIND_REVERSE_PROXY* 对多数常见设置就足够了，但你还必须调整*IP_PROXY_HEADER* 和*IP_PROXY_OFFSET*。

另一件需要注意的事情是 Host header。它应该与*SITE_DOMAIN* 中的内容一致。在反向代理中可能需要其他配置（例如，在 Apache 使用 “ProxyPreserveHost On“，或在 nginx 中使用 `proxy_set_header Host $host;`）。

参见:

针对垃圾电子邮件的保护, 频次限制, 审计日志, *IP_BEHIND_REVERSE_PROXY*, *IP_PROXY_HEADER*, *IP_PROXY_OFFSET*, SECURE_PROXY_SSL_HEADER

HTTP 代理

Weblate 执行版本控制系统（VCS）命令，这些命令接受来自环境的代理配置。推荐在 `settings.py` 中定义代理设置:

```
import os

os.environ["http_proxy"] = "http://proxy.example.com:8080"
os.environ["HTTPS_PROXY"] = "http://proxy.example.com:8080"
```

参见:

代理环境变量

2.1.7 调整配置

参见:

配置的示例

将 `weblate/settings_example.py` 复制到 `weblate/settings.py`，并且调整它与你的设置匹配。你可能想要调整后面的选项： ADMINS

出现问题时接收通知站点管理员名单，例如合并失败或 Django 错误的通知。

参见:

ADMINS, 是当地配置管理设置

ALLOWED_HOSTS

需要设置这个，来列出你的网站支持服务的主机。例如:

```
ALLOWED_HOSTS = ["demo.weblate.org"]
```

另外可以包括通配符:

```
ALLOWED_HOSTS = ["*"]
```

参见:

ALLOWED_HOSTS, *WEBLATE_ALLOWED_HOSTS*, 允许主机设置

SESSION_ENGINE

配置如何存储会话。在保持默认的数据库后端引擎的情况下，应该安排 **weblate clearsessions** 从数据库中删除旧的会话。

如果使用 Redis 作为缓存（请参见允许缓存），推荐也使用它作为会话：

```
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
```

参见:

配置会话 (session) 引擎, SESSION_ENGINE

DATABASES

连接到数据库服务器，请查阅 Django 的文档了解详情。

参见:

Weblate 的数据库设置, DATABASES, 数据库

DEBUG

对于任何生产服务器禁止这项。允许调试模式时，Django 会在出错的情况下向用户显示回溯信息，当禁止时，错误将每封电子邮件发送到 ADMINS （请参见上面）。

调试模式还使 Weblate 变慢，因为在这种情况下 Django 内部存储了非常多的信息。

参见:

DEBUG, 禁止调试模式

DEFAULT_FROM_EMAIL

用于发送电子邮件的电子邮件发件人地址，例如注册电子邮箱。

参见:

DEFAULT_FROM_EMAIL

SECRET_KEY

Django 使用的密钥，用于在 cookie 中登录一些信息，更多信息请参见*Django* 密钥。

参见:

SECRET_KEY

SERVER_EMAIL

用作向管理员发送电子邮件的发送者地址的电子邮箱，例如通知失败的合并。

参见:

SERVER_EMAIL

2.1.8 填满数据库

在配置准备好之后，可以运行 `weblate migrate` 来建立数据库结构。现在你将能够使用管理界面建立翻译项目。

在想要非交互式地运行安装的情况下，可以使用 `weblate migrate --noinput`，然后使用 *createadmin* 命令来建立管理用户。

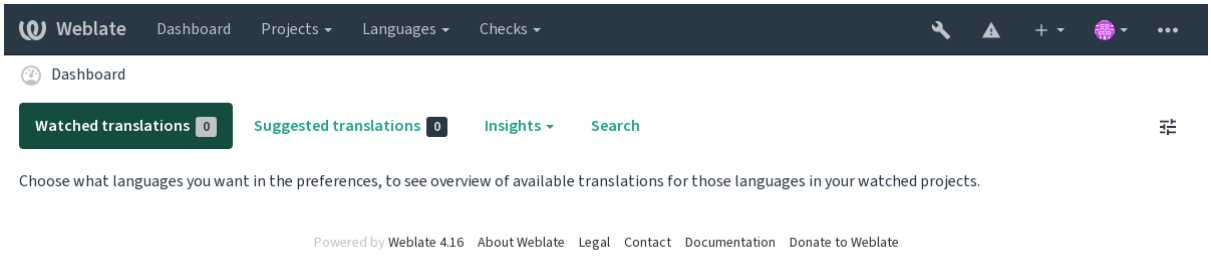
完成后，你还应该在管理界面查看 *Performance report*，它会提示你站点上可能存在的非最佳配置。

参见：

配置, 权限和内置角色列表

2.1.9 生产设置

对于生产设置，你应该进行以下部分中所述的调整。最关键的设置将触发警告：如果以超级用户身份登录，则会在顶部栏中显示一个叹号：



同样也推荐查看由 Django 触发的检查（尽管可能不需要修复所有的检查）：

```
weblate check --deploy
```

您也可以从管理界面 查看完全一样的检查清单。

参见：

部署清单

禁止调试模式

禁止 Django 的调试模式（*DEBUG*）：

```
DEBUG = False
```

在调试模式打开时，Django 存储所有执行的查询，并将错误的回溯显示给用户，这在生产设置中是不需要的。

参见：

调整配置

是当地配置管理设置

将正确的管理地址设置到*ADMINS*设置中，来确定服务器出现一些故障时谁接收电子邮件，例如：

```
ADMINS = (("Your Name", "your_email@example.com"),)
```

参见：

调整配置

设置正确的网站域名

在管理界面调整网站名称和域名，否则 RSS 中的链接或注册电子邮箱地址将不工作。这使用*SITE_DOMAIN*来配置，它应该包含网站域名。

在 4.2 版本发生变更: 在 4.2 版本之前，替代使用了 Django 网站框架，请参见 “站点”框架。

参见：

允许主机设置, 正确配置 *HTTPS* *SITE_DOMAIN*, *WEBLATE_SITE_DOMAIN*, *ENABLE_HTTPS*

正确配置 HTTPS

强烈推荐使用加密的 HTTPS 协议运行 Weblate。将其允许后，可以在设置中设置*ENABLE_HTTPS*：

```
ENABLE_HTTPS = True
```

提示： 你还会想要新建 HSTS，更多细节请参见 SSL/HTTPS。

参见：

ENABLE_HTTPS, 允许主机设置, 设置正确的网站域名

适当设置 SECURE_HSTS_SECONDS

如果你的网站基于 SSL 上提供服务，那么必须考虑 `settings.py` 中 `SECURE_HSTS_SECONDS` 的设置值，来允许 HTTP Strict Transport Security （HTTP 脚本传输安全）。默认设置为 0，如下所示。

```
SECURE_HSTS_SECONDS = 0
```

如果设置为非 0 整数，那么在所有还不曾具有它的响应时，`django.middleware.security.SecurityMiddleware` 设置 HTTP 严格传输安全 标头。

警告： 不正确的设置这项会导致你的网站不可逆地（在一段时间内）崩溃。请首先阅读 HTTP 严格传输安全 文档。

使用强力的数据库引擎

- 请使用 PostgreSQL 作为生产环境，更多信息请参见*Weblate 的数据库设置*。
- 请在邻近的位置部署数据库服务器，否则网络性能或可靠性问题可能会破坏你的 Weblate 体验。
- 检查数据库服务器性能或调整其配置，例如使用 PGTune.。

参见:

Weblate 的数据库设置, *从其它数据库迁移到 PostgreSQL*, *调整配置*, 数据库

允许缓存

如果可能，通过调整 CACHES 配置变量来使用来自 Django 的 Redis，例如：

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/0",
        # If redis is running on same host as Weblate, you might
        # want to use unix sockets instead:
        # 'LOCATION': 'unix:///var/run/redis/redis.sock?db=0',
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "PARSER_CLASS": "redis.connection.HiredisParser",
        },
    }
}
```

提示: 在为缓存更改 Redis 设置的情况下，也会需要为 Celery 来调整，请参见*使用 Celery 的后台任务*。

参见:

头像缓存, Django 缓存框架

头像缓存

除了 Django 的缓存，Weblate 还执行头像缓存。推荐使用单独的、文件后端缓存来用于这个目的：

```
CACHES = {
    "default": {
        # Default caching backend setup, see above
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "unix:///var/run/redis/redis.sock?db=0",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "PARSER_CLASS": "redis.connection.HiredisParser",
        },
    },
    "avatar": {
        "BACKEND": "django.core.cache.backends.filebased.FileBasedCache",
        "LOCATION": os.path.join(DATA_DIR, "avatar-cache"),
        "TIMEOUT": 604800,
        "OPTIONS": {
            "MAX_ENTRIES": 1000,
        },
    },
}
```

参见:

ENABLE_AVATARS, *AVATAR_URL_PREFIX*, 头像, 允许缓存, Django 缓存框架

配置电子邮件发送的设置

Weblate 需要在几种情况下发送电子邮件，这些电子邮件应具有正确的发送者地址，请配置:setting:*SERVER_EMAIL* 和*DEFAULT_FROM_EMAIL*，与你的环境匹配，例如:

```
SERVER_EMAIL = "admin@example.org"
DEFAULT_FROM_EMAIL = "weblate@example.org"
```

备注: 为了禁止 Weblate 发送电子邮件，将 EMAIL_BACKEND 设置为 `django.core.mail.backends.dummy.EmailBackend`。

这将禁止 所有电子邮件的投递，包括注册或密码重置电子邮件。

参见:

调整配置, 配置电子邮件发件箱, EMAIL_BACKEND, DEFAULT_FROM_EMAIL, SERVER_EMAIL

允许主机设置

Django 需要*ALLOWED_HOSTS* 保存你的网站允许服务的域名列表，将其保持空置会屏蔽任何请求。

在没有配置来匹配 HTTP 服务器的情况下，会得到错误信息，如 `Invalid HTTP_HOST header: '1.1.1.1'. You may need to add '1.1.1.1' to ALLOWED_HOSTS.`

提示: 在 Docker 容器上，这可以使用，为 *WEBLATE_ALLOWED_HOSTS*。

参见:

ALLOWED_HOSTS, *WEBLATE_ALLOWED_HOSTS*, 设置正确的网站域名

Django 密钥

SECRET_KEY 设置由 Django 使用来进行 cookies 签名，应该真正产生自己的值，而不是使用来自举例的设置的值。

可以使用与 Weblate 一起上市的 **weblate-generate-secret-key**，来产生新的密钥。

参见:

SECRET_KEY

运行维护任务

为了优化性能，在后台运行一些维护任务是个好方法。现在这由使用 *Celery* 的后台任务 自动进行，并且包括后面的任务:

- 配置健康性的检查（每小时）。
- 提交待处理的更改（每小时），请参见惰性提交 和*commit_pending*。
- 更新部件警报（每天）。
- 更新远程分支（每晚），请参见*AUTO_UPDATE*。
- 翻译记忆库备份到 JSON （每天），请参见*dump_memory*。

- 全文本和数据库维护任务（每天和每周任务），请参见*cleanuptrans*。

在 3.2 版本发生变更: 从 3.2 版本开始，执行这些任务的默认方式是使用 Celery，并且 Weblate 已经具有一些适当的配置，请参见*使用 Celery 的后台任务*。

系统的地区与编码

系统的地区应该设置为兼容 UTF-8 的。在多数 Linux 发行版中这是默认设置。在你的系统不能兼容的情况下，请将地区更改为 UTF-8 变体。

例如通过编辑 `/etc/default/locale` 并设置 `LANG="C.UTF-8"`。

某些情况下，各个服务具有不同的区域设置配置。这在不同发行版和 web 服务器之间是不同的，所以请查阅你的 web 服务器包的文档。

Ubuntu 系统下的 Apache 使用 `/etc/apache2/envvars`:

```
export LANG='en_US.UTF-8'
export LC_ALL='en_US.UTF-8'
```

CentOS 系统下的 Apache 使用 `/etc/sysconfig/httpd`` (或 :file:/opt/rh/httpd24/root/etc/sysconfig/httpd‘)
:

```
LANG='en_US.UTF-8'
```

使用定制的证书授权

Weblate 在 HTTP 请求时验证 SSL 证书。在使用定制的证书授权的情况下，这样定制的证书授权在默认 bundles 的中不被信任，你必须将其证书添加为可信任。

首选方法是在系统层面上进行，请查看你的发行版文档了解详情（例如在 debian 中，这可以通过将 CA 证书放置在 `/usr/local/share/ca-certificates/`，并运行 **update-ca-certificates** 来完成）。

一旦完成，系统工具就会信任证书，这包括 Git。

对于 Python 代码，需要配置请求来使用系统 CA bundle，而不是与它一起上市的那个。这可以通过将后面的模板放到 `settings.py` 来实现（路径是 Debian 特有的）:

```
import os

os.environ["REQUESTS_CA_BUNDLE"] = "/etc/ssl/certs/ca-certificates.crt"
```

压缩客户端资源

Weblate 带有一组 JavaScript 和 CSS 文件。由于性能的原因，在将其发送到客户端前最好进行压缩。在默认配置中，这通过耗费一点经常资源而在运行中完成。在大型安装中，推荐允许离线压缩模式。这需要在配置中完成，并且必须在每次 weblate 升级时触发压缩。

配置切换很简单，通过允许 `django.conf.settings.COMPRESS_OFFLINE`，并配置 `django.conf.settings.COMPRESS_OFFLINE_CONTEXT` （后者已经包括在示例的配置中）:

```
COMPRESS_OFFLINE = True
```

在每个部署中，您需要压缩文件来匹配当前的版本:

```
weblate compress
```

提示: 官方 Docker 镜像已经允许了这个特性。

参见:

Common Deployment Scenarios, 为静态文件提供服务

2.1.10 运行服务器

提示: 如果你对下面描述的服务没有经验，你可以尝试使用 *Docker* 安装。

需要几个服务来运行 Weblate，推荐的设置包括:

- 数据库服务器（请参见*Weblate* 的数据库设置 ）
- 缓存服务器（请参见允许缓存 ）
- 用于静态文件和终结 SSL 的前端 web 服务器（请参见为静态文件提供服务 ）
- 用于动态内容的 WSGI 服务器（请参见*NGINX* 和 *uWSGI* 的配置示例 ）
- 用于执行后台任务的 Celery （请参见使用 *Celery* 的后台任务 ）

备注: 这些服务之间由一些依赖项，例如当启动 Celery 或 uwsgi 进程时，缓存和数据库应该运行。

在多数情况下，需要在单一（虚拟）服务器上运行所有服务，但在您的安装是重载的情况下，可以将这些服务拆开。对此的唯一限制是 Celery 和 Wsgi 服务器需要访问*DATA_DIR*。

备注: WSGI 进程和 Celery 进程必须在同一用户下被执行，否则*DATA_DIR* 中的文件将以混合的所有权来存储，导致运行问题。

还请参见文件系统权限 和使用 *Celery* 的后台任务。

运行 web 服务器

运行 Weblate 与运行其他任何基于 Django 的程序没什么不同。Django 通常作为 uWSGI 或 fcgi 执行（请参见下面不同 web 服务器的示例）。

为了检测的目的，您可以在 Django 中使用内建的 web 服务器:

```
weblate runserver
```

警告: 不要使用在生产设置中这个服务器。它还没有通过安全审查或性能检测。请参见 Django 文档中的 `runserver`。

提示: Django 内建服务只通过允许*DEBUG* 来为静态文件提供服务，因为它只用于开发的目的。对于生产使用，请参见*NGINX* 和 *uWSGI* 的配置示例 、*Apache* 的配置示例、*Apache* 和 *Gunicorn* 的配置示例 和为静态文件提供服务 中的 wsgi 设置。

为静态文件提供服务

在 2.4 版本发生变更: 在 2.4 版本之前，Weblate 没有正确使用 Django 静态文件框架，相关设置也更为复杂。

Django 需要将其静态文件收集在一个单一文件夹中。为此，执行 `weblate collectstatic --noinput`。这会将静态文件复制到 `STATIC_ROOT` 设置指定的文件夹中（这默认为*DATA_DIR* 内的 `static` 文件夹）。

推荐直接从你的 web 服务器为静态文件提供服务，对于后面的路径应该使用它：

/static/
: 为 Weblate 的静态文件和管理界面（由 `STATIC_ROOT` 定义）提供服务。

/media/
: 用于上传用户媒体（例如截屏）。

/favicon.ico
: 应该重写，重写规则为 `/static/favicon.ico` 提供服务。

参见:

NGINX 和 uWSGI 的配置示例, *Apache 的配置示例*, *Apache 和 Gunicorn 的配置示例*, 压缩客户端资源, 如何部署 Django, 如何部署静态文件

内容安全政策

默认的 Weblate 配置允许 `weblate.middleware.SecurityMiddleware` 中间件，它设置与 HTTP 标头相关的安全，如 `Content-Security-Policy` 或 `X-XSS-Protection`。这些被默认新建，与 Weblate 及其配置一起工作，但这对你的环境需要定制化。

参见:

CSP_SCRIPT_SRC, *CSP_IMG_SRC*, *CSP_CONNECT_SRC*, *CSP_STYLE_SRC*, *CSP_FONT_SRC*

NGINX 和 uWSGI 的配置示例

为了运行生产 web 服务器，使用与 Weblate 一起安装的 wsgi 封装（在虚拟 env 的情况下，它安装为 `~/weblate-env/lib/python3.9/site-packages/weblate/wsgi.py`）。别忘了将 Python 的搜索路径同样设置为您的虚拟 env（例如在 uWSGI 中使用 `virtualenv = /home/user/weblate-env`）。

后面的配置将 Weblate 作为 NGINX web 服务器下的 uWSGI 来运行。

NGINX 的配置（还作为 `weblate/examples/weblate.nginx.conf` 来获得）：

```
#
# nginx configuration for Weblate
#
# You will want to change:
#
# - server_name
# - change /home/weblate/weblate-env to location where Weblate virtualenv is placed
# - change /home/weblate/data to match your DATA_DIR
# - change python3.9 to match your Python version
# - change weblate user to match your Weblate user
#
server {
    listen 80;
    server_name weblate;
    # Not used
    root /var/www/html;
```

(续下页)

(接上页)

```
    location ~ ^/favicon.ico$ {
        # DATA_DIR/static/favicon.ico
        alias /home/weblate/data/static/favicon.ico;
        expires 30d;
    }

    location /static/ {
        # DATA_DIR/static/
        alias /home/weblate/data/static/;
        expires 30d;
    }

    location /media/ {
        # DATA_DIR/media/
        alias /home/weblate/data/media/;
        expires 30d;
    }

    location / {
        include uwsgi_params;
        # Needed for long running operations in admin interface
        uwsgi_read_timeout 3600;
        # Adjust based to uwsgi configuration:
        uwsgi_pass unix:///run/uwsgi/app/weblate/socket;
        # uwsgi_pass 127.0.0.1:8080;
    }
}
```

uWSGI 的配置（还作为 `weblate/examples/weblate.uwsgi.ini` 来获得）：

```
#
# uWSGI configuration for Weblate
#
# You will want to change:
#
# - change /home/weblate/weblate-env to location where Weblate virtualenv is placed
# - change /home/weblate/data to match your DATA_DIR
# - change python3.9 to match your Python version
# - change weblate user to match your Weblate user
#
[uwsgi]
plugins       = python3
master        = true
protocol      = uwsgi
socket        = 127.0.0.1:8080
wsgi-file     = /home/weblate/weblate-env/lib/python3.9/site-packages/weblate/wsgi.
↪py

# Add path to Weblate checkout if you did not install
# Weblate by pip
# python-path   = /path/to/weblate

# In case you're using virtualenv uncomment this:
virtualenv = /home/weblate/weblate-env

# Needed for OAuth/OpenID
buffer-size   = 8192

# Reload when consuming too much of memory
reload-on-rss = 250
```

(续下页)

(接上页)

```
# Increase number of workers for heavily loaded sites
workers       = 8

# Enable threads for Sentry error submission
enable-threads = true

# Child processes do not need file descriptors
close-on-exec = true

# Avoid default 0000 umask
umask = 0022

# Run as weblate user
uid = weblate
gid = weblate

# Enable harakiri mode (kill requests after some time)
# harakiri = 3600
# harakiri-verbose = true

# Enable uWSGI stats server
# stats = :1717
# stats-http = true

# Do not log some errors caused by client disconnects
ignore-sigpipe = true
ignore-write-errors = true
disable-write-exception = true
```

参见:

如何用 uWSGI 托管 Django

Apache 的配置示例

推荐当 Weblate 使用 WSGI 时使用 prefork MPM。

后面的配置将 Weblate 作为 WSGI 来运行，您需要允许“mod_wsgi“（作为 `weblate/examples/apache.conf` 来获得）：

```
#
# VirtualHost for Weblate
#
# You will want to change:
#
# - ServerAdmin and ServerName
# - change /home/weblate/weblate-env to location where Weblate virtualenv is placed
# - change /home/weblate/data to match your DATA_DIR
# - change python3.9 to match your Python version
# - change weblate user to match your Weblate user
#
<VirtualHost *:80>
    ServerAdmin admin@weblate.example.org
    ServerName weblate.example.org

    # DATA_DIR/static/favicon.ico
    Alias /favicon.ico /home/weblate/data/static/favicon.ico

    # DATA_DIR/static/
    Alias /static/ /home/weblate/data/static/
```

(续下页)

(接上页)

```
    <Directory /home/weblate/data/static/>
        Require all granted
    </Directory>

    # DATA_DIR/media/
    Alias /media/ /home/weblate/data/media/
    <Directory /home/weblate/data/media/>
        Require all granted
    </Directory>

    # Path to your Weblate virtualenv
    WSGIDaemonProcess weblate python-home=/home/weblate/weblate-env user=weblate␣
↪request-timeout=600
    WSGIProcessGroup weblate
    WSGIApplicationGroup %{GLOBAL}

    WSGIScriptAlias / /home/weblate/weblate-env/lib/python3.9/site-packages/
↪weblate/wsgi.py process-group=weblate
    WSGIPassAuthorization On

    <Directory /home/weblate/weblate-env/lib/python3.9/site-packages/weblate/>
        <Files wsgi.py>
        Require all granted
        </Files>
    </Directory>

</VirtualHost>
```

备注： Weblate 需要 Python 3，所以请确认您运行 modwsgi 的 Python 3 变体。它通常作为独立的包来获得，例如 `libapache2-mod-wsgi-py3`。

参见：

系统的地区与编码, 如何使用 Apache 和 mod_wsgi 托管 Django

Apache 和 Gunicorn 的配置示例

后面的配置在 Gunicorn 和 Apache 2.4 中运行 Weblate（作为 `weblate/examples/apache.gunicorn.conf` 获得）：

```
#
# VirtualHost for Weblate using gunicorn on localhost:8000
#
# You will want to change:
#
# - ServerAdmin and ServerName
# - change /home/weblate/weblate-env to location where Weblate virtualenv is placed
# - change /home/weblate/data to match your DATA_DIR
# - change python3.9 to match your Python version
# - change weblate user to match your Weblate user
#
<VirtualHost *:443>
    ServerAdmin admin@weblate.example.org
    ServerName weblate.example.org

    # DATA_DIR/static/favicon.ico
    Alias /favicon.ico /home/weblate/data/static/favicon.ico
```

(续下页)

(接上页)

```
    # DATA_DIR/static/
    Alias /static/ /home/weblate/data/static/
    <Directory /home/weblate/data/static/>
        Require all granted
    </Directory>

    # DATA_DIR/media/
    Alias /media/ /home/weblate/data/media/
    <Directory /home/weblate/data/media/>
        Require all granted
    </Directory>

    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/https_cert.cert
    SSLCertificateKeyFile /etc/apache2/ssl/https_key.pem
    SSLProxyEngine On

    ProxyPass /favicon.ico !
    ProxyPass /static/ !
    ProxyPass /media/ !

    ProxyPass / http://localhost:8000/
    ProxyPassReverse / http://localhost:8000/
    ProxyPreserveHost On
</VirtualHost>
```

参见:

如何使用 Gunicorn 托管 Django

在路径下运行 Weblate

在 1.3 版本加入.

推荐当 Weblate 使用 WSGI 时使用 prefork MPM。

为 “/weblate“ 下的 Weblate 提供服务的 Apache 配置的示例。再次使用 `mod_wsgi` （还作为 `weblate/examples/apache-path.conf` 获得）:

```
#
# VirtualHost for Weblate, running under /weblate path
#
# You will want to change:
#
# - ServerAdmin and ServerName
# - change /home/weblate/weblate-env to location where Weblate virtualenv is placed
# - change /home/weblate/data to match your DATA_DIR
# - change python3.9 to match your Python version
# - change weblate user to match your Weblate user
#
<VirtualHost *:80>
    ServerAdmin admin@weblate.example.org
    ServerName weblate.example.org

    # DATA_DIR/static/favicon.ico
    Alias /weblate/favicon.ico /home/weblate/data/static/favicon.ico

    # DATA_DIR/static/
    Alias /weblate/static/ /home/weblate/data/static/
    <Directory /home/weblate/data/static/>
        Require all granted
```

(续下页)

(接上页)

```
    </Directory>

    # DATA_DIR/media/
    Alias /weblate/media/ /home/weblate/data/media/
    <Directory /home/weblate/data/media/>
        Require all granted
    </Directory>

    # Path to your Weblate virtualenv
    WSGIDaemonProcess weblate python-home=/home/weblate/weblate-env user=weblate␣
↪request-timeout=600
    WSGIProcessGroup weblate
    WSGIApplicationGroup %{GLOBAL}

    WSGIScriptAlias /weblate /home/weblate/weblate-env/lib/python3.9/site-packages/
↪weblate/wsgi.py process-group=weblate
    WSGIPassAuthorization On

    <Directory /home/weblate/weblate-env/lib/python3.9/site-packages/weblate/>
        <Files wsgi.py>
        Require all granted
        </Files>
    </Directory>

</VirtualHost>
```

此外，您必须调整 `weblate/settings.py`：

```
URL_PREFIX = "/weblate"
```

2.1.11 使用 Celery 的后台任务

在 3.2 版本加入.

Weblate 使用 Celery 来执行常规和后台任务。你应该运行一个 Celery 服务来执行这些任务。例如，它负责处理以下操作（此列表并不完整）：

- 接收来自外部服务的 webhooks（请见通知钩子）。
- 运行定期的维护任务，如备份、清理、日常附加组件或更新（请见备份和移动 *Weblate*, *BACKGROUND_TASKS*, 附加组件）。
- 正在运行自动翻译.
- 发送摘要通知。
- 从 wsgi 进程中卸载高负载操作。
- 提交待处理更改（参见惰性提交）。

使用 Redis 作为后端的典型安装看上去是这样的：

```
CELERY_TASK_ALWAYS_EAGER = False
CELERY_BROKER_URL = "redis://localhost:6379"
CELERY_RESULT_BACKEND = CELERY_BROKER_URL
```

参见：

Redis broker configuration in Celery

您应该启动 Celery worker 来处理任务，并且定时任务，这可以直接在命令行完成（调试和开发时最有用）：

```
./weblate/examples/celery start
./weblate/examples/celery stop
```

备注： Celery 进程和 WSGI 进程必须在同一用户下被执行，否则*DATA_DIR* 中的文件将以混合的所有权来存储，导致运行问题。

还请参见文件系统权限 和运行服务器。

在 wsgi 中使用 eager 模式执行 Celery 任务

备注： 这将对 web 界面的性能产生严重影响，并会破坏依赖于常规触发器的功能（例如提交挂起的更改、摘要通知或备份）。

对于开发，你可能希望使用 eager configuration，它可以适当地处理所有任务：

```
CELERY_TASK_ALWAYS_EAGER = True
CELERY_BROKER_URL = "memory://"
CELERY_TASK_EAGER_PROPAGATES = True
```

运行 Celery 作为系统服务

您更可能想要运行 Celery 作为守护进程，这由 Daemonization 来涵盖。对于使用 systemd 的最通常的 Linux 设置，您可以使用与下面列出的 `examples` 文件夹一起上市的示例文件。

Systemd 单元作为 `/etc/systemd/system/celery-weblate.service` 放置：

```
[Unit]
Description=Celery Service (Weblate)
After=network.target

[Service]
Type=forking
User=weblate
Group=weblate
EnvironmentFile=/etc/default/celery-weblate
WorkingDirectory=/home/weblate
RuntimeDirectory=celery
RuntimeDirectoryPreserve=restart
LogsDirectory=celery
ExecStart=/bin/sh -c '${CELERY_BIN} multi start ${CELERYD_NODES} \
  -A ${CELERY_APP} --pidfile=${CELERYD_PID_FILE} \
  --logfile=${CELERYD_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} ${CELERYD_OPTS}'
ExecStop=/bin/sh -c '${CELERY_BIN} multi stopwait ${CELERYD_NODES} \
  --pidfile=${CELERYD_PID_FILE}'
ExecReload=/bin/sh -c '${CELERY_BIN} multi restart ${CELERYD_NODES} \
  -A ${CELERY_APP} --pidfile=${CELERYD_PID_FILE} \
  --logfile=${CELERYD_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} ${CELERYD_OPTS}'

[Install]
WantedBy=multi-user.target
```

环境配置作为 `/etc/default/celery-weblate` 放置：

```
# Name of nodes to start
CELERYD_NODES="celery notify memory backup translate"
```

(续下页)

(接上页)

```
# Absolute or relative path to the 'celery' command:
CELERY_BIN="/home/weblate/weblate-env/bin/celery"

# App instance to use
# comment out this line if you don't use an app
CELERY_APP="weblate.utils"

# Extra command-line arguments to the worker,
# increase concurrency if you get weblate.E019
CELERYD_OPTS="--beat:celery --queues:celery=celery --prefetch-multiplier:celery=4 \
    --queues:notify=notify --prefetch-multiplier:notify=10 \
    --queues:memory=memory --prefetch-multiplier:memory=10 \
    --queues:translate=translate --prefetch-multiplier:translate=4 \
    --concurrency:backup=1 --queues:backup=backup  --prefetch-multiplier:backup=2"

# Logging configuration
# - %n will be replaced with the first part of the nodename.
# - %I will be replaced with the current child process index
#   and is important when using the prefork pool to avoid race conditions.
CELERYD_PID_FILE="/run/celery/weblate-%n.pid"
CELERYD_LOG_FILE="/var/log/celery/weblate-%n%I.log"
CELERYD_LOG_LEVEL="INFO"
```

使用 **logrotate** 来旋转 Celery 记录的额外配置将被放置为 `/etc/logrotate.d/celery`:

```
/var/log/celery/*.log {
        weekly
        missingok
        rotate 12
        compress
        notifempty
}
```

使用 Celery beat 的周期性任务

Weblate 带有内建的定时任务设置。然而您可以在 `settings.py` 中定义另外的任务，例如请参见惰性提交。

任务应该由 Celery beats 守护进程执行。在不能正常工作的情况下，它可能不会运行，或者其数据库崩溃。在这样的情况下检查 Celery 启动日志，来找出根本原因。

监测 Celery 状态

你可以在管理界面 中找到 Celery 任务队列的当前长度，或者你可以使用在命令中使用*celery_queues*。如果队列太长，你也会在管理界面中得到配置错误。

> **警告：** Celery 错误默认之存储在 Celery 日志中，并且用户不可见。在您想要了解故障概况的情况下，推荐配置收集错误报告。

参见：

监测 *Weblate*, 如何检查 *Weblate* 是否被正确地设置？, Configuration and defaults, Workers Guide, Daemonization, Monitoring and Management Guide, *celery_queues*

2.1.12 监测 Weblate

Weblate 提供了 `/healthz/` URL，可用于简单的健康检查，例如使用 Kubernetes。Docker 容器内置了使用此 URL 的健康检查。

为了检测 Weblate 的各项指标，你可以使用*`GET /api/metrics/`* API 端点。

参见：

如何检查 Weblate 是否被正确地设置？, *监测 Celery 状态*, Weblate 的 Munin 插件

2.1.13 收集错误报告

与其他任何软件一样，Weblate 可能会失败。为了收集有用的故障状态，我们推荐使用第三方服务来收集此类信息。这在 Celery 任务失败的情况下尤其有用，否则将只会向日志报告错误，而您不会收到有关它们的通知。Weblate 支持以下服务：

Sentry

Weblate 内置了对 Sentry 的支持。要使用它，只需在 `settings.py` 中设置*`SENTRY_DSN`*：

```
SENTRY_DSN = "https://id@your.sentry.example.com/"
```

Rollbar

Weblate 具有对 Rollbar 的内置支持。要使用它，只需遵循 Rollbar notifier for Python 的说明即可。

简而言之，您需要调整 `settings.py`：

```
# Add rollbar as last middleware:
MIDDLEWARE = [
    # … other middleware classes …
    "rollbar.contrib.django.middleware.RollbarNotifierMiddleware",
]

# Configure client access
ROLLBAR = {
    "access_token": "POST_SERVER_ITEM_ACCESS_TOKEN",
    "client_token": "POST_CLIENT_ITEM_ACCESS_TOKEN",
    "environment": "development" if DEBUG else "production",
    "branch": "main",
    "root": "/absolute/path/to/code/root",
}
```

其他所有内容都是自动集成的，您现在将同时收集服务器和客户端错误。

2.1.14 将 Weblate 迁移到其他服务其中

将 Weblate 迁移到其他服务器应该非常简单，然而它将数据存储在几个位置，您应该小心迁移。最佳的方式时停止 Weblate 再迁移。

迁移数据库

根据你的数据库后端，你可能有几个选项来迁移数据库。最直接的方法是使用数据库原生工具，因它们通常最有效（如 **mysqldump** 或 **pg_dump**）。如果你的数据库支持的话，你也可以使用复制。

参见：

在数据库之间进行迁移的内容，见*从其它数据库迁移到 PostgreSQL*。

迁移版本控制系统（VCS）仓库

存储在*DATA_DIR* 下的版本控制系统（VCS）同样需要迁移。您可以简单地复制它们，或使用 **rsync** 来更有效地迁移。

其他注意事项

不要忘记移动 Weblate 会使用的其他服务，如 Redis、Cron 任务或自定义的身份验证后端。

2.2 Weblate 部署

Weblate 可以容易地安装到你的云中。请针对你的平台找到具体的指南：

- *使用 Docker 安装*
- *在 OpenShift 上安装*
- *在 Kubernetes 上安装*

2.2.1 用于 Weblate 的第三方部署

备注： 以下部署不是由 Weblate 团队开发或支持的。部分设置可能与本文档中的描述有所不同。

Bitnami Weblate 栈

Bitnami 为许多平台提供 Weblate 一键安装包，详情请访问 <https://bitnami.com/stack/weblate>。

参见：

Bitnami 打包的 Weblate

Weblate Cloudron 包

Cloudron 是自托管 web 应用的平台。安装有 Cloudron 的 Weblate 会自动更新。软件包由 Cloudron 团队在它们的 Weblate package repo 上维护。

YunoHost 中的 Weblate

自托管项目 YunoHost 为 Weblate 提供了包。一旦安装了 YunoHost，就可以同其它应用一样安装 Weblate。它还为你提供带有备份和恢复的完全工作栈，但你必须为特定应用编辑设置文件。

可以使用管理界面，或这个按钮（它将带你到你的服务器）：

还能够使用命令行界面：

```
yunohost app install https://github.com/YunoHost-Apps/weblate_ynh
```

2.3 升级 Weblate

2.3.1 Docker 镜像升级

官方 Docker 镜像（请参见使用 *Docker* 安装）已集成了所有 Weblate 升级步骤。除了拉取最新的版本外通常无需进行手动操作。

参见：

升级 Docker 容器

2.3.2 一般的升级指示

在升级前，请检查当前的软件要求，因为他们可能被更改。一旦所有的要求被安装或升级，请调整你的 `settings.py`，来匹配配置中的更改（正确的值请咨询 `settings_example.py`）。

升级前，请务必查阅与特定版本相关的指示。如果您要跳过某些版本，请按照升级中要跳过的所有版本的说明进行操作。有时最好先升级到某个中间版本，以确保顺利迁移。跨越多个版本的升级应该是可行的，但不像单一版本的升级那样经过良好的测试。

备注： 推荐在升级前执行全数据库备份，使你可以在升级失败的情况下回滚数据库，请参见备份和移动 *Weblate*。

---

1. 停止 WSGI 和 Celery 进程。升级可能执行数据库的不兼容更改，因此在升级中避免旧的进程运行总是安全的。
2. 升级 Weblate 代码。

   对于 pip 安装，可以通过以下命令实现：

   ```
   pip install -U "Weblate[all]==version"
   ```

   或者，如果您只想获取最新发布的版本：

   ```
   pip install -U "Weblate[all]"
   ```

   如果不想安装所有可选依赖项，请执行以下操作：

   ```
   pip install -U Weblate
   ```

   通过 Git 核实，你需要取回新的源代码并升级你的安装：

```
cd weblate-src
git pull
# Update Weblate inside your virtualenv
. ~/weblate-env/bin/pip install -e .
# Install dependencies directly when not using virtualenv
pip install --upgrade -r requirements.txt
# Install optional dependencies directly when not using virtualenv
pip install --upgrade -r requirements-optional.txt
```

3. 新版本 Weblate 可能有新的可选依赖项，快来看看有没有你感兴趣的功能。
4. 升级配置文件，所需的步骤请参考 `settings_example.py` 或与特定版本相关的指示。
5. 升级数据库架构：

```
weblate migrate --noinput
```

6. 收集升级的静态文件（请参见运行服务器 和为静态文件提供服务）：

```
weblate collectstatic --noinput --clear
```

7. 压缩 JavaScript 和 CSS 文件（可选步骤，请参见压缩客户端资源 ）：

```
weblate compress
```

8. 如果你运行来自 Git 的版本，每次升级时还应该重新生成 locale 文件。可以通过调用后面的来进行：

```
weblate compilemessages
```

9. 验证您的设置合理（还请参见生产设置）：

```
weblate check --deploy
```

10. 重新启动 Celery worker（请参见使用 *Celery* 的后台任务）。

2.3.3 与特定版本相关的指示

从 2.x 升级

如果从 2.x 发布版本升级，首先总是升级到 3.0.1，然后继续在 3.x 系列中升级。跳过这步的升级不被支持，并且会中断。

参见：

Weblate 3.0 文档中关于从 2.20 升级到 3.0

从 3.x 升级

如果从 3.x 发布版本升级，首先总是升级到 4.0.4 或 4.1.1，然后继续在 4.x 系列中升级。跳过这步的升级不被支持，并且会中断。

参见：

Weblate 4.0 文档中关于从 3.11 升级到 4.0

从 4.0 升级到 4.1

请按照一般的升级指示 来执行升级。

显著的配置与依赖项更改：

- 在 settings_example.py 中有几项更改，最显著的是中间件的更改，请由此调整你的设置。
- 有几个新的文件格式，在修改*WEBLATE_FORMATS* 的情况下，你会想要将他们包括进来。
- 有几个新的质量检查，在修改*CHECK_LIST* 的情况下，你会想要将他们包括进来。
- 在 DEFAULT_THROTTLE_CLASSES 设置中有几项更改，来允许在 API 中报告速率限制。
- 有几个新的且更新的要求。
- 在 INSTALLED_APPS 中有一些更改。
- MT_DEEPL_API_VERSION 设置已在 V4.7 中移除，*DeepL* 机器翻译现在使用新的 MT_DEEPL_API_URL 代替，您可能需要调整 MT_DEEPL_API_URL 以匹配您的订阅。

参见：

一般的升级指示

从 4.1 升级到 4.2

请按照一般的升级指示 来执行升级。

显著的配置与依赖项更改：

- 从 3.x 发布版本升级不再支持，请首先升级到 4.0 或 4.1。
- 有几个新的且更新的要求。
- 在 settings_example.py 中有几项更改，最显著的是新中间件和更改的应用订购。
- 基于 JSON 格式的密钥是不再包括前导的点。在数据库迁移过程中调整字符串，但在你依赖于导出或 API 中的密钥时，外部部件会需要调整。
- Celery 配置更改，不再使用 memory 队列。请调整你的启动脚本和 CELERY_TASK_ROUTES 设置。
- 现在在设置中配置 Weblate 域，请参见 SITE_DOMAIN`（或:envvar:`WEBLATE_SITE_DOMAIN）。在运行 Weblate 前你将不得不配置它。
- 用户数据库上的用户名和电子邮件字段现在应该不因为大小写敏感而不同。它之前错误地没有被 PostgreSQL 强制。

参见：

一般的升级指示

从 4.2 升级到 4.3

请按照一般的升级指示 来执行升级。

显著的配置与依赖项更改：

- 在质量检查中有一些更改，在你调整*CHECK_LIST* 的情况下会想将他们包括进来。
- 源语言属性从项目移动到 API 中暴露的部件。在使用时你会需要更新*Weblate 客户端*。
- 数据库迁移到 4.3 可能需要很长时间，取决于要翻译的字符串数（预计每 10 万条字符串大约需要 1 小时的迁移时间）。
- 在 INSTALLED_APPS 中有一些更改。
- 有个新的设置*SESSION_COOKIE_AGE_AUTHENTICATED*，补充了 SESSION_COOKIE_AGE。

- 如果你使用 **hub** 或 **lab** 与 GitHub 或 GitLab 集成，则需要重新配置，请参见*GITHUB_CREDENTIALS* 和*GITLAB_CREDENTIALS*。

在 4.3.1 版本发生变更:

- Celery 配置更改，加入了 `memory` 队列。请调整你的启动脚本和 `CELERY_TASK_ROUTES` 设置。

在 4.3.2 版本发生变更:

- 附加组件的 `post_update` 方法现在接受额外的 `skip_push` 参数。

参见:

一般的升级指示

从 4.3 升级到 4.4

请按照一般的升级指示 来执行升级。

显著的配置与依赖项更改:

- 在 `INSTALLED_APPS` 中有一处更改，必须将 `weblate.configuration` 添加在那里。
- 现在需要 Django 3.1。
- 在使用 MySQL 或 MariaDB 的情况下，需要的最低版本提高了，请参见*MySQL 和 MariaDB*。

在 4.4.1 版本发生变更:

- *单语 gettext* 现在同时使用 `msgid` 和“msgctxt“（若存在）。这将改变此类文件中翻译字符串的标识，破坏到与 Weblate 扩展数据（如截图和审校状态）的链接。请确保在升级之前提交此类文件的待处理更改，建议使用*loadpo* 强制加载受影响的部件。
- 增加了 translate-toolkit 的最低要求版本，以解决几个文件格式问题。

参见:

一般的升级指示

从 4.4 升级到 4.5

请按照一般的升级指示 来执行升级。

显著的配置与依赖项更改:

- 如果你的的术语表过大，迁移可能需要相当长的时间。
- 术语表现在存储为常规部件。
- 术语表 API 已移除，请使用常规翻译 API 访问术语表。
- `INSTALLED_APPS` 中有一处更改，应添加 `weblate.metrics`。

在 4.5.1 版本发生变更:

- *pyahocorasick* 模块有一个新的依赖项。

参见:

一般的升级指示

从 4.5 升级到了 4.6

请按照一般的升级指示 来执行升级。

显著的配置与依赖项更改：

- 有几个新的文件格式，在修改*WEBLATE_FORMATS* 的情况下，你会想要将他们包括进来。
- 创建部件的 API 现在自动使用*Weblate 内部网址*，参考*POST /api/projects/(string:project)/components/* 。
- 在依赖关系和 PASSWORD_HASHERS 中，有一个变化，即在密码散列中优先使用 Argon2。

参见：

一般的升级指示

从 4.6 升级到了 4.7

请按照一般的升级指示 来执行升级。

显著的配置与依赖项更改：

- 在 settings_example.py 中有几项更改，最显著的是中间件的更改 (MIDDLEWARE)，请由此调整你的设置。
- mt-deepl‘机器翻译现在有一个通用的 “MT_DEEPL_API_URL‘ 设置，以更灵活地适应不同的订阅模式。MT_DEEPL_API_VERSION 设置不再使用。
- 现在需要 Django 3.2。

参见：

一般的升级指示

从 4.7 升级到 4.8

请按照一般的升级指示 来执行升级。

此版本中不需要额外的升级步骤。

参见：

一般的升级指示

从 4.8 升级到 4.9

请按照一般的升级指示 来执行升级。

- 存储指标有变化，在较大的站点上升级可能需要很长时间。

参见：

一般的升级指示

从 4.9 升级到 4.10

请按照一般的升级指示 来执行升级。

- 每个项目组有一个变化，在有成千上万个项目的网站上，升级可能需要很长的时间。
- Django 4.0 做了一些不兼容的改动，见 Backwards incompatible changes in 4.0。Weblate 仍然支持 Django 3.2，以防其中有问题。可能影响 Weblate 的最显著的变化：
  - 去掉了对 PostgreSQL 9.6 的支持，Django 4.0 支持 PostgreSQL 10 及以上版本。
  - `CSRF_TRUSTED_ORIGINS` 的格式已经改变。
- Docker 容器现在使用 Django 4.0，有关更改，请参见上文。

参见：

一般的升级指示

从 4.10 升级到 4.11

请按照一般的升级指示 来执行升级。

- Weblate 现在需要 Python 3.7 或更新版本。
- 管理每个项目的访问控制 的实现已经改变，从组名中删除了项目前缀。这影响到了 API 用户。
- Weblate 现在使用 `charset-normalizer` 而不是 `chardet` 模块来检测字符集。
- **4.11.1 更改：** `REST_FRAMEWORK` 设置有一处改动 (删去了 `DEFAULT_AUTHENTICATION_CLASSES` 中的一个后端)。

参见：

一般的升级指示

从 4.11 升级到 4.12

请按照一般的升级指示 来执行升级。

- 无需特殊步骤。

参见：

一般的升级指示

从 4.12 升级到 4.13

请按照一般的升级指示 来执行升级。

- 语言定义 现在会在升级时自动更新，使用 *`UPDATE_LANGUAGES`* 来禁用它。
- 对 *Windows RC 文件*, *HTML 文件*, *IDML 格式*, 和 *文本文件* 文件格式的上下文和位置的处理已经改变. 在大多数情况下，上下文现在被显示为位置。
- 机器翻译服务现在可以使用用户界面进行配置，配置文件中的设置将在数据库迁移期间导入。

参见：

一般的升级指示

从 4.13 升级到 4.14

请按照一般的升级指示 来执行升级。

- Java 格式检查现在匹配 GNU gettext 标记。Weblate 中设置的标记会自动迁移，但第三方脚本需要使用 `java-printf-format` 代替 `java-format` 和 `java-format` 代替 `java-messageformat`.
- *jellyfish* 依赖项已被 *rapidfuzz* 取代。
- **在 4.14.2 版中的变化：** 弃用通过 _TOKEN/_USERNAME 配置而不是 _CREDENTIALS 列表对版本控制系统（VCS）服务 API 密钥进行不安全配置。在 Docker 中，请添加匹配的 _HOST 指令。示例参考 *WEBLATE_GITHUB_HOST* 和*GITHUB_CREDENTIALS*。

参见：

一般的升级指示

从 4.14 升级到 4.15

请按照一般的升级指示 来执行升级。

- Weblate 现在需要 PostgreSQL 中的 `btree_gin` 扩展。如果具有足够的权限，迁移过程中将安装它。请参阅创建 *PostgreSQL* 数据库 进行手动设置。
- 该 Docker 镜像不再默认启用调试模式。如果你想要它，使用 *WEBLATE_DEBUG* 在环境中启用。
- 由于要重新创建一些索引，数据库迁移在较大的实例上可能需要花费数小时。
- **在 4.15.1 版中发生改变：** 更改了 rest 框架设置中 `DEFAULT_PAGINATION_CLASS` 的默认值。

参见：

一般的升级指示

从 4.15 升级到 4.16

请按照一般的升级指示 来执行升级。

- Celery beat 现在在数据库中保存任务计划，需要为此更改 `CELERY_BEAT_SCHEDULER` 和 `INSTALLED_APPS`。
- 不再支持已废弃的 VCS 凭据设置，见从 *4.13 升级到 4.14*。
- 升级 *django-crispy-forms* 需要在 `INSTALLED_APPS` 中进行更改。
- *django-cors-headers* 集成需要在 `INSTALLED_APPS` 和 `MIDDLEWARE` 中做出更改。

参见：

一般的升级指示

2.3.4 从 Python 2 升级到 Python 3

Weblate 不再支持早于 3.6 版本的 Python。在仍然运行在较早版本的情况下，请首先在现有版本上执行到 Python 3 的迁移，并在后面进行升级。请参见 Weblate 3.11.1 文档中的从 Python 2 升级到 Python 3（英文）。

2.3.5 从其它数据库迁移到 PostgreSQL

如果在 PostgreSQL 以外的数据库上运行 Weblate，你应该考虑迁移到 PostgreSQL，因为 Weblate 与它搭配表现最佳。后面的步骤将引导你在数据库之间迁移数据。请记住迁移前要停止 web 和 Celery 服务器，否则会导致不一致的数据。

创建 PostgreSQL 数据库

在另一个单独的数据库中运行 Weblate，并将用户账户分开通常是个好方法：

```
# If PostgreSQL was not installed before, set the main password
sudo -u postgres psql postgres -c "\password postgres"

# Create a database user called "weblate"
sudo -u postgres createuser -D -P weblate

# Create the database "weblate" owned by "weblate"
sudo -u postgres createdb -E UTF8 -O weblate weblate
```

使用 Django JSON 转储来迁移

最简单的迁移方法是使用 Django JSON 转储。这对于较小的安装工作得很好。在更大的网站，你会想要使用 pgloader 代替，请参见*使用 pgloader 迁移到 PostgreSQL*。

1. 将 PostgreSQL 作为附加数据库连接添加到 `settings.py`：

```
DATABASES = {
    "default": {
        # Database engine
        "ENGINE": "django.db.backends.mysql",
        # Database name
        "NAME": "weblate",
        # Database user
        "USER": "weblate",
        # Database password
        "PASSWORD": "password",
        # Set to empty string for localhost
        "HOST": "database.example.com",
        # Set to empty string for default
        "PORT": "",
        # Additional database options
        "OPTIONS": {
            # In case of using an older MySQL server, which has MyISAM as a␣
↪default storage
            # 'init_command': 'SET storage_engine=INNODB',
            # Uncomment for MySQL older than 5.7:
            # 'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
            # If your server supports it, see the Unicode issues above
            "charset": "utf8mb4",
            # Change connection timeout in case you get MySQL gone away error:
            "connect_timeout": 28800,
        },
    },
    "postgresql": {
        # Database engine
        "ENGINE": "django.db.backends.postgresql",
        # Database name
        "NAME": "weblate",
        # Database user
```

(续下页)

(接上页)

```
        "USER": "weblate",
        # Database password
        "PASSWORD": "password",
        # Set to empty string for localhost
        "HOST": "database.example.com",
        # Set to empty string for default
        "PORT": "",
    },
}
```

2. 运行迁移，并将任何插入到表格中的数据 drop 掉：

```
weblate migrate --database=postgresql
weblate sqlflush --database=postgresql | weblate dbshell --database=postgresql
```

3. 将遗留数据库进行转储，并导入 PostgreSQL

```
weblate dumpdata --all --output weblate.json
weblate loaddata weblate.json --database=postgresql
```

4. 调整 DATABASES 而只使用 PostgreSQL 数据库作为默认，将遗留连接删除掉。

现在 Weblate 应该准备好从 PostgreSQL 数据库运行了。

使用 pgloader 迁移到 PostgreSQL

pgloader 是通用迁移工具，将数据迁移到 PostgreSQL。你可以使用它来迁移 Weblate 数据库。

1. 调整 *settings.py* 文件而将 PostgreSQL 用作数据库。
2. 迁移 PostgreSQL 中的模式：

```
weblate migrate
weblate sqlflush | weblate dbshell
```

3. 运行 pgloader 来转移数据。后面的脚本可以用于迁移数据库，但你会想要学习更多关于 pgloader 的知识，来理解它做什么以及调整它来匹配你的设置：

```
LOAD DATABASE
     FROM      mysql://weblate:password@localhost/weblate
     INTO postgresql://weblate:password@localhost/weblate

WITH include no drop, truncate, create no tables, create no indexes, no␣
↪foreign keys, disable triggers, reset sequences, data only

ALTER SCHEMA 'weblate' RENAME TO 'public'
;
```

2.3.6 从 Pootle 迁移

由于 Weblate 最初是作为 Pootle 的替代品而编写的，因此支持从 Pootle 迁移用户账户。你可以从 Pootle 转储用户，并使用 *importusers* 将其导入。

2.4 备份和移动 Weblate

2.4.1 项目级别备份

在 4.14 版本加入.

> **警告：** 仅当使用 PostgreSQL 或 MariaDB 10.5+ 作为数据库时，才支持还原备份。

该项目备份了 Weblate 的所有翻译内容（项目、部件、翻译、字符串注释、建议或检查），它适用于将一个项目转移到另一个 Weblate 实例。

你可以在 管理 ↓ 备份中进行项目备份。在创建项目时可以恢复备份（参见添加翻译项目和部件）。

备份当前不包括访问控制信息和历史记录。

评论和建议由创建它们的用户的用户名备份。导入后，它会分配给匹配的用户。如果没有具有该用户名的用户，则将其分配给匿名用户。

生成的备份按照*PROJECT_BACKUP_KEEP_DAYS* 和:setting:‘PROJECT_BACKUP_KEEP_COUNT‘的配置保留在服务器上（默认情况下最多保留 3 个备份 30 天）。

2.4.2 使用 BorgBackup 进行的自动化备份

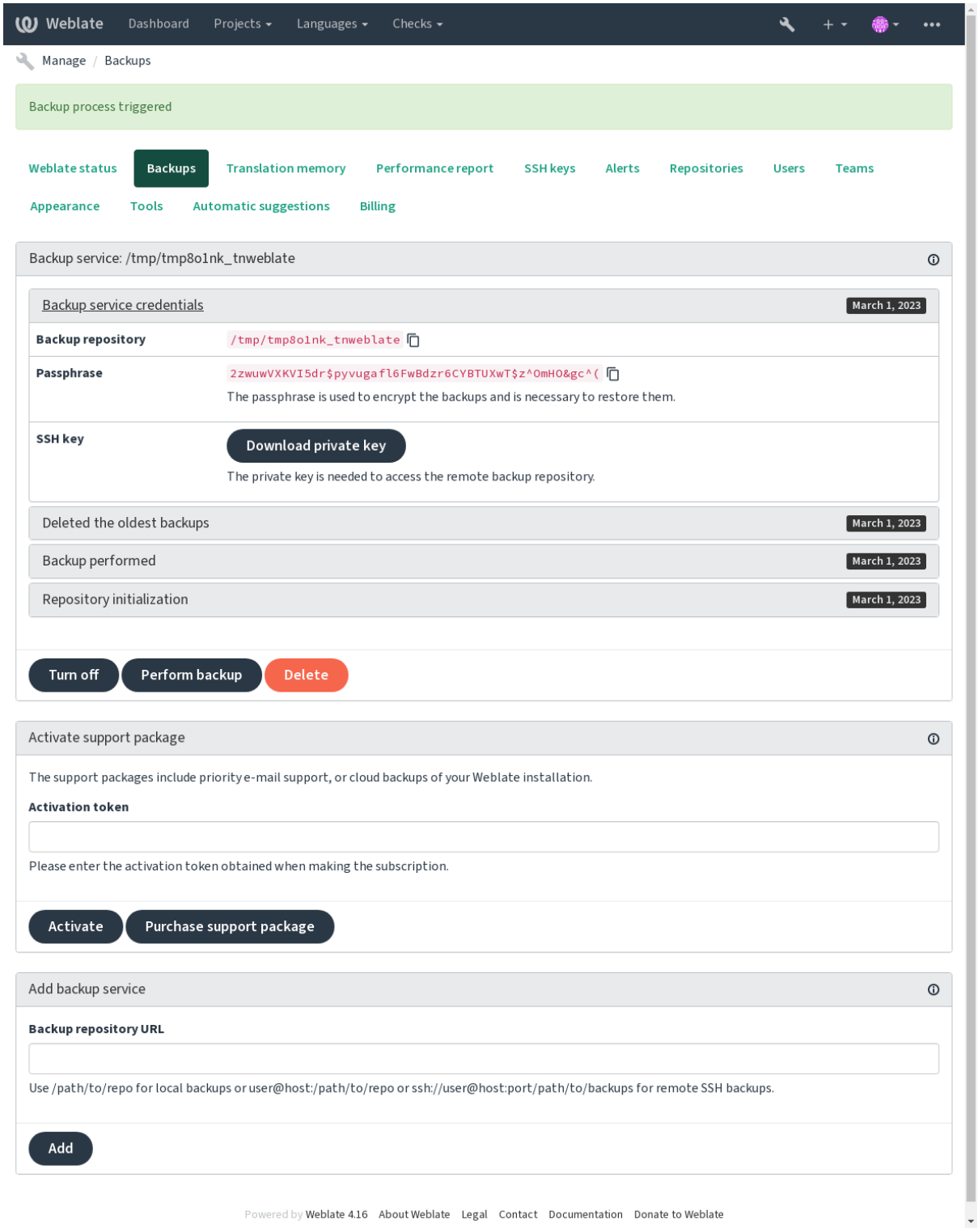
在 3.9 版本加入.

Weblate 内置了对使用 BorgBackup 创建服务备份的支持。Borg 创建了节省空间的加密备份，可以安全地存储在云中。可以从管理界面中的 备份选项卡上控制备份。

在 4.4.1 版本发生变更: PostgreSQL 和 MySQL/MariaDB 数据库都包括在自动备份中。

使用 Borg 的备份是增量的，Weblate 被配置为保留以下备份：

- 每天备份之前 14 天的内容
- 前 8 周的每周备份
- 前 6 个月的每月备份


Weblate
Dashboard
Projects
Languages
Checks
Manage / Backups
Backup process triggered
Weblate status
Backups
Translation memory
Performance report
SSH keys
Alerts
Repositories
Users
Teams
Appearance
Tools
Automatic suggestions
Billing
Backup service: /tmp/tmp8o1nk_tnweblate
Backup service credentials
March 1, 2023
Backup repository
/tmp/tmp8o1nk_tnweblate
Passphrase
2zwuwVXKVI5dr$pyvugafl6FwBdzr6CYBTUXwT$z^OmHO&gc^(
The passphrase is used to encrypt the backups and is necessary to restore them.
SSH key
Download private key
The private key is needed to access the remote backup repository.
Deleted the oldest backups
March 1, 2023
Backup performed
March 1, 2023
Repository initialization
March 1, 2023
Turn off
Perform backup
Delete
Activate support package
The support packages include priority e-mail support, or cloud backups of your Weblate installation.
Activation token
Please enter the activation token obtained when making the subscription.
Activate
Purchase support package
Add backup service
Backup repository URL
Use /path/to/repo for local backups or user@host:/path/to/repo or ssh://user@host:port/path/to/backups for remote SSH backups.
Add
Powered by Weblate 4.16
About Weblate
Legal
Contact
Documentation
Donate to Weblate

Borg 加密密钥

BorgBackup 创建加密的备份，如果没有密码，你将无法恢复它们。密码是在添加新的备份服务时生成的，你应该复制它并将其保存在一个安全的地方。

如果你在使用*Weblate 提供的备份存储*，请同样备份你的私有 SSH 密钥，因为它用来访问你的备份。

参见：

borg init

定制备份

- 可通过*DATABASE_BACKUP* 配置数据库备份。
- 可以使用:setting:‘BORG_EXTRA_ARGS ‘自定义备份创建。

2.4.3 Weblate 提供的备份存储

备份你的 Weblate 实例的最简单方式是购买**‘weblate.org 的备份服务 <https://weblate.org/support/#backup>‘_**。这是你如何让它运行起来的方式：

1. 在 https://weblate.org/support/#backup 购买 备份服务。
2. 在管理界面输入得到的密钥，请参见集成支持。
3. Weblate 将连接到云服务，并获取备份的访问信息。
4. 在 备份选项卡下开启新的备份配置。
5. 备份你的 Borg 凭据，以便能够恢复备份，请参见*Borg 加密密钥*。

提示： 为了安全起见，有打开所有东西的手动步骤。没有你的同意，就不会有数据发送到通过注册步骤得到的备份仓库。

2.4.4 使用客户的备份存储

也可以使用自己的存储来备份。SSH 可以用于在远程目的地存储备份，目标服务器需要安装 BorgBackup。

参见：

Borg 文档中的 General

本地文件系统

推荐去指定本地备份的绝对路径，例如 */path/to/backup*。该目录必须可由运行 weblate 的用户写入（请参见文件系统权限）。在目录不存在的情况下，Weblate 会尝试新建它，但需要适当的权限才能这么做。

提示： 在 Docker 中运行 Weblate 时，请确保备份位置暴露为来自 Weblate 容器的一个卷。否则，备份文件将在其所在的容器重启时被 Docker 丢弃。

一个选项是将备份放置在一个现有的卷中。例如，`/app/data/borgbackup`。这是容器中的一个现有的卷。

你也可以在 Docker 的编写文件中为备份目的添加一个新的容器，例如使用 `/borgbackup`:

```
services:
  weblate:
    volumes:
      - /home/weblate/data:/app/data
      - /home/weblate/borgbackup:/borgbackup
```

备份所存储的目录由 UID 1000 所有，否则 Weblate 将无法把备份写入那里。

远程备份

要创建远程备份，你必须将 BorgBackup 安装到另一台服务器上，你的 Weblate 部署要能使用如下 Weblate SSH 密钥通过 SSH 访问此服务器：

1. 准备一个用于存储备份的服务器。
2. 在上面安装 SSH 服务器（大多数 Linux 发行版默认情况下都会安装 SSH 服务器）。
3. 在该服务器上安装 BorgBackup；大多数 Linux 发行版都有可用的软件包（参见 Installation）。
4. 选择一个现有用户或创建一个用于备份的新用户。
5. 为用户添加 Weblate SSH 密钥，使 Weblate 可以在不需要密码的情况下 SSH 到服务器（参见*Weblate SSH 密钥*）。
6. 将 Weblate 备份位置配置为 `user@host:/path/to/backups` 或 `ssh://user@host:port/path/to/backups`.

提示： *Weblate 提供的备份存储* 为你提供毫不费力的自动远程备份。

参见：

Weblate SSH 密钥, General

2.4.5 从 BorgBackup 恢复

1. 恢复功能会访问你的备份仓库，并准备备份密码。
2. 用 `borg list REPOSITORY` 列出服务器上的所有备份。
3. 使用 `borg extract REPOSITORY::ARCHIVE` 将所需备份恢复到当前目录。
4. 从放置在 Weblate 数据目录下 `backup` 目录中的 SQL 备份中恢复数据库（请参见下载的数据用于备份）。
5. 将 Weblate 配置（`backups/settings.py`，请参见下载的数据用于备份）复制到正确的位置，请参见调整配置。

   使用 Docker 容器时，设置文件已经包含在容器中，您应该恢复原始环境变量。`environment.yml` 文件可能会帮助你解决这个问题（参见下载的数据用于备份）。
6. 将整个存储的数据目录复制到*DATA_DIR* 所配置的位置。

   使用 Docker 容器时，将数据置于数据卷，见*Docker 容器卷*。

   请确保文件具有正确的所有权和权限，请参阅文件系统权限。

Borg 会话可能看上去是这个样子的：

```
$ borg list /tmp/xxx
Enter passphrase for key /tmp/xxx:
2019-09-26T14:56:08                  Thu, 2019-09-26 14:56:08␣
↪[de0e0f13643635d5090e9896bdaceb92a023050749ad3f3350e788f1a65576a5]
```

(续下页)

(接上页)

```
$ borg extract /tmp/xxx::2019-09-26T14:56:08
Enter passphrase for key /tmp/xxx:
```

参见:

borg list, borg extract

2.4.6 手动备份

根据要保存的内容，备份 Weblate 在各个地方存储的数据类型。

提示: 如果你正进行手动备份，你也许想要关闭 Weblate 关于缺乏备份的警告，方法是添加 `weblate.I028` 到 `settings.py` 中的 `SILENCED_SYSTEM_CHECKS`；对于 Docker，则是 *`WEBLATE_SILENCED_SYSTEM_CHECKS`*。

```
SILENCED_SYSTEM_CHECKS.append("weblate.I028")
```

数据库

实际存储位置取决于数据库的设置。

提示: 数据库是最重要的存储。定期对数据库进行备份。没有数据库，所有的翻译都会消失。

本地数据库备份

推荐的方式是使用数据库自带工具如 **`pg_dump`** 或 **`mysqldump`** 来保存数据库的转储。这通常比 Django 备份表现得好，并且可以连同数据一道，恢复完整表格。

你可以在较新的 Weblate 发行版中恢复这个备份，当运行于 `migrate` 时，它将执行所有必需的迁移。请参考*升级 Weblate* 了解如何在版本间升级的更多详细信息。

Django 数据库备份

另外，可以使用 Django 的 `dumpdata` 命令备份你的数据库。那种方式是不依托数据库的，并且可以用于先要更改数据库后端的情况。

恢复数据库之前，你需要确保恢复备份和执行备份的实例运行的是完全相同的 Weblate 版本。这是必要的，因为数据库结构在不同版本之间会发生变化，你可能会以某种方式破坏数据。安装相同版本后，用 `migrate` 运行所有数据库迁移。

之后，一些条目将已经在数据库中创建，你也会在数据库备份中看到它们。推荐的方法是使用管理 shell 手动删除这些条目 (见：ref:*invoke-manage*)：

```
weblate shell
>>> from weblate.auth.models import User
>>> User.objects.get(username='anonymous').delete()
```

文件

如果你有足够的备份空间，只需备份整个*DATA_DIR*。这是一个安全带，即使它包含一些你不想要的文件。下面的部分详细描述了应该备份和可以跳过的内容。

下载的数据用于备份

在 4.7 版本发生变更: 环境变量被转储在 `environment.yml`，以帮助在 Docker 中恢复环境。

存储在*DATA_DIR* `/backups` 中。

Weblate 这里备份各种数据，可以包括这些文件用于更完整的备份。文件每日更新（需要运行 Celery beat 服务器，请参见*使用 Celery 的后台任务* ）。当前，这包括:

- Weblate 设置为 `settings.py` （还有扩展版，在 `settings-expanded.py` ）。
- PostgreSQL 数据库备份为 `database.sql`。
- 环境变量转储在 `environment.yml`。

数据库备份默认保存为纯文本，但也可以通过:setting:‘DATABASE_BACKUP‘来进行压缩或整个跳过。

要恢复数据库备份，请使用数据库工具加载它，例如:

```
psql --file=database.sql weblate
```

版本控制仓库

存储在*DATA_DIR* “/vcs“中。

版本控制仓库包含带有 Weblate 更改的上游仓库的副本。如果你对所有翻译部件启用了:ref:*component-push_on_commit*‘，那么所有 Weblate 变化都包括在上游。不需要在 Weblate 端备份仓库，因为它们可以从上游位置再次克隆，不会丢失数据。

SSH 和 GPG 密钥

存储在*DATA_DIR* `/ssh` 和*DATA_DIR* `/home` 中。

如果正在使用 Weblate 生成的 SSH 或 GPG 密钥，你应该备份这些位置。否则将丢失私有密钥，并且你将不得不重新生成新的密钥。

用户上传的文件

存储在*DATA_DIR* `/media` 中。

你应当备份所有用户上传的文件（例如*字符串的可视化上下文* ）。

Celery 任务

Celery 任务队列可能会包含一些信息，但通常无需进行备份。你最多会丢失尚未被翻译记忆库处理的更新。无论如何，建议在恢复时执行全文或仓库更新，这样就不会有丢失这些内容的问题。

参见:

使用 Celery 的后台任务

手动备份的命令行

使用 cron 作业，您可以设置一条每天执行的 Bash 命令，例如：

```
$ XZ_OPT="-9" tar -Jcf ~/backup/weblate-backup-$(date -u +%Y-%m-%d_%H%M%S).xz␣
↪backups vcs ssh home media fonts secret
```

'XZ_OPT'后面引号之间的字符串允许你选择自己的 xz 选项，例如用于压缩的内存量；见 https://linux.die.net/man/1/xz

你可以根据需要调整文件夹和文件的列表。为了避免保存翻译记忆库（在备份文件夹中），你可以使用：

```
$ XZ_OPT="-9" tar -Jcf ~/backup/weblate-backup-$(date -u +%Y-%m-%d_%H%M%S).xz␣
↪backups/database.sql backups/settings.py vcs ssh home media fonts secret
```

2.4.7 恢复手动备份

1. 将已经备份的所有数据恢复。
2. 使用*updategit* 更新所有仓库。

   ```
   weblate updategit --all
   ```

2.4.8 移动 Weblate 安装

按照上面备份与恢复说明，将你的安装迁移到不同系统。

参见：

从 Python 2 升级到 Python 3, *从其它数据库迁移到 PostgreSQL*

2.5 身份验证

2.5.1 注册用户

Weblate 的默认设置使用 python-social-auth，网站上处理新用户注册的一种形式。确定电子邮箱后，新用户可以通过使用一种第三方服务来贡献或证实。

还可以使用*REGISTRATION_OPEN* 关闭新用户注册。

身份验证尝试服从于*频次限制*。

2.5.2 身份验证后端

Django 的内置解决方案用途是进行身份验证，包括用各种社交登录选项进行验证。使用它意味着可以导入其他基于 Django 项目的用户数据库（请参见*从 Pootle 迁移*）。

Django 还可以通过其他方式进行身份验证。

参见：

身份验证设置 描述了如何配置官方 Docker 镜像的身份验证。

2.5.3 社交身份验证

由于 Python 社交认证，Weblate 支持很多使用第三方服务的身份验证，如 GitLab 、Ubuntu 、Fedora 等。

请查阅 Django 框架 文档中的通用配置说明。

备注： Weblate 默认依赖于第三方身份验证服务来提供合法的电子邮箱地址。如果想要使用的一些服务不支持，请通过为其配置 FORCE_EMAIL_VALIDATION，来强制 Weblate 网站上的电子邮箱验证：

```
SOCIAL_AUTH_OPENSUSE_FORCE_EMAIL_VALIDATION = True
```

参见：

Pipeline

启用单独的后端非常简单，只需在 AUTHENTICATION_BACKENDS 设置中添加一个条目（可能还需要添加特定认证方法所需的密钥）。请注意，一些后端默认不提供用户电子邮件，你必须明确地请求，否则 Weblate 将无法正确记入用户所做的贡献。

提示： 大多数身份验证后端都需要 HTTPS。在您的 Web 服务器中启用 HTTPS 后，请使用*ENABLE_HTTPS* 或 Docker 容器中的 *WEBLATE_ENABLE_HTTPS* 配置 Weblate 以正确报告它。

参见：

Python 社交认证后端

OpenID 身份验证

对于基于 OpenID 的服务，通常只要启用它们就行了。后面的部分关于对于 OpenSUSE 、Fedora 和 Ubuntu 允许 OpenID 身份验证：

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.email.EmailAuth",
    "social_core.backends.suse.OpenSUSEOpenId",
    "social_core.backends.ubuntu.UbuntuOpenId",
    "social_core.backends.fedora.FedoraOpenId",
    "weblate.accounts.auth.WeblateUserBackend",
)
```

参见：

OpenID

GitHub 身份验证

需要在 GitHub 上注册一个 OAuth 应用，然后告诉 Weblate 所有的 secrets：

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.github.GithubOAuth2",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Social auth backends setup
SOCIAL_AUTH_GITHUB_KEY = "GitHub Client ID"
```

(续下页)

(接上页)

```
SOCIAL_AUTH_GITHUB_SECRET = "GitHub Client Secret"
SOCIAL_AUTH_GITHUB_SCOPE = ["user:email"]
```

应该配置 GitHub 具有的回调 URL 为 `https://example.com/accounts/complete/github/`。

GitHub for Organizations 和 GitHub for Teams 也有类似的认证后端。他们的设置名为 "SOCIAL_AUTH_GITHUB_ORG_*"和 "SOCIAL_AUTH_GITHUB_TEAM_*"。它们需要额外设置范围 - `SOCIAL_AUTH_GITHUB_ORG_NAME``或 ``SOCIAL_AUTH_GITHUB_TEAM_ID`。它们的回调 URL 是 "https://example.com/accounts/complete/github-org/和 *"https://example.com/accounts/complete/github-teams/*"。

备注： Weblate 在身份验证时提供的回调 URL。在得到 URL 不匹配的错误时，可以根据需要来修复，请参见设置正确的网站域名。

参见：

GitHub

Bitbucket 身份验证

需要在 Bitbucket 上注册应用，然后告诉 Weblate 所有的秘密：

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.bitbucket.BitbucketOAuth2",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Social auth backends setup
SOCIAL_AUTH_BITBUCKET_OAUTH2_KEY = "Bitbucket Client ID"
SOCIAL_AUTH_BITBUCKET_OAUTH2_SECRET = "Bitbucket Client Secret"
SOCIAL_AUTH_BITBUCKET_OAUTH2_VERIFIED_EMAILS_ONLY = True
```

备注： Weblate 在身份验证时提供的回调 URL。在得到 URL 不匹配的错误时，可以根据需要来修复，请参见设置正确的网站域名。

参见：

Bitbucket

Google OAuth 2

为了使用 Google OAuth 2，可以在 <https://console.developers.google.com/> 上注册应用，并允许 Google+ API。

重定向 URL 为 `https://WEBLATE SERVER/accounts/complete/google-oauth2/`

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.google.GoogleOAuth2",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Social auth backends setup
```

(续下页)

(接上页)

```
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = "Client ID"
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = "Client secret"
```

备注： Weblate 在身份验证时提供的回调 URL。在得到 URL 不匹配的错误时，可以根据需要来修复，请参见设置正确的网站域名。

参见：

Google

Facebook OAuth 2

通常通过 OAuth2 服务，需要用 Facebook 来注册应用。一旦完成，就可以新建 Weblate 来使用了：

重定向 URL 为 `https://WEBLATE SERVER/accounts/complete/facebook/`

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.facebook.FacebookOAuth2",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Social auth backends setup
SOCIAL_AUTH_FACEBOOK_KEY = "key"
SOCIAL_AUTH_FACEBOOK_SECRET = "secret"
SOCIAL_AUTH_FACEBOOK_SCOPE = ["email", "public_profile"]
```

备注： Weblate 在身份验证时提供的回调 URL。在得到 URL 不匹配的错误时，可以根据需要来修复，请参见设置正确的网站域名。

参见：

Facebook

GitLab OAuth 2

为了使用 GitLab OAuth 2，需要在 <https://gitlab.com/profile/applications> 上注册应用。

重定向 URL 为 `https://WEBLATE SERVER/accounts/complete/gitlab/`，并确保你标记 *read_user* 范围。

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.gitlab.GitLabOAuth2",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Social auth backends setup
SOCIAL_AUTH_GITLAB_KEY = "Application ID"
SOCIAL_AUTH_GITLAB_SECRET = "Secret"
SOCIAL_AUTH_GITLAB_SCOPE = ["read_user"]

# If you are using your own GitLab
# SOCIAL_AUTH_GITLAB_API_URL = 'https://gitlab.example.com/'
```

备注： Weblate 在身份验证时提供的回调 URL。在得到 URL 不匹配的错误时，可以根据需要来修复，请参见设置正确的网站域名。

参见：

GitLab

Microsoft Azure Active Directory

可以配置 Weblate，使用一般或特定租户进行身份验证。

常见的重定向 URL 为 `https://WEBLATE SERVER/accounts/complete/azuread-oauth2/`，`https://WEBLATE SERVER/accounts/complete/azuread-tenant-oauth2/` 用于租户特定身份验证。

```
# Azure AD common

# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.azuread.AzureADOAuth2",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# OAuth2 keys
SOCIAL_AUTH_AZUREAD_OAUTH2_KEY = ""
SOCIAL_AUTH_AZUREAD_OAUTH2_SECRET = ""
```

```
# Azure AD Tenant

# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.azuread_tenant.AzureADTenantOAuth2",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# OAuth2 keys
SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_KEY = ""
SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_SECRET = ""
# Tenant ID
SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_TENANT_ID = ""
```

备注： Weblate 在身份验证时提供的回调 URL。在得到 URL 不匹配的错误时，可以根据需要来修复，请参见设置正确的网站域名。

参见：

Microsoft Azure Active Directory

Slack

为了使用 Slack OAuth 2，需要在 <https://api.slack.com/apps> 上注册应用。

重定向 URL 为 `https://WEBLATE SERVER/accounts/complete/slack/`。

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.slack.SlackOAuth2",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Social auth backends setup
SOCIAL_AUTH_SLACK_KEY = ""
SOCIAL_AUTH_SLACK_SECRET = ""
```

备注： Weblate 在身份验证时提供的回调 URL。在得到 URL 不匹配的错误时，可以根据需要来修复，请参见设置正确的网站域名。

参见：

Slack

覆盖身份验证方法名称和图标

您可以使用 `SOCIAL_AUTH_<NAME>_IMAGE` 和 `SOCIAL_AUTH_<NAME>_TITLE` 等设置覆盖身份验证方法显示名称和图标。例如，Auth0 的覆盖命名如下所示：

```
SOCIAL_AUTH_AUTH0_IMAGE = "custom.svg"
SOCIAL_AUTH_AUTH0_TITLE = "Custom auth"
```

关闭密码身份验证

通过从 `AUTHENTICATION_BACKENDS` 删除 `social_core.backends.email.EmailAuth`，可以关闭电子邮箱和密码身份验证。总是将 `weblate.accounts.auth.WeblateUserBackend` 保留在那里，它用于 Weblate 核心功能。

禁用电子邮件身份验证将禁用所有电子邮件相关的功能 - 用户邀请或密码重置功能。

小技巧： 对于手动建立的用户，可以仍然在管理界面使用密码身份验证。只需导航到 `/admin/login/`。

例如，使用后面的设置可以实现只是用 openSUSE Open ID 的身份验证：

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.suse.OpenSUSEOpenId",
    "weblate.accounts.auth.WeblateUserBackend",
)
```

2.5.4 密码身份验证

默认 settings.py 与一组合理的设置 AUTH_PASSWORD_VALIDATORS 在一起：

- 密码不能与其它个人信息太相似。
- 密码必须至少包含 10 个字符。
- 密码不能是通常使用的密码。
- 密码不能完全是数字。
- 密码不能只由单个字符或仅空格组成。
- 密码与你过去使用的密码不匹配。

可以自定义这个设置来匹配密码政策。

可以另外安装 django-zxcvbn-password 这会非常实际地估计密码的难度，并允许拒绝低于下面适当阈值的密码。

2.5.5 SAML 身份验证

在 4.1.1 版本加入.

请遵守 Python 社交认证的指示来配置。显著的差异有：

- Weblate 支持单一 IDP，在 SOCIAL_AUTH_SAML_ENABLED_IDPS 中必须称之为 weblate。
- SAML XML 元数据 URL 为 /accounts/metadata/saml/。
- 以下设置会自动填充：SOCIAL_AUTH_SAML_SP_ENTITY_ID、SOCIAL_AUTH_SAML_TECHNICAL_CONTACT、SOCIAL_AUTH_SAML_SUPPORT_CONTACT

配置的示例：

```
# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.email.EmailAuth",
    "social_core.backends.saml.SAMLAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Social auth backends setup
SOCIAL_AUTH_SAML_SP_ENTITY_ID = f"https://{SITE_DOMAIN}/accounts/metadata/saml/"
SOCIAL_AUTH_SAML_SP_PUBLIC_CERT = "-----BEGIN CERTIFICATE-----"
SOCIAL_AUTH_SAML_SP_PRIVATE_KEY = "-----BEGIN PRIVATE KEY-----"
SOCIAL_AUTH_SAML_ENABLED_IDPS = {
    "weblate": {
        "entity_id": "https://idp.testshib.org/idp/shibboleth",
        "url": "https://idp.testshib.org/idp/profile/SAML2/Redirect/SSO",
        "x509cert": "MIIEDjCCAvagAwIBAgIBADA ... 8Bbnl+ev0peYzxFyF5sQA==",
        "attr_name": "full_name",
        "attr_username": "username",
        "attr_email": "email",
    }
}
SOCIAL_AUTH_SAML_ORG_INFO = {
    "en-US": {
        "name": "example",
        "displayname": "Example Inc.",
        "url": "http://example.com"
    }
}
SOCIAL_AUTH_SAML_TECHNICAL_CONTACT = {
```

(续下页)

(接上页)

```
    "givenName": "Tech Gal",
    "emailAddress": "technical@example.com"
}
SOCIAL_AUTH_SAML_SUPPORT_CONTACT = {
    "givenName": "Support Guy",
    "emailAddress": "support@example.com"
}
```

默认配置从以下属性提取用户详细信息，通过配置您的 IDP 来配置它们：

属性	SAML URI 参照
全名	`urn:oid:2.5.4.3`
名字	`urn:oid:2.5.4.42`
姓氏	`urn:oid:2.5.4.4`
电子邮箱	`urn:oid:0.9.2342.19200300.100.1.3`
用户名	`urn:oid:0.9.2342.19200300.100.1.1`

提示： 上面的示例和 Docker 镜像定义了一个叫做 `weblate` 的 IDP。您可能需要在 IDP 中将此字符串配置为 *Relay*。

参见：

在 Docker 中配置 SAML, SAML

2.5.6 LDAP 身份验证

LDAP 身份验证可以使用 *django-auth-ldap* 软件包而最好地实现。可以使用通常的方式安装：

```
# Using PyPI
pip install django-auth-ldap>=1.3.0

# Using apt-get
apt-get install python-django-auth-ldap
```

提示： 此包包含于 Docker 容器中，见*使用 Docker 安装*。

备注： 在 Python LDAP 3.1.0 模块中有一些不兼容，导致可能无法使用那个版本。如果得到错误信息 AttributeError: ‘module’ object has no attribute ‘_trace_level’，将 python-ldap 降回到 3.0.0 版可能会有帮助。

一旦安装了软件包，就可以将其钩入 Django 身份验证了：

```
# Add LDAP backed, keep Django one if you want to be able to sign in
# even without LDAP for admin account
AUTHENTICATION_BACKENDS = (
    "django_auth_ldap.backend.LDAPBackend",
    "weblate.accounts.auth.WeblateUserBackend",
)

# LDAP server address
AUTH_LDAP_SERVER_URI = "ldaps://ldap.example.net"
```

(续下页)

(接上页)

```
# DN to use for authentication
AUTH_LDAP_USER_DN_TEMPLATE = "cn=%(user)s,o=Example"
# Depending on your LDAP server, you might use a different DN
# like:
# AUTH_LDAP_USER_DN_TEMPLATE = 'ou=users,dc=example,dc=com'

# List of attributes to import from LDAP upon sign in
# Weblate stores full name of the user in the full_name attribute
AUTH_LDAP_USER_ATTR_MAP = {
    "full_name": "name",
    # Use the following if your LDAP server does not have full name
    # Weblate will merge them later
    # 'first_name': 'givenName',
    # 'last_name': 'sn',
    # Email is required for Weblate (used in VCS commits)
    "email": "mail",
}

# Hide the registration form
REGISTRATION_OPEN = False
```

备注： 你应当从设置的 AUTHENTICATION_BACKENDS 部分移除 'social_core.backends.email.EmailAuth'，否则用户不能够在 Weblate 中设置他们的密码，并使用它进行身份验证。为了生成权限和方便匿名用户，仍需保留 'weblate.accounts.auth.WeblateUserBackend'。它还允许你使用一个本地管理账户登录，如果你已经创建了它（如通过使用*createadmin*）。

使用绑定密码

如果可以为身份验证使用直接绑定，那么需要使用搜索，并为用户搜索提供绑定，例如：

```
import ldap
from django_auth_ldap.config import LDAPSearch

AUTH_LDAP_BIND_DN = ""
AUTH_LDAP_BIND_PASSWORD = ""
AUTH_LDAP_USER_SEARCH = LDAPSearch(
    "ou=users,dc=example,dc=com", ldap.SCOPE_SUBTREE, "(uid=%(user)s)"
)
```

Active Directory 集成

```
import ldap
from django_auth_ldap.config import LDAPSearch, NestedActiveDirectoryGroupType

AUTH_LDAP_BIND_DN = "CN=ldap,CN=Users,DC=example,DC=com"
AUTH_LDAP_BIND_PASSWORD = "password"

# User and group search objects and types
AUTH_LDAP_USER_SEARCH = LDAPSearch(
    "CN=Users,DC=example,DC=com", ldap.SCOPE_SUBTREE, "(sAMAccountName=%(user)s)"
)

# Make selected group a superuser in Weblate
AUTH_LDAP_USER_FLAGS_BY_GROUP = {
```

(续下页)

(接上页)

```
    # is_superuser means user has all permissions
    "is_superuser": "CN=weblate_AdminUsers,OU=Groups,DC=example,DC=com",
}

# Map groups from AD to Weblate
AUTH_LDAP_GROUP_SEARCH = LDAPSearch(
    "OU=Groups,DC=example,DC=com", ldap.SCOPE_SUBTREE, "(objectClass=group)"
)
AUTH_LDAP_GROUP_TYPE = NestedActiveDirectoryGroupType()
AUTH_LDAP_FIND_GROUP_PERMS = True

# Optionally enable group mirroring from LDAP to Weblate
# AUTH_LDAP_MIRROR_GROUPS = True
```

参见:

Django Authentication Using LDAP, Authentication

2.5.7 CAS 身份验证

可以使用软件包如 *django-cas-ng* 来实现 CAS 身份验证。

第一步通过 CAS 揭示了用户电子邮箱字段。这必须在 CAS 服务器自身来配置，并需要至少运行 CAS v2，因为 CAS v1 不支持属性。

第二步更新 Weblate，来使用 CAS 服务器和属性。

为了安装 *django-cas-ng*：

```
pip install django-cas-ng
```

一旦安装了软件包，就可以通过修改 `settings.py` 文件将其钩连到 Django 身份验证系统：

```
# Add CAS backed, keep the Django one if you want to be able to sign in
# even without LDAP for the admin account
AUTHENTICATION_BACKENDS = (
    "django_cas_ng.backends.CASBackend",
    "weblate.accounts.auth.WeblateUserBackend",
)

# CAS server address
CAS_SERVER_URL = "https://cas.example.net/cas/"

# Add django_cas_ng somewhere in the list of INSTALLED_APPS
INSTALLED_APPS = (..., "django_cas_ng")
```

最后，可以使用信号将电子邮箱字段投射到用户对象上。为了生效，必须将信号从 *django-cas-ng* 软件包导入，并将你的代码与这个信号连接。在设置文件中这样做可能产生问题，这样建议将它放进去：

- 在你的 app 配置的 `django.apps.AppConfig.ready()` 方法
- 在项目的 `urls.py` 文件中（当没有模块存在时）

```
from django_cas_ng.signals import cas_user_authenticated
from django.dispatch import receiver


@receiver(cas_user_authenticated)
def update_user_email_address(sender, user=None, attributes=None, **kwargs):
    # If your CAS server does not always include the email attribute
    # you can wrap the next two lines of code in a try/catch block.
```

(续下页)

(接上页)

```
    user.email = attributes["email"]
    user.save()
```

参见:

Django CAS NG

2.5.8 配置第三方 Django 身份验证

一般地，任何 Django 身份认证插件应该可以在 Weblate 上工作。只需要按照插件的说明，只记住安装了 Weblate 用户后台。

参见:

LDAP 身份验证, *CAS 身份验证*

典型的安装包括，将身份验证后台添加到 `AUTHENTICATION_BACKENDS`，并将身份验证 app （如果有的话）安装到 `INSTALLED_APPS`：

```
AUTHENTICATION_BACKENDS = (
    # Add authentication backend here
    "weblate.accounts.auth.WeblateUserBackend",
)

INSTALLED_APPS += (
    # Install authentication app here
)
```

2.6 访问控制

Weblate 带有细粒度的权限系统，可以为整个实例或在有限范围内分配用户权限。

在 3.0 版本发生变更: 在 Weblate 3.0 之前，权限系统只是基于 Django 的权限系统，但现在是专门为 Weblate 打造的。如果使用的是更旧的版本，请查阅所使用的特定版本的文档。

2.6.1 简单的访问控制

如果你不是掌控完整的 Weblate 安装，而仅仅是需要访问管理当前项目（如在 Hosted Weblate），你的访问控制管理选项将被设置所限制。如果你不需要任何复杂的步骤，这些就足够了。

项目访问控制

备注: 此功能对于在 Hosted Weblate 上运行自由套餐的项目不可用。

您可以通过选择不同的 *访问控制* 设置来限制用户对单个项目的访问。可用的选项有:

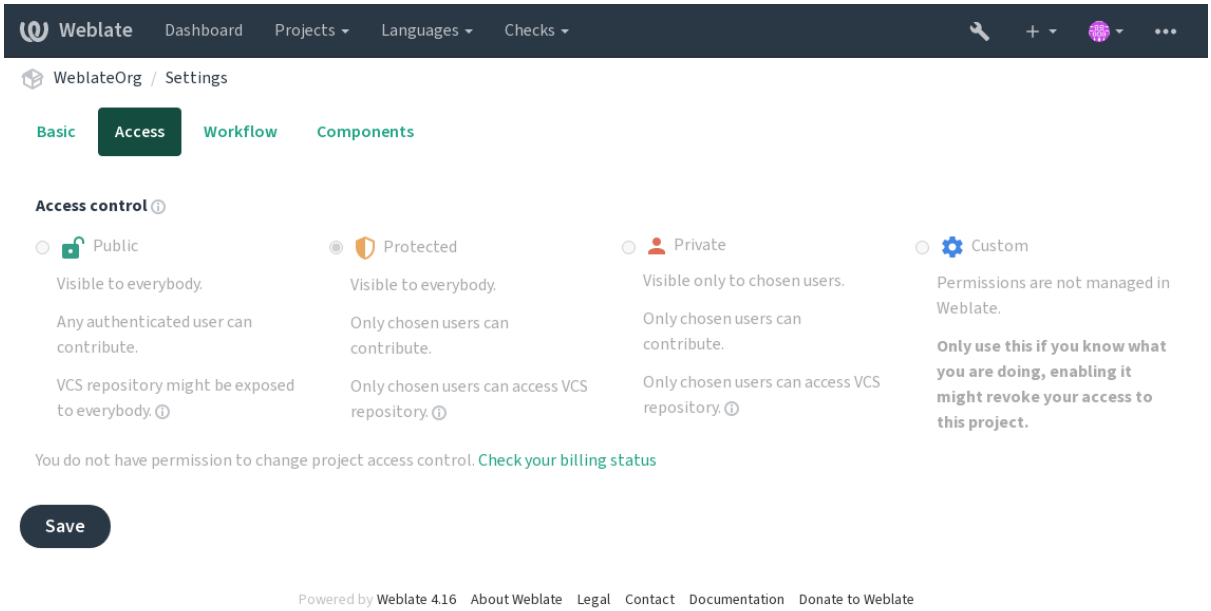
公开的
: 公开可见，所有登录用户均可进行翻译。

受保护的
: 公开可见，但仅选定的用户可进行翻译。

私有的
: 仅选定的用户可见和可进行翻译。

自定义
: 用户管理 功能将被禁用；默认情况下，禁止所有用户对项目执行任何操作。您必须使用自定义访问控制 来设置所有权限。

可以在每个项目的配置（管理 ↓ 设置）的 访问选项卡中更改 访问控制。



可以通过*DEFAULT_ACCESS_CONTROL* 更改默认值。

备注： 即使是 私有项目，也会暴露有关项目的一些信息：尽管进行了访问控制设置，但整个实例的统计信息和语言概况仍将包括所有项目的计数。你的项目名称和其他信息不会因此暴露。

备注： Weblate 实例管理员可通过自定义设置 重新定义 公开、受保护和 私有项目中默认提供的用户权限组合。

参见：

访问控制

管理每个项目的访问控制

有 管理项目访问权特权的用户可以通过将用户添加到团队中来管理他们。最初的一组团队由 Weblate 提供，但可以定义其他团队，以提供更细粒度的访问控制。你可以限制团队到特定语言，并为他们分配指定的访问角色（参见权限和内置角色列表）。

Weblate 会自动为每个项目创建以下团队：

适用于 公开、受保护和 私有项目：

管理
: 包括项目可用的所有权限。

审校（仅当开启审校流程 时）
: 可以在审校时核准翻译。

只适用于 受保护和 私有项目：

翻译
: 可以翻译项目，并将离线的翻译上传。

原文
可以编辑源字符串（如果项目设置 中允许）和源字符串信息。

语言
可以管理翻译语言（添加或删除翻译）。

术语表
可以管理术语表（添加或删除条目，也可以上传）。

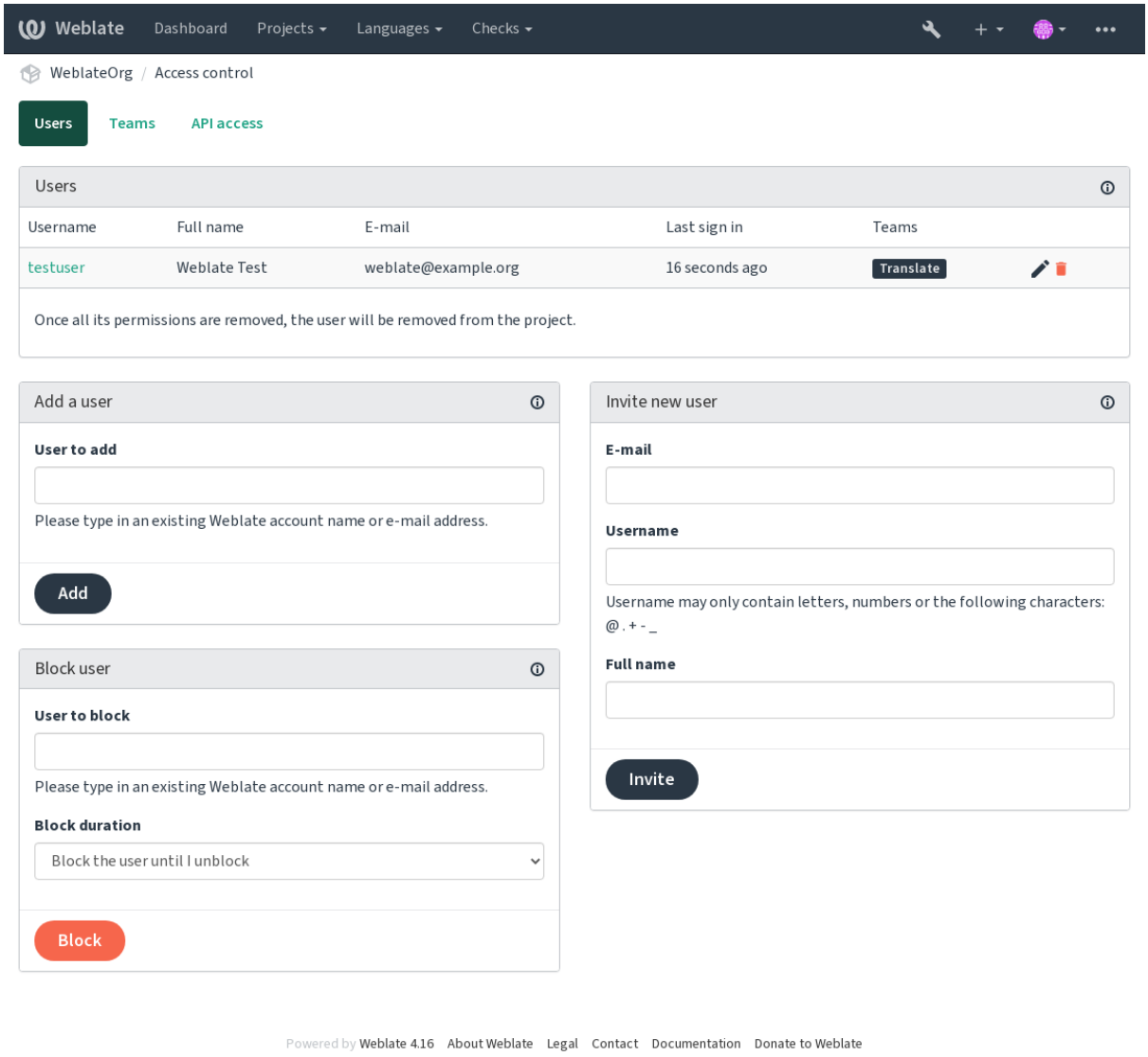
记忆存储
可以管理翻译记忆库。

截屏
可以管理截屏（添加或删除截屏，并将其与源字符串关联）。

自动翻译
可以使用自动翻译。

版本控制系统（VCS）
可以管理版本控制系统（VCS）并访问导出的仓库。

账单
可以访问账单信息和设置（请参见账单 ）。



这些功能可在 访问控制页面上找到，可以从项目的菜单 管理 ↓ 用户 进入。

团队管理员

在 4.15 版本加入.

每个团队都可以有团队管理员，他们可以在团队中添加和删除用户。这在您想要构建自治团队时很有用。

邀请新用户

此外，除了将现有用户添加到项目之外，还可以邀请新用户。将立即创建任何新用户，但是该账户将保持不活动状态，直到使用通过电子邮件发送的邀请中的链接登录为止。不需要具有任何站点范围的权限，就可以这么做，项目级别的访问管理权限（如 *Administration* 团队成员资格）就足够了。

提示： 如果被邀请的用户错过了邀请的有效性，则可以在密码重置表单中使用被邀请的电子邮箱地址设置密码，因为已经创建了账户。

在 3.11 版本加入: 可以重新发送用户邀请电子邮件（将使之前发送的所有邀请失效）。

通过 *用户* 选项卡下的*管理界面* 获取可在站点范围内使用的同样的邀请。

阻止用户

在 4.7 版本加入.

如果有些用户在您的项目中行为不佳，您可以选择阻止他们进行贡献。如果被阻止的用户有权限，那他仍然可以看到项目，但他将不能贡献。

每个项目的权限管理

你可以将你的项目设置为 *受保护*或 *私有*，并在 Weblate 的界面上进行每个项目的*用户管理*。

默认情况下，这可防止 Weblate 因 *用户*和 *查看者* 默认团队本身的配置而授予其提供的访问权限。但这并不妨碍您向这些项目授予站点范围的权限，可用的方法有改变默认团队、创建一个新团队，或如下方*自定义访问控制* 中所述为单个部件创建额外的自定义设置。

通过 Weblate 用户界面管理权限的一个主要好处是，您可以将其委托给其他用户，而无需授予他们超级用户权限。为此，请将 TA 们添加到项目的 *Administration* 团队中。

2.6.2 自定义访问控制

备注： 此功能对于在 Hosted Weblate 上运行自由套餐的项目不可用。

权限系统基于团队和角色，其中角色定义了一组权限，团队将它们链接到用户和翻译，请参阅*用户、角色、团队和权限* 获取更多详细信息。

Weblate 访问控制系统最强大的功能目前只能通过*Django 管理界面* 来实现。您可以使用它来管理任何项目的权限。你不一定要切换到 *自定义访问控制* 来使用它。但是你必须有超级用户权限才能使用它。

如果您对实现的细节不感兴趣，只是想创建一个基于默认值的足够简单的配置，或者不能访问整个 Weblate 安装（如 Hosted Weblate），请参见*简单的访问控制* 部分。

通用设置

本节包含一些您可能感兴趣的常见配置的概述。

站点范围内的权限管理

要一次性管理整个实例的权限，请将用户添加到适当的 默认团队：

- 用户（默认情况下，这是由自动团队分配 完成的）。
- 审校员（如果您有专门的审校员，正在使用审校流程）。
- 管理者（如果您想把大部分的管理操作委托给别人）。

您应该将所有项目配置为 公开的（参见项目访问控制），否则 用户和 审校员团队成员资格提供的站点范围权限将不起任何作用。

您还可以向默认团队授予一些您选择的额外权限。例如，你可能想给所有 用户管理截图的权限。

您还可以定义一些新的自定义团队。如果希望继续在站点范围内管理这些团队的权限，请为 项目选择选择一个合适的值（如 所有项目或 所有公开项目）。

语言、部件或项目的自定义权限

您可以创建自己的专门团队来管理不同对象（如语言、部件和项目）的权限。虽然这些团队只能授予额外的权限，但您不能通过添加另一个自定义团队来撤销全站或每个项目团队授予的任意权限。

示例：

如果您希望（不管出于什么原因）只允许一组封闭的可靠译者来进行特定语言（比如 捷克语）的翻译，同时将其他语言的翻译保持公开，你必须：

1. 移除所有用户翻译 捷克语的权限。在默认配置中，这可以通过修改 用户默认团队 来实现。

表 1: 用户群组

语言选择	定义
语言	除 捷克语之外的所有语言

2. 为 捷克语译者添加一个专门的团队。

表 2: 捷克语译者小组

角色	高级用户
项目选择	所有公共项目
语言选择	定义
语言	捷克语

3. 将您希望授予此权限的用户添加到此团队中。

如您所见，这种方式的权限管理功能强大，但可能是一项相当乏味的工作。 您不能将其委托给其他用户，除非授予超级用户权限。

用户、角色、团队和权限

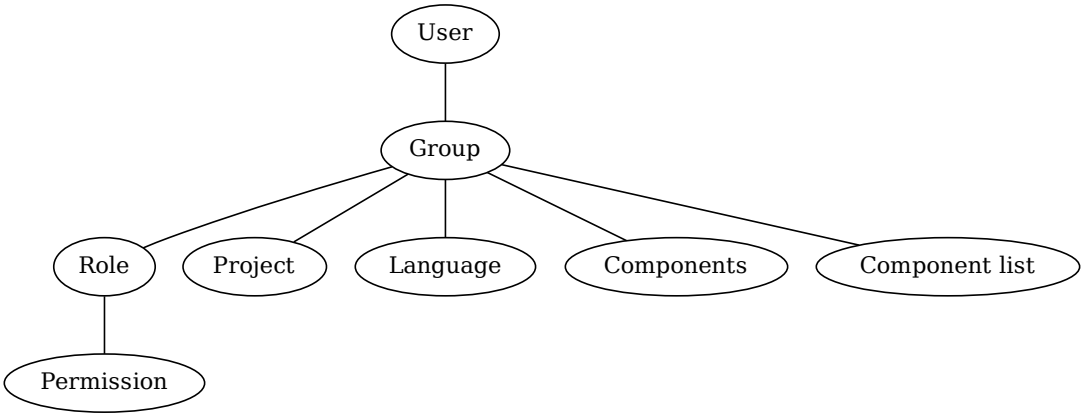
身份验证模型由几个对象组成:

权限
Weblate 定义的个人权限。权限不能分配给用户。这只能通过分配角色来完成。

角色
角色定义了一组权限。这允许在多个地方重用这些集，使管理更加容易。

用户
用户可以属于多个团队。

群组
分组连接角色、用户和身份验证对象（项目、语言和部件列表）。



备注: 团队可以不分配任何角色，但在这种情况下任何人都可以浏览它的项目（见下文）。

浏览到项目的访问权限

用户必须是链接到项目的团队或项目中的任何部件的成员。拥有成员身份就足够了，浏览项目不需要特定的权限（这被用于默认的 查看者团队，见团队列表 ）。

浏览部件的权限

用户一旦能够访问部件的项目，就可以访问不受限制的部件（并将拥有该项目授予用户的所有权限）。在开启*受限制的访问* 的情况下，访问部件需要对该部件（或该部件所在的部件列表）具有显式权限。

团队范围

团队内角色分配的权限范围适用如下规则：

- 如果团队指定了任何 *部件列表*，则授予该团队成员的所有权限都适用于附加到该团队的部件列表中的所有部件，并为这些部件所在的所有项目授予没有额外权限的访问权限。*部件* 和 *项目* 将被忽略。
- 如果该团队指定了任何 *部件*，则授予该团队成员的所有权限都适用于附加到该团队的所有部件，并为这些部件所在的所有项目授予没有额外权限的访问权限。*项目* 将被忽略。
- 否则，如果团队指定了任何 *项目*，通过直接列出它们或通过将 *项目选择* 设置为类似 *所有公开项目* 的值，所有这些权限将应用于所有项目，它有效地授予访问所有项目的相同权限*不受限制的部件*。
- 当验证用户是否有权限执行某些操作时，由团队的 *语言* 施加的限制将单独应用。也就是说，它仅适用于与翻译过程本身直接相关的操作，例如审校、保存翻译、添加建议等。

提示： 使用 *语言选择* 或 *项目选择* 来自动包括所有语言或项目。

示例：

假设有一个项目 `foo` 包含以下部件：`foo/bar` 和 `foo/baz`，以及以下团队：

表 3: 西班牙语管理审校员群组

角色	*审校字符串*、*管理仓库*
部件	foo/bar
语言	*西班牙语*

该团队的成员将具有以下权限（假设是默认的角色设置）：

- 对整个项目 `foo` 的一般（浏览）访问，包括其中的两个部件：`foo/bar` 和 `foo/baz`。
- 审校 `foo/bar` 西班牙语翻译（而非其他地方）中的字符串。
- 管理整个 `foo/bar` 仓库的版本控制系统（VCS），例如提交所有语言的译者所做的待处理更改。

自动团队分配

在*Django 管理界面* 的 *群组编辑*页面的底部，可以指定 *自动团队分配*，这是一个正则表达式列表，用于根据他们的电子邮箱地址自动将新创建的用户分配到团队。此分配仅在创建账户时发生。

该功能最常见的用例是将所有新用户分配到某个默认团队。为此，您可能希望在正则表达式字段中保留默认值（`^.*$`）。此选项的另一个用例可能是默认为您公司的员工提供一些额外的权限。假设他们都使用您域中的公司电子邮箱地址，这可以通过像 `^.*@mycompany.com` 这样的表达式来完成。

备注： 从一个 Weblate 版本升级到另一个版本时，总是会重新创建对 *用户* 和 *查看者* 的自动团队分配。如果你想关闭它，将正则表达式设置为 `^$` （不匹配任何内容）。

备注： 就目前而言，还没有办法通过用户界面将现有用户批量添加到某个团队。为此，您可以通过*REST API* 来实现。

默认团队和角色

安装完成后，将创建一组默认团队（参见团队列表）。

这些角色和团队是在安装时创建的。升级时，内置角色始终通过数据库迁移保持最新状态。实际上你无法更改它们，如果要定义自己的权限组合，请定义一个新角色。

权限和内置角色列表

范围	权限	角色
账单（请参见账单）	查看账单信息	管理、账单
变化	下载变化	管理
评论	发表评论	管理、编辑原文、高级用户、审校字符串、翻译
	删除评论	管理
	解决评论	管理、审校字符串
部件	编辑部件设置	管理
	锁定部件，阻止翻译	管理
术语表	添加术语表条目	管理、管理术语表、高级用户
	编辑术语表条目	管理、管理术语表、高级用户
	删除术语表条目	管理、管理术语表、高级用户
	上传术语表条目	管理、管理术语表、高级用户
自动建议	使用自动建议	管理、编辑原文、高级用户、审校字符串、翻译
翻译记忆库	编辑翻译记忆	管理、管理翻译记忆库
	删除翻译记忆	管理、管理翻译记忆库
项目	编辑项目设置	管理
	管理项目访问权	管理
报告	下载报告	管理
截屏	添加截图	管理、管理截图
	编辑截图	管理、管理截图
	删除截图	管理、管理截图
源字符串	编辑字符串额外信息	管理、编辑原文
字符串	添加新字符串	管理
	移除字符串	管理
	忽略未通过的检查	管理、编辑原文、高级用户、审校字符串、翻译
	编辑字符串	管理、编辑原文、高级用户、审校字符串、翻译
	审校字符串	管理、审校字符串
	当建议被强制执行时需要编辑字符串	管理、审校字符串
	编辑源字符串	管理、编辑原文、高级用户
建议	接受建议	管理、编辑原文、高级用户、审校字符串、翻译
	添加建议	管理、编辑原文、添加建议、高级用户、审校字符串、翻
	删除建议	管理、高级用户
	为建议投票	管理、编辑原文、高级用户、审校字符串、翻译
翻译	添加翻译语言	管理、高级用户、管理语言
	执行自动翻译	管理语言、自动翻译
	删除现有翻译	管理、管理语言
	下载翻译文件	管理、编辑原文、访问仓库、高级用户、审校字符串、翻
	添加多种翻译语言	管理、管理语言
上传	定义上传译文的作者	管理
	上传时覆盖已有字符串	管理、编辑原文、高级用户、审校字符串、翻译
	上传译文	管理、编辑原文、高级用户、审校字符串、翻译
版本控制系统（VCS）	访问内部仓库	管理、访问仓库、高级用户、管理仓库
	将更改提交到内部仓库	管理、管理仓库
	从内部仓库推送更改	管理、管理仓库
	重置内部仓库的更改	管理、管理仓库
	查看上游仓库位置	管理、访问仓库、高级用户、管理仓库

表 4 – 接上页

范围	权限	角色
全网站范围的特权	更新内部仓库	管理、管理仓库
	使用管理界面	
	添加新项目	
	添加语言定义	
	管理语言定义	
	管理团队	
	管理用户	
	管理角色	
	管理公告	
	管理翻译记忆库	
	机器翻译	
	管理部件列表	

备注： 站点范围的特权不会被授予任何默认角色。它们功能强大，非常接近超级用户的地位。它们中的大多数都会影响到你的 Weblate 安装中的所有项目。

团队列表

下面的团队在安装时建立（或在执行`setupgroups`后），您可以自由修改它们。但是，如果它们被删除或重命名，迁移后将重新创建这些名称。

访客
定义非授权用户的权限。

这个团队只包括匿名用户（请参见`ANONYMOUS_USER_NAME`）。

你可以从团队中去掉角色，来限制非授权用户的权限。

默认角色：添加建议、访问仓库

查看者
这一角色确保公开项目对所有用户可见。默认情况下，所有用户都是该团队的成员。

默认情况下，自动团队分配 将会使所有新账户在加入时成为该团队的成员。

默认角色：无

用户
所有用户的默认团队。

默认情况下，自动团队分配 将会使所有新账户在加入时成为该团队的成员。

默认角色：高级用户

审校员
审校员的群组（参见翻译工作流）。

默认角色：审校字符串

管理者
管理员的群组。

默认角色：管理

警告： 切勿移除预先定义的 Weblate 团队和用户，因为这可能会导致意想不到的问题！如果你用不到它们，你可以移除它们的全部权限。

2.6.3 额外访问限制

如果您想以不那么公开的方式使用您的 Weblate 安装，即只允许新用户在受邀的基础上，可以通过配置 Weblate 以这样一种方式来实现，即只有已知用户才能访问它。为此，您可以将*REGISTRATION_OPEN* 设置为 `False` 以防止任何新用户注册，并将*REQUIRE_LOGIN* 设置为 `/.*` 以要求登录访问所有网站页面。这基本上是锁定您的 Weblate 安装的方法。

提示： 您可以使用内置的邀请新用户 来添加新用户。

2.7 翻译项目

2.7.1 翻译组织

Weblate 将项目/部件的可翻译版本控制系统（VCS）内容组织成树状结构。

- 底层对象是项目配置，该项目配置应将所有翻译归在一起（例如，多个版本的应用程序翻译和/或随附的文档）。
- 在上面的级别上，即部件配置，实际上是要翻译的部件，您定义要使用的版本控制系统（VCS）仓库以及要翻译的文件掩码。
- 在部件配置 上方有单独的翻译，当版本控制系统（VCS）仓库中出现翻译文件（与部件配置 中定义的文件掩码 匹配）时，Weblate 会自动处理这些翻译。

Weblate 支持 Translate Toolkit 支持的多种翻译格式（双语和单语），请参阅支持的文件格式。

备注： 您可以使用 *Weblate 内部网址* 共享克隆的版本控制系统（VCS）仓库。当您有许多共享同一版本控制系统（VCS）的部件时，强烈推荐使用此功能。它提高了性能并减少了所需的磁盘空间。

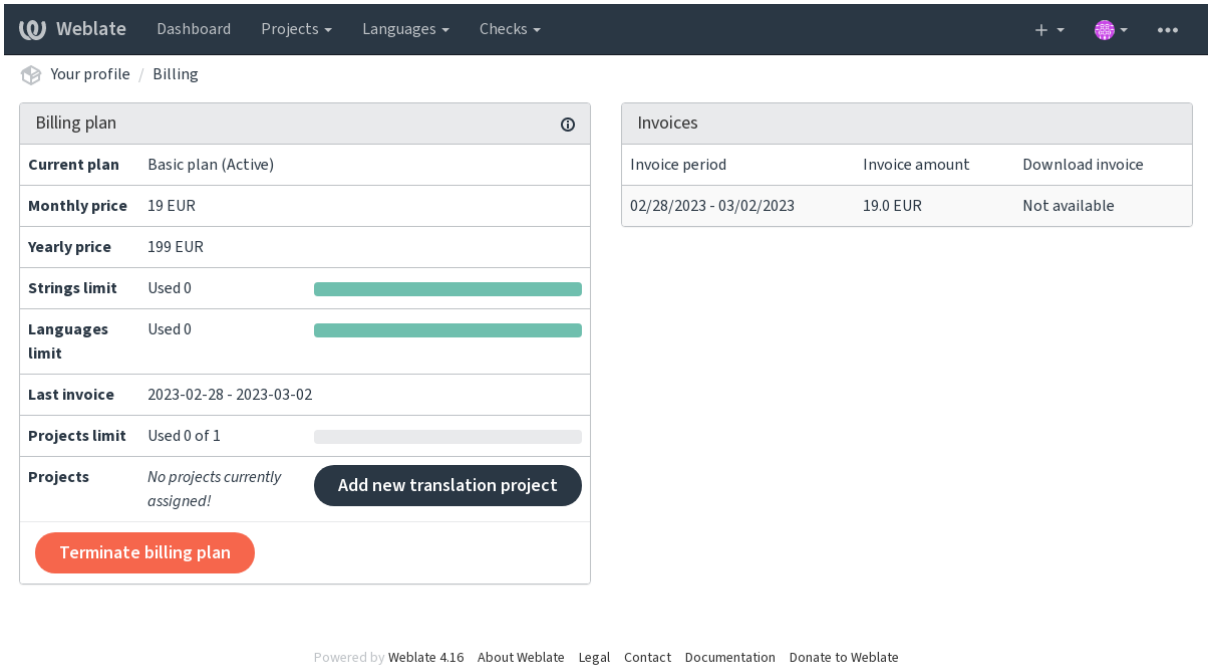
2.7.2 添加翻译项目和部件

在 3.2 版本发生变更: 已包含用于添加项目和部件的界面，您不再需要使用*Django 管理界面*。

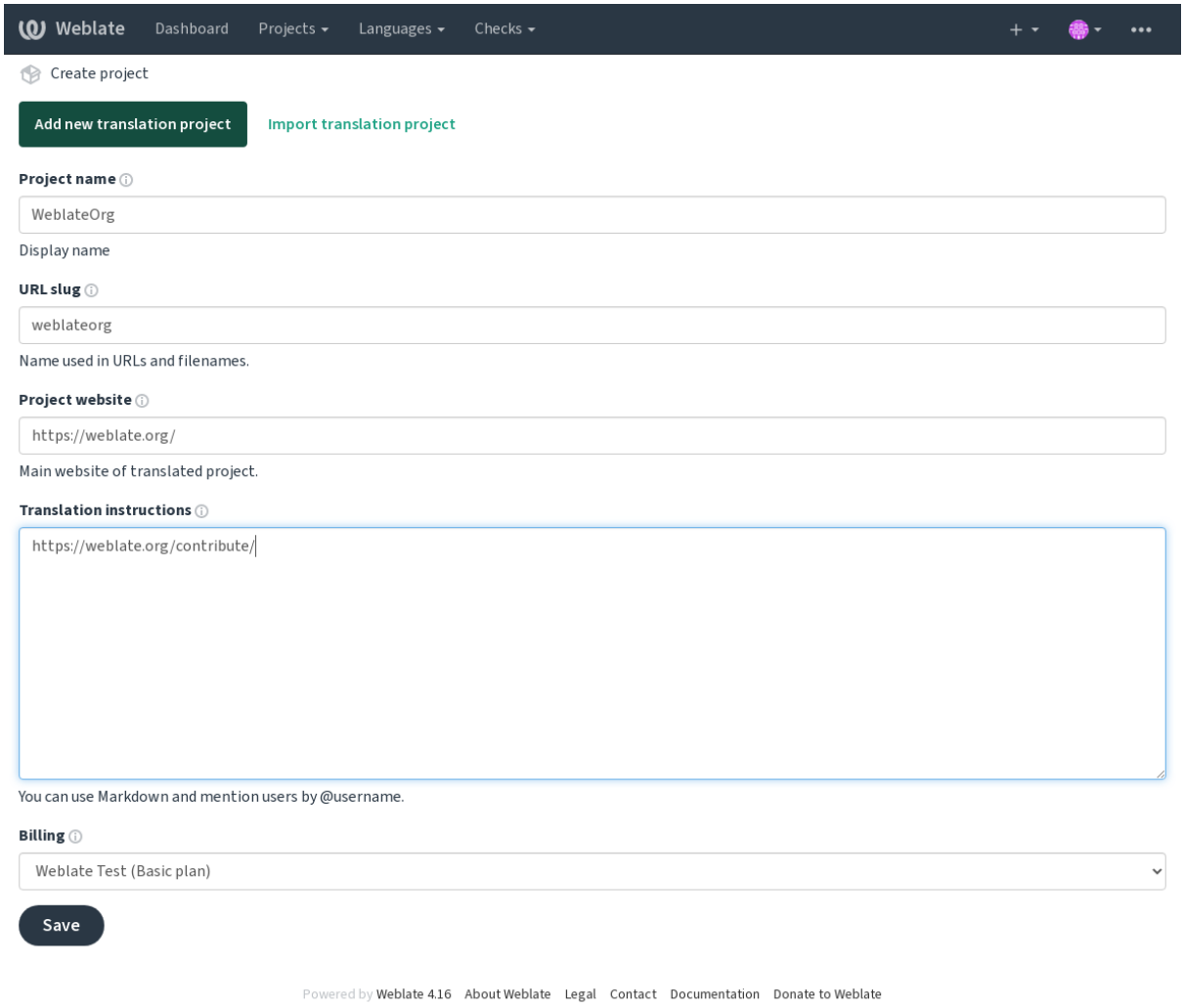
在 3.4 版本发生变更: 现在，添加部件的过程是多阶段的，可以自动发现大多数参数。

根据你的权限，可以创建新的翻译项目和部件。具备 *Add new projects* 权限的用户总是可以这么做。如果你的实例使用付费托管（如 https://hosted.weblate.org/，请参见账单），你还可以从管理账单的用户账户基于套餐限额创建它们。

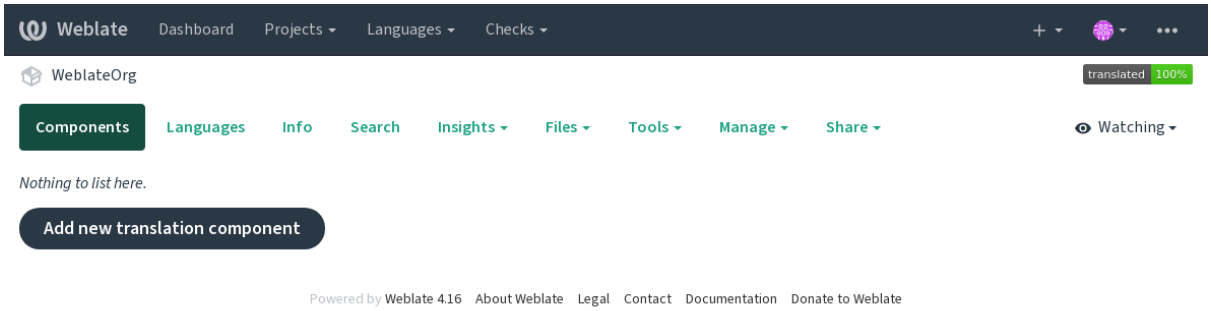
您可以在单独的页面上查看当前的付费套餐：



您可以从此处开始创建项目，也可以使用导航栏中的菜单来填写翻译项目的基本信息以完成添加：



创建项目后，您将直接进入项目页面：



只需单击一次即可启动创建新翻译部件的操作。创建部件的过程是多阶段的，并自动检测大多数翻译参数。有几种创建部件的方法：

从版本控制
从远程版本控制仓库创建部件。

从现有部件
通过选择不同的文件为现有部件创建其他部件。

其他分支
仅针对不同分支，为现有部件创建其他部件。

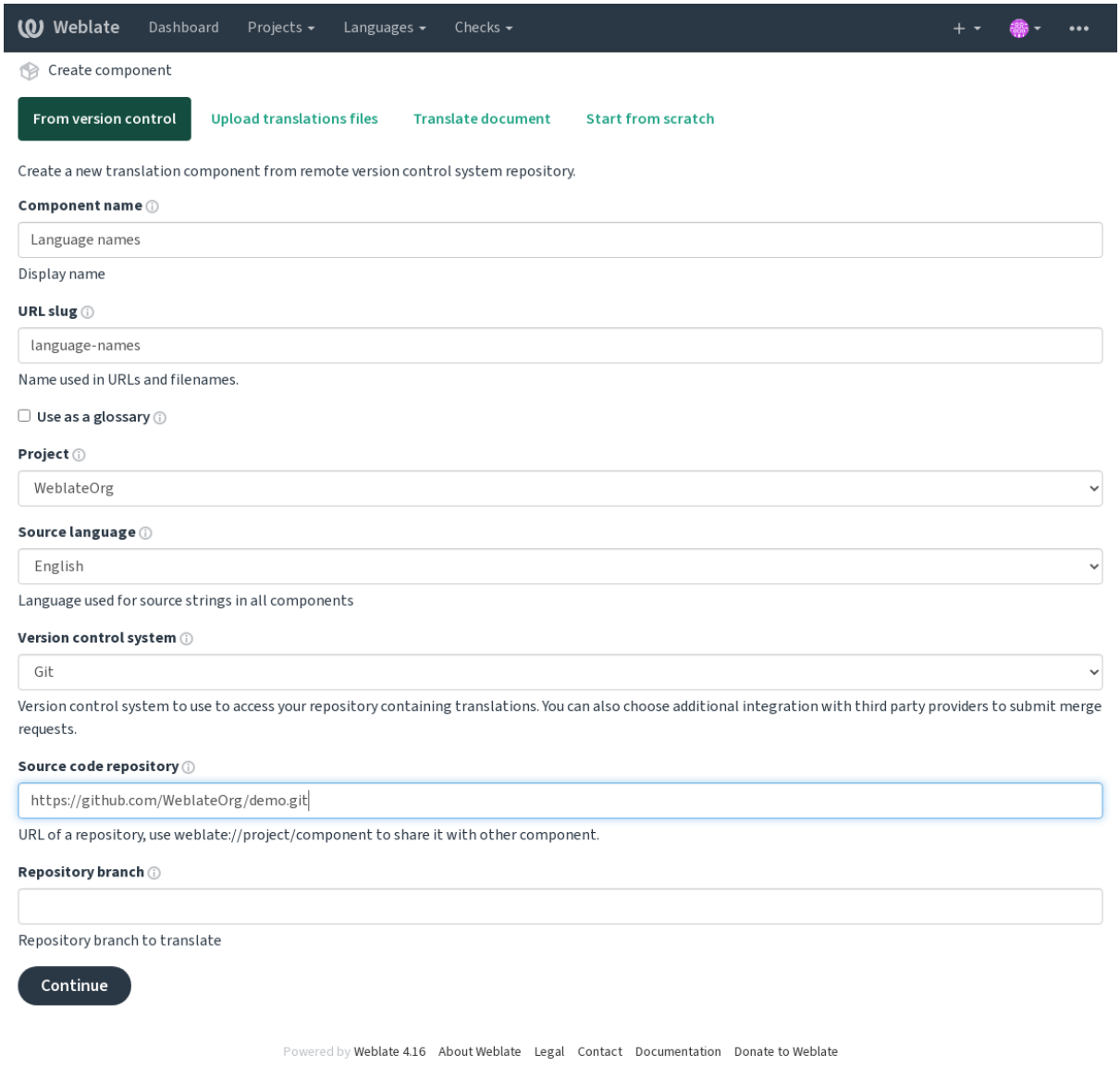
上传翻译文件
如果您没有版本控制或不想将其与 Weblate 集成，则将翻译文件上传到 Weblate。您以后可以使用网络界面或 *Weblate 的 REST API* 更新内容。

翻译文档
上传单个文档或翻译文件并进行翻译。

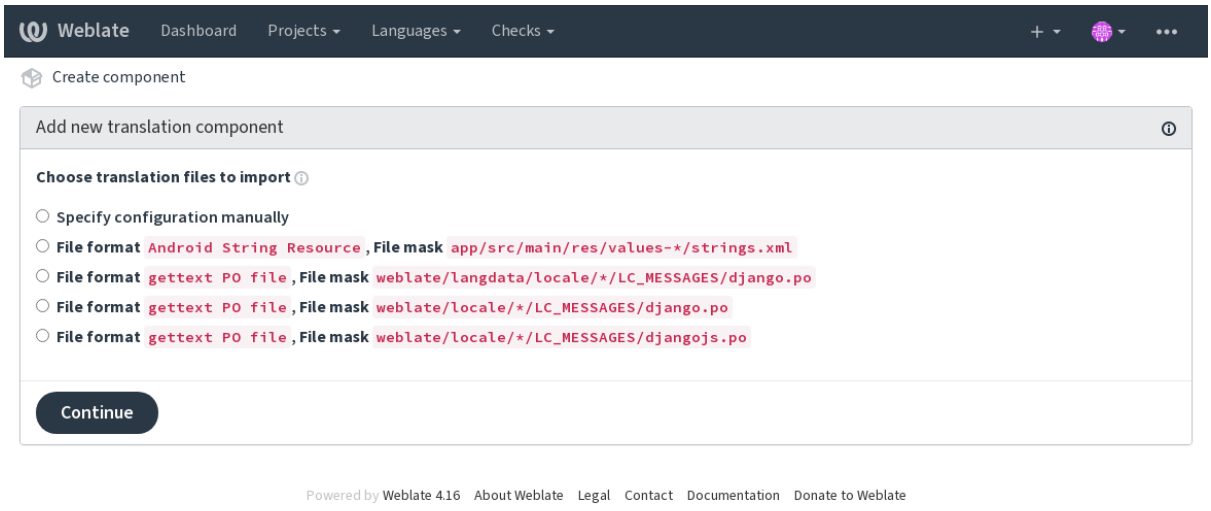
从头开始
创建空白翻译项目并手动添加字符串。

一旦有了现有的翻译部件，就可以使用同一仓库轻松地为其他文件或分支添加新的部件。

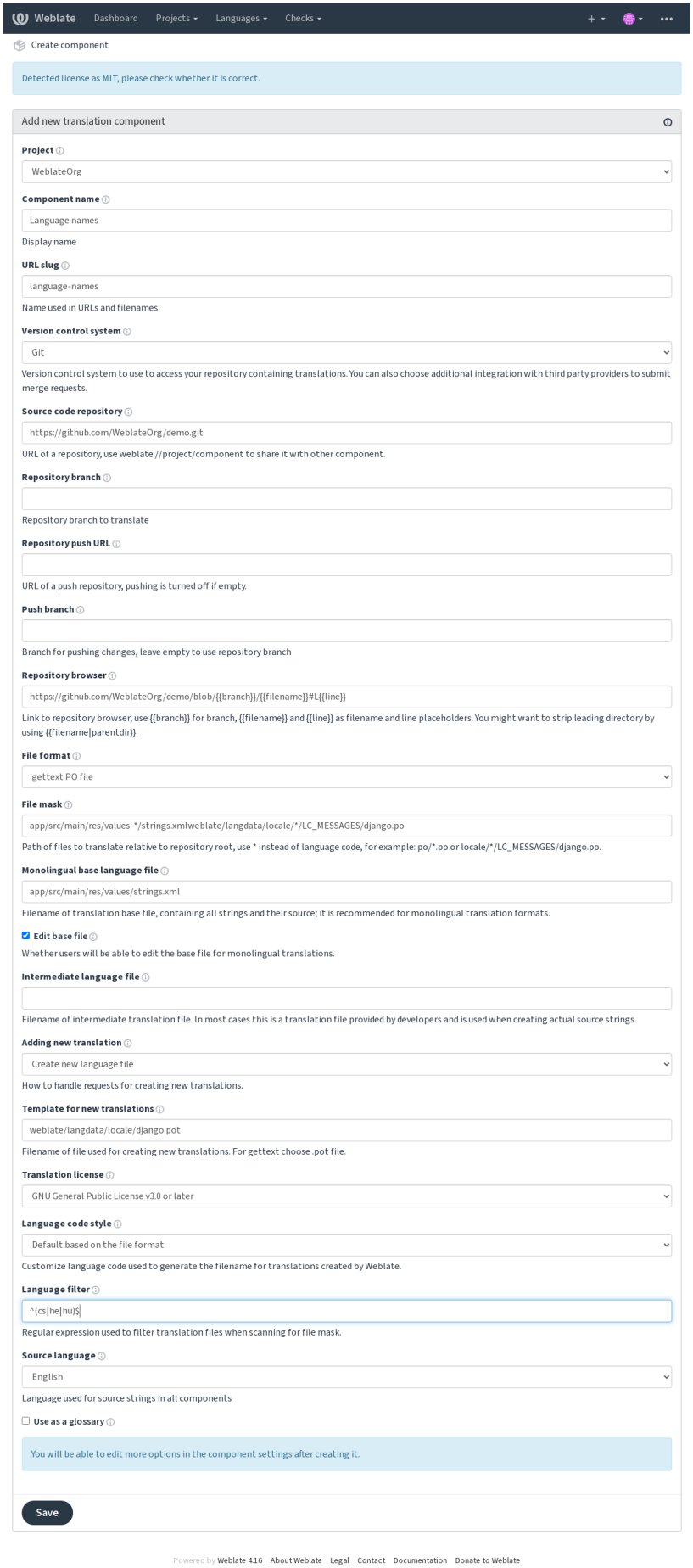
首先，您需要填写名称和仓库位置：



在下一页上，将显示已发现的可翻译资源的列表：



最后，您检查翻译部件信息并填写可选详细信息：


Weblate
Dashboard
Projects
Languages
Checks
Create component
Detected license as MIT, please check whether it is correct.
Add new translation component
Project
WeblateOrg
Component name
Language names
Display name
URL slug
language-names
Name used in URLs and filenames.
Version control system
Git
Version control system to use to access your repository containing translations. You can also choose additional integration with third party providers to submit merge requests.
Source code repository
https://github.com/WeblateOrg/demo.git
URL of a repository, use weblate://project/component to share it with other component.
Repository branch
Repository branch to translate
Repository push URL
URL of a push repository, pushing is turned off if empty.
Push branch
Branch for pushing changes, leave empty to use repository branch
Repository browser
https://github.com/WeblateOrg/demo/blob/{{branch}}/{{filename}}#L{{line}}
Link to repository browser, use {{branch}} for branch, {{filename}} and {{line}} as filename and line placeholders. You might want to strip leading directory by using {{filename|parentdir}}.
File format
gettext PO file
File mask
app/src/main/res/values-*/strings.xmlweblate/langdata/locale/*/LC_MESSAGES/django.po
Path of files to translate relative to repository root, use * instead of language code, for example: po/*.po or locale/*/LC_MESSAGES/django.po.
Monolingual base language file
app/src/main/res/values/strings.xml
Filename of translation base file, containing all strings and their source; it is recommended for monolingual translation formats.
Edit base file
Whether users will be able to edit the base file for monolingual translations.
Intermediate language file
Filename of intermediate translation file. In most cases this is a translation file provided by developers and is used when creating actual source strings.
Adding new translation
Create new language file
How to handle requests for creating new translations.
Template for new translations
weblate/langdata/locale/django.pot
Filename of file used for creating new translations. For gettext choose .pot file.
Translation license
GNU General Public License v3.0 or later
Language code style
Default based on the file format
Customize language code used to generate the filename for translations created by Weblate.
Language filter
^(cs|he|hu)$
Regular expression used to filter translation files when scanning for file mask.
Source language
English
Language used for source strings in all components
Use as a glossary
You will be able to edit more options in the component settings after creating it.
Save
Powered by Weblate 4.16 About Weblate Legal Contact Documentation Donate to Weblate

参见:

Django 管理界面, 项目配置, 部件配置

2.7.3 项目配置

创建一个翻译项目，然后在其中添加一个新的翻译部件。这个项目就像一个架子，里面堆放着真正的翻译。同一项目中的所有部件共享建议及其字典；翻译也将自动传播到单个项目中的所有部件（除非在部件配置中关闭），请参见翻译记忆库。

参见:

/devel/integration

这些基本属性被新建并通知翻译人员项目：

项目名称

详细的项目名称，用于显示项目名称。

参见:

PROJECT_NAME_RESTRICT_RE

URL 标识串

适用于 URL 的项目名称。

项目网站

译者可以在其中找到有关该项目的更多信息的 URL。

这是一个必须的参数，除非通过:setting:‘WEBSITE_REQUIRED‘关闭。

参见:

PROJECT_WEB_RESTRICT_HOST, *PROJECT_WEB_RESTRICT_NUMERIC*, *PROJECT_WEB_RESTRICT_RE*

翻译说明

描述项目中的本地化过程的文本，以及任何其他对翻译人员有用的信息。Markdown 可用于文本格式化或插入链接。

设置“Language-Team”标头

Weblate 是否应该管理 `Language-Team` 标头（目前这是一个*GNU gettext* 独占功能）。

使用共享的翻译记忆库

是否使用共享翻译记忆库，有关更多详细信息，请参见共享翻译记忆库。

可以通过*DEFAULT_SHARED_TM* 更改默认值。

贡献到共享的翻译记忆库

是否贡献到共享翻译记忆库，请参阅共享翻译记忆库 以获取更多详细信息。

可以通过*DEFAULT_SHARED_TM* 更改默认值。

访问控制

配置每个项目的访问控制，请参阅项目访问控制 以获取更多详细信息。

可以通过*DEFAULT_ACCESS_CONTROL* 更改默认值。

启用审校

允许复核翻译的工作流程，请参见专门的审校员。

启用原文审校

允许复核源字符串的工作流程，请参见源字符串复查。

参见：

report-source, 评论

启用钩子

是否将未经身份验证的通知钩子 用于此仓库。

参见：

中间语言文件, 源字符串质量把关, 双语和单语格式, 语言定义

语言别名

将翻译导入到 Weblate 时定义语言代码映射。当您的仓库中的语言代码不一致，并且您希望在 Weblate 中获得一致的视图，或者如果您想使用翻译文件的非标准命名时，可以使用此方法。

典型的使用情况会是将美国英语映射到英语：`en_US:en`

由逗号分隔的多个映射：`en_GB:en,en_US:en`

使用非标准代码：`ia_FOO:ia`

提示： 当匹配翻译文件时映射语言代码，并且映射是大小写敏感的，所以您确保使用与文件名中使用的形式相同的源语言代码。

参见：

添加新的翻译, 语言代码, 分析语言代码

2.7.4 部件配置

部件是用于翻译的内容的分组。您输入版本控制系统（VCS）仓库位置和想要翻译那个文件的掩码，Weblate 会自动地从这个版本控制系统（VCS）中取回，并找到所有匹配的翻译文件。

参见：

/devel/integration

您可以在支持的文件格式 中找到一些典型配置的示例。

备注： 建议将翻译部件保持在合理的规模——将翻译分成对你有意义的部分（单个应用程序或附加组件、书籍章节或网站）。

Weblate 可以轻松处理有 10000 多条字符串的翻译，但对于如此庞大的翻译部件，译者之间的分工和协调更为困难。

如果翻译的语言定义丢失，会新建一个空的定义，并且命名为 “cs_CZ (generated)”。您应该调整定义，并将其反馈给 Weblate 的作者，从而丢失的语言可以包括在下一次的发布版本中。

使用版本控制系统（VCS）工作的所有重要参数都包含在部件中，并且从中取出翻译：

部件名称

冗长部件名称，用于显示部件的名称。

部件标识串

适用于 URLs 的部件名称。

部件项目

部件所属的项目配置。

版本控制系统（VCS）

使用的版本控制系统（VCS），细节请参见：版本控制集成。

参见：

推送 *Weblate* 的更改

源代码仓库

版本控制系统（VCS）仓库，用于拉取更改。

参见：

指定 URLs 的更多细节请参见访问仓库。

提示： 这可以是真实的版本控制系统（VCS）的 URL，也可以是 `weblate://project/component`，表示与其它部件共享该仓库。更多细节请参见 *Weblate* 内部网址。

仓库推送 URL

用于推送的仓库 URL。这个设置用于*Git* 和*Mercurial*，并且当这个空白时推送支持为这些关闭。

对于链接的仓库，这一点不被使用，链接部件的设置适用。

参见：

关于如何指定仓库 URL 的更多细节请见访问仓库，并且关于从 Weblate 推送更改的更多细节，请参见推送 *Weblate* 的更改。

仓库浏览器

用于显示源文件（所用消息的位置）的仓库浏览器的 URL。此处为空时，将不生成这样的链接。你可以使用模板标记。

例如在 GitHub 上，使用类似：`https://github.com/WeblateOrg/hello/blob/{{branch}}/{{filename}}#L{{line}}`

如果你的路径是相对于不同文件夹的（路径包含 `..`），您可能想使用 `parentdir` 过滤器（见模板标记）来剥除前导文件夹：`https://github.com/WeblateOrg/hello/blob/{{branch}}/{{filename|parentdir}}#L{{line}}`

已导出仓库 URL

由 Weblate 进行的更改被导出的 URL。当不使用持续本地化 时，或者当需要手动合并更改时，这是重要的。您可以为 Git 仓库使用*Git* 导出器，来将其自动化。

仓库分支

从版本控制系统（VCS）核实哪个分支，以及从哪里寻找翻译。

对于链接的仓库，这一点不被使用，链接部件的设置适用。

推送分支

用于推送更改的分支，留空则使用仓库分支。

对于链接的仓库，这一点不被使用，链接部件的设置适用。

备注： 此功能目前只支持 Git、GitLab 和 GitHub，不支持其他版本控制系统（VCS）集成。

参见：

推送 *Weblate* 的更改

文件掩码

要翻译的文件的掩码，包括路径。它应包含一个“*”替换语言代码（有关处理方式的信息，请参阅语言定义）。如果您的仓库包含多个翻译文件（例如，多个 gettext 域），则您需要为每个文件创建一个部件。

例如 `po/*.po` 或 `locale/*/LC_MESSAGES/django.po`。

如果文件名包含特殊字符（例如 `[` 或 `]`），则需要将这些特殊字符转义为 `[[]` 或 `[]]`。

参见：

双语和单语格式，“*There are more files for the single language (en)*”（单一语言‘英语’有多个文件）是什么意思？

单语言译文模版语言文件

包含字符串定义的译文模板文件，用于单语言部件。

参见：

双语和单语格式，“*There are more files for the single language (en)*”（单一语言‘英语’有多个文件）是什么意思？

编辑译文模版文件

对于单语言部件 是否允许编辑译文模板文件。

中间语言文件

对于单语言部件 的单一语言文件。在多数情况下，这是开发者提供的翻译文件，并且在新建真正的源字符串时使用。

设置好后，源字符串将基于此文件，但所有其他语言都基于单语言译文模版语言文件。如果字符串没有被翻译成源语言，则禁止翻译成其他语言。这样就提供了源字符串质量把关。

参见：

源字符串质量把关, 双语和单语格式, “*There are more files for the single language (en)*”（单一语言‘英语’有多个文件）是什么意思？

新翻译的翻译模版

用于生成新翻译的翻译模板文件，例如 gettext 的 `.pot` 文件。

提示： 在很多单语言格式中，Weblate 默认以空文件开始。新建新的翻译时，在您想要所有的字符串都以空值出现的情况下来使用。

参见：

adding-translation, 添加新的翻译, 添加新译文, 双语和单语格式, “*There are more files for the single language (en)*”（单一语言‘英语’有多个文件）是什么意思？

文件格式

翻译文件格式，还请参见支持的文件格式。

源字符串缺陷报告地址

用于报告上游缺陷的电子邮箱地址。这个地址还会收到在 Weblate 中作出的任何源字符串评论的通知。

允许同步翻译

您可以关闭项目内从其它部件到这个部件的翻译的传播。这实际上取决于您要翻译的内容，有时最好能多次使用同一个翻译。

对于单语言翻译，除非您跨越整个项目中使用相同的 ID，通常关闭它是个好主意。

可以通过*DEFAULT_TRANSLATION_PROPAGATION* 更改默认值。

参见：

跨部件保持翻译一致

启用建议

对于这个部件，建议的翻译是否被接受。

建议投票

为建议打开投票，请参见建议投票。

自动接受建议

自动接收被投票的建议，请参见建议投票。

翻译标记

质量检查和其他 Weblate 行为的定制，请参见使用标记定制行为。

强制检查

检查哪个不能被忽视的列表，请参见强制检查。

备注： 强制执行检查并不能自动启用它，你仍然应该使用使用标记定制行为 在翻译标记 或源字符串另外的信息 中启用它。

翻译许可证

翻译的许可（不需要与源代码的许可证相同）。

贡献者协议

翻译此部件前需先同意的用户协议。

添加新译文

如何处理创建新语言的请求。可用选项：

联系维护者
: 用户可以选择所需的语言，项目维护者将收到相关通知。由项目维护者来决定是否将该语言添加到仓库中。

指向翻译指引 URL
: 用户会看到一个链接，该链接指向的页面描述了开始新翻译的过程。如果需要更正式的过程（例如在开始实际翻译之前组建一个团队），请使用此选项。

创建新语言文件
: 用户可以选择语言，Weblate 会自动为其创建文件，然后就可以开始翻译了。

禁用添加新翻译
: 用户将无法开始新的翻译。

提示： 项目管理员可以添加新的翻译，即使这里是禁用的，只要有可能（要么:新翻译的翻译模版，要么文件格式支持从一个空文件开始）。

参见：

adding-translation, 添加新的翻译

管理字符串

在 4.5 版本加入.

配置 Weblate 中的用户是否被允许添加新字符串和删除现有字符串。调整这一点以配合你的本地化工作流程–新字符串应该如何被引入。

对于双语格式，字符串通常是从源代码中提取的（例如通过使用 **xgettext**），在 Weblate 中添加新的字符串应该被禁用（下次更新翻译文件时，它们会被丢弃）。在 Weblate 中，你可以为每个翻译管理字符串，但它并不强制要求所有翻译中的字符串都是一致的。

对于单语格式，字符串只在源语言上管理，并在翻译中自动添加或删除。一旦翻译完毕，这些字符串就会出现在翻译文件中。

参见：

双语和单语格式, adding-new-strings, *POST /api/translations/(string:project)/(string:component)/(string:language)/units/*

语言代码风格

自定义 Weblate 创建的用于生成翻译文件名的语言代码。

参见：

添加新的翻译, 语言代码, 分析语言代码

合并方式

你可以配置如何处理来自上游仓库的更新。实际部署取决于版本控制系统，见版本控制集成。

变基
: 在更新时，将 Weblate 的提交重新放在上游仓库之上。这提供了干净的历史，没有额外的合并提交。

  在复杂融合的情况下，变基可能使你产生麻烦，因此请仔细考虑是否允许它们。

  你可能需要通过选择:ref:‘vcs-git-force-push‘作为:ref:‘component-vcs‘来启用强制推送，特别是在推送到不同的分支时。

合并
: 上游仓库更改合并到 Weblate 之一。此设置尽可能使用快进。这是最安全的方式，但可能会产生大量合并提交。

不快进合并
: 上游仓库的修改会被合并到 Weblate 仓库中，每次都要做一次合并提交（即使是在可以快进的时候）。每一个 Weblate 的修改都会在 Weblate 仓库中显示为合并提交。

可以通过*DEFAULT_MERGE_STYLE* 更改默认值。

提交、添加、删除、合并、附加组件及合并请求说明

在提交翻译时使用的说明，请参见模板标记。

可以通过*DEFAULT_ADD_MESSAGE*、*DEFAULT_ADDON_MESSAGE*、*DEFAULT_COMMIT_MESSAGE*、*DEFAULT_DELETE_MESSAGE*、*DEFAULT_MERGE_MESSAGE* 、*DEFAULT_PULL_MESSAGE* 更改默认值。

提交时推送

是否提交更改应该被自动推送到上游仓库。当允许时，一旦 Weblate 将更改提交给基础仓库，推动就被启动（请参见惰性提交）。为了真正允许推送，还要配置 *Repository push URL*。

对更改进行提交的延时时间

设置在后台任务或*commit_pending* 管理命令提交更改前，这些更改存在的时长（以小时为单位）。一旦存在至少一个比该时长更旧的更改，便会提交部件中的所有更改。

可以通过*COMMIT_PENDING_HOURS* 更改默认值。

提示： 还有一些情况下，待定的修改可能会被提交，见惰性提交。

出错时锁定

在第一次推送或合并到上游仓库失败后，锁定该部件（及其链接部件，参见*Weblate 内部网址*），或从上游仓库拉出。这样可以避免增加必须手动解决的冲突。

一旦仓库没有故障留下来了，部件将会自动解锁。

源语言

用于源字符串的语言。如果您要翻译的不是英语，请更改此选项。

提示： 如果你正在从英语翻译双语文件，但又希望能够在英语翻译中进行修复，选择 *English (Developer)* 作为一种源语言以避免源语言和现有翻译之间名称上的冲突。

对于单语言翻译，您可以使用这种情况下的中间翻译，请参见中间语言文件。

语言筛选

当扫描文件掩码时用于将翻译过滤的正则表达式。它可以用于限制 Weblate 管理的语言列表。

备注： 单出现在文件名中时，您需要列出语言代码。

过滤的一些示例：

过滤器的描述	正则表达式
只有选择的语言	`^(cs|de|es)$`
排除语言	`^(?!(it|fr)$).+$`
只筛选两个字母的代码	`^..$`
排除非语言文件	`^(?!(blank)$).+$`
包括所有文件（默认）	`^[^.]+$`

正则表达式变体

用于确定字符串变体的正则表达式，请见 variants。

备注： 多数字段可以由项目所有者或管理员在 Weblate 界面上编辑。

参见：

Weblate 支持 Git 和 Mercurial 以外的其它版本控制系统（VCS）吗？, alerts

优先级

高优先级的部件将最先提供给译者。

在 4.15 版本发生变更: 这同样影响匹配的术语表术语的顺序。

受限制的访问

部件默认对访问项目的任何人都可见，即使不能在部件中进行任何更改。这会容易地使翻译在项目内保持一致。

在部件或部件列表级别限制访问，会接管对部件的访问权限，而不考虑项目层面的权限。你必须明确地授予它访问权。这可以通过授予一个新的用户组并将用户放入其中，或者使用默认的 自定义或 私有访问控制组来实现。

默认设置可在*DEFAULT_RESTRICTED_COMPONENT* 中更改。

提示： 这也应用于项目管理员—请确认切换状态后，您不会丢失对部件的访问。

在项目中分享

可以选择部件可见的其他项目。这在分享不同项目间使用的库时是有用的。

备注： 分享部件不更改其访问控制。这样做只是让它在浏览其它项目时可见。用户仍然需要访问实际部件的权限来浏览或翻译它。

用作术语表

在 4.5 版本加入.

允许将此部件用作术语表。您可以用术语表颜色 配置它的列出方式。

术语表将可以在所有由:ref:‘component-links‘定义的项目中访问。

建议在术语表上启用管理字符串，以允许添加新词。

参见：

术语表

术语表颜色

显示术语表的颜色，在显示单词匹配时使用。

2.7.5 模板标记

Weblate 在几个需要进行文本渲染的地方使用了简单的标记语言。它基于 Django 模板语言，所以它可以非常强大。

当前它用在：

- 提交说明格式，请参见部件配置
- **几个附加组件**
  - 部件发现
  - 统计数据生成器
  - 从附加组件执行脚本

在部件模板中，有以下变量可用：

{{ language_code }}
语言代码

{{ language_name }}
语言名称

{{ component_name }}
部件名称

{{ component_slug }}
部件标识串

{{ project_name }}
项目名称

{{ project_slug }}
项目标识串

{{ url }}
翻译 URL

{{ filename }}
翻译文件名

{{ stats }}
翻译统计数据，这具有进一步的属性，示例如下。

{{ stats.all }}
字符串总量计数

{{ stats.fuzzy }}
需要复查的字符串计数

{{ stats.fuzzy_percent }}
需要复查的字符串百分比

{{ stats.translated }}
已翻译的字符串计数

{{ stats.translated_percent }}
已翻译的字符串百分比

{{ stats.allchecks }}
带有未通过检查的字符串数量

{{ stats.allchecks_percent }}
带有未通过检查的字符串百分比

{{ author }}
当前提交的作者，只在提交范围可用。

{{ addon_name }}
当前执行的附加组件名称，旨在附加组件提交说明中可用。

后面的变量在仓库浏览器或编辑器模板中可用：

{{branch}}
当前的分支

{{line}}
文件的行数

{{filename}}
文件名，您也可以使用 `parentdir` 过滤器，例如 `{{filename|parentdir}}`，来剥除前导部分

您可以将它们与过滤器结合：

```
{{ component|title }}
```

您可以使用条件：

```
{% if stats.translated_percent > 80 %}Well translated!{% endif %}
```

有另外的标签用于替换字符：

```
{% replace component "-" " " %}
```

您可以将它与过滤器结合：

```
{% replace component|capfirst "-" " " %}
```

还有另外的过滤器来操作文件名：

```
Directory of a file: {{ filename|dirname }}
File without extension: {{ filename|stripext }}
File in parent dir: {{ filename|parentdir }}
It can be used multiple times:  {{ filename|parentdir|parentdir }}
```

……以及其他 Django 模板特性。

2.7.6 导入速度

从版本控制系统（VCS）仓库获取并将翻译导入 Weblate 可能会是漫长的过程，具体取决于您的翻译的大小。以下是一些提示：

优化配置

默认配置对于测试和调试 Weblate 很有用。而对于生产设置，您应该做一些调整。许多调整对性能有相当大的影响。请查阅生产设置 了解详情，特别是：

- 配置 Celery 来执行后台任务（请参见使用 *Celery* 的后台任务）
- 允许缓存
- 使用强力的数据库引擎
- 禁止调试模式

检查资源的限制

如果导入巨大的翻译或仓库，您会遭到服务器资源限制的打击。

- 检查空闲内存的量，通过操作系统来缓存翻译，将极大地提高性能。
- 如果有很多字符串需要处理的话，磁盘操作会是瓶颈——磁盘被 Weblate 和数据库施加压力。
- 另外的 CPU 核心会帮助提高后台任务的性能（请参见使用 *Celery* 的后台任务）。

禁止不必要的检查

一些质量检查可能相当昂贵，如果不需要，在导入时省略可以节省一些时间。配置的信息请参见*CHECK_LIST*。

2.7.7 自动新建部件

在您的项目有十多个翻译文件的情况下（例如不同的 gettext 域，或 Android 应用的一部分），您会想要将它们自动导入。可以通过使用*import_project* 或*import_json*，或者通过安装部件发现 附加组件，通过命令行来实现。

要使用附加组件，首先您需要为一个翻译文件（选择未来最不可能改名或删除的那个）创建一个部件，然后在该部件上安装附加组件。

对于管理命令，您需要新建包含所有部件的项目，然后运行*import_project* 或*import_json*。

参见：

管理命令, 部件发现

2.8 语言定义

为了恰当地呈现不同的翻译，需要提供有关语言名称、文本方向、复数定义和语言代码的信息。

2.8.1 内置语言定义

Weblate 中包含了大约 600 种语言的定义，而且该语言列表在每个发行版中都会进行扩充。每当升级 Weblate 时（更确切地说，是每当执行 **weblate migrate** 时，请参见一般的升级指示）语言数据库都会更新，以包含 Weblate 提供的所有语言定义。

这个特性可以使用 *UPDATE_LANGUAGES* 来禁止。还可以使用 `setuplang` 来强制更新数据库，从而匹配 Weblate 内建数据。

参见：

扩展内置语言定义, 当前语言定义

2.8.2 分析语言代码

解析翻译时，Weblate 试图从文件掩码 将语言代码（通常为 ISO 639-1）映射到任何现有的语言对象。

您可以通过语言别名 在项目层次来进一步调整这种映射。

如果无法找到精确的匹配，将尝试把其融入一种现有的语言。已尝试以下措施：

- 大小写不敏感的查询。
- 将下划线和破折号标准化。
- 查询内置的语言别名。
- 按语言名称查找。
- 忽略给定语言的默认国家地区代码——选择 `cs` 而非 `cs_CZ`。

如果这也失败了，将使用默认值（从左到右的文本方向，一个复数）创建一个新的语言定义。自动创建的代码为 `xx_XX` 的语言将被命名为 *xx_XX (generated)*。您可能想稍后在管理界面中更改这个（见更改语言定义），并将它报告给问题跟踪器（见为 *Weblate* 作贡献），这样的话，正确的定义就可以添加到即将发布的 Weblate 版本中。

提示： 在您看到有些不想要的内容作为语言的情况下，您会想要调整语言筛选，当分析翻译时忽略这样的文件。

参见：

语言代码, 添加新的翻译

2.8.3 更改语言定义

您可以在语言界面来更改语言定义（`/languages/` URL ）。

当编辑时，确认所有字段都是正确的（特别是复数和正文方向），否则译者将不能正常编辑这些翻译。

2.8.4 歧义语言代码和宏语言

在很多情况下，最好不要为翻译使用宏语言代码。可能会出问题的典型示例是库尔德语，它可以用阿拉伯文或拉丁字母书写，这取决于实际的变体。为了让 Weblate 可以正确地运行，建议只使用单独的语言代码，避免使用宏语言代码。

参见：

宏语言定义，宏语言列表

2.8.5 语言定义

每种语言都包括后面的字段：

语言代码

识别语言的代码。Weblate 使用两个字母代码，如 ISO 639-1 所定义的，但对于没有两个字母代码的语言，使用会使用 ISO 639-2 或 ISO 639-3 代码。它还支持 BCP 47 定义的扩展代码。

参见：

分析语言代码, 添加新的翻译

语言名称

语言的可见名称。还要根据用户界面语言将 Weblate 中包括的语言名称进行本地化。

文字方向

确定语言是从右向左还是从左向右书写。对于大多数语言来说，此属性都能正确地自动检测出来。

复数数量

语言中使用的复数数量。

复数公式

Gettxt 兼容的复数公式，用于确定给定数量使用哪种复数形式。

参见：

复数形式, GNU gettext 工具: 复数形式, Language Plural Rules by the Unicode Consortium

语言使用者数量

全世界说这种语言的人的数目。

2.8.6 添加新的翻译

在 2.18 版本发生变更: 在 2.18 以前的版本中，添加新的翻译的行为因文件格式而不同。

Weblate 可以为所有文件格式自动开始新的翻译。

一些格式希望以一个空文件开始，并只包含已翻译的字符串（如*Android 字符串资源*），而另一些格式则希望包含所有 key（如*GNU gettext*）。基于文档的格式（如*OpenDocument 格式*）以源文档的副本开始，所有字符串均被标记为需要编辑。某些情况下，这实际上并不取决于格式，而更取决于用来处理翻译的框架（如使用*JSON 文件*）。

当在部件配置 中指定新翻译的翻译模版 时，Weblate 将使用这个文件开始新的翻译。当执行时任何现有翻译将从文件中删除。

当新翻译的翻译模版 是空的，并且文件格式支持时，新建空文件，一旦新的字符串被翻译就添加进去。

语言代码风格 允许在生成的文件名中将语言代码个性化：

基于文件格式的默认值
: 取决于文件格式，其中大多数文件格式用的是 POSIX 风格。

POSIX 风格，使用下划线作为分隔符
: 通常由 gettext 和相关工具使用，生成像 `pt_BR` 这样的语言代码。

POSIX 风格，使用下划线作为分隔符，包括国家地区代码
: POSIX 风格的语言代码，包含国家地区代码，即使没有必要（例如 `cs_CZ`）。

BCP 风格，使用连字符作为分隔符
: 通常用于网络平台，生成像 `pt-BR` 这样的语言代码。

BCP 风格，使用连字符作为分隔符，包含国家地区代码
: BCP 风格的语言代码，包含国家地区代码，即使没有必要（例如 `cs-CZ`）。

BCP 风格，使用连字符作为分隔符，旧式语言代码
: 对中文和 BCP 样式符号使用旧代码。

BCP 样式，使用连字符作为分隔符，小写
: BCP 样式表示法，全部为小写（例如 `cs-cz` ）。

Apple 应用商店元数据样式
: 适用于将元数据上传到 Apple 应用商店的样式。

Google Play 元数据样式
: 适用于将元数据上传到 Google Play 应用商店的样式。

Android 风格
: 只在 Android 应用中使用，生成 `pt-rBR` 这样的语言代码。

Linux 风格
: Linux 使用的语言环境，使用中文和 POSIX 样式符号的遗留代码。

此外，语言别名 中定义的任何映射都反向应用。

备注： Weblate 在解析翻译文件时能识别其中的任何一种，上面的设置只影响新文件的创建方式。

参见：

语言代码, 语言别名, 分析语言代码

2.9 持续本地化

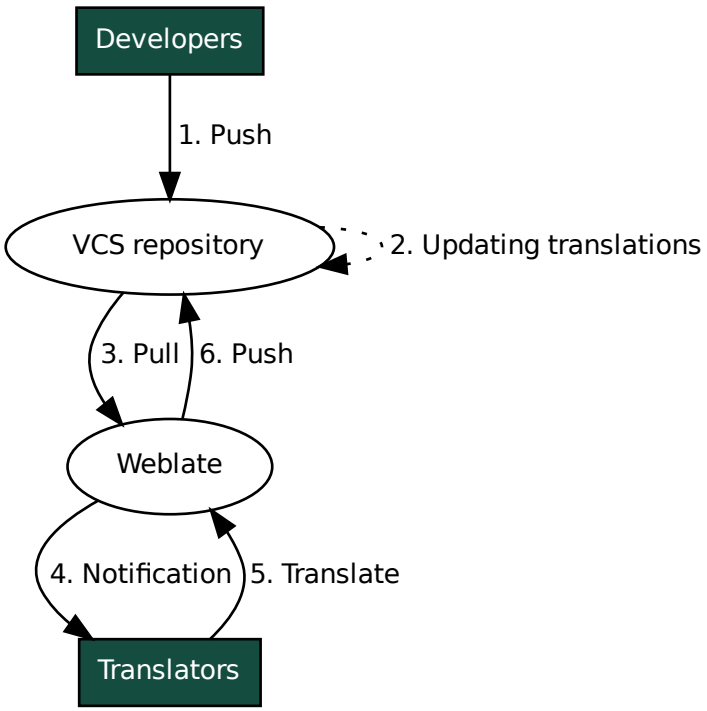
有适当的基础结构，因此你的翻译紧随开发。这样，翻译人员可以一直进行翻译，而不必在发布之前处理大量的新文本。

参见:

/devel/integration 描述了将您的开发集成到 Weblate 中的基本方式。

这是过程：

1. 开发人员进行更改并将其推送到版本控制系统（VCS）仓库。
2. 可以选择更新翻译文件（这取决于文件格式，请参阅*当已经更新了模板时，为什么 Weblate 仍然显示旧的字符串？*）。
3. Weblate 从版本控制系统（VCS）仓库中拉取更改，请参阅*更新仓库*。
4. 一旦 Weblate 检测到翻译更改，便会根据译者的订阅设置通知他们。
5. 译者使用 Weblate Web 界面提交翻译，或上传离线更改。
6. 译者完成后，Weblate 会将更改提交到本地仓库（请参阅*惰性提交*），如果有权限将其推回（请参阅*推送 Weblate 的更改*）。



2.9.1 更新仓库

你应该设置某种方式，从后端仓库的源头更新它们。

- 使用通知钩子 来与多数常见的代码托管服务集成：
  - 从 *GitHub* 自动接收更改
  - 从 *GitLab* 自动接收更改
  - 从 *Bitbucket* 自动接收更改
  - 从 *Pagure* 自动接受更改
  - 从 *Azure Repos* 自动接收更改
  - 从 *Gitea Repos* 自动接收更改
- 在仓库管理中或使用 *Weblate 的 REST API* 或 *Weblate 客户端* 来手动触发更新
- 启用 *AUTO_UPDATE* 以自动更新你的 Weblate 实例上的所有部件
- 执行 *updategit*（选择项目，或 `--all` 来更新全部）

每当 Weblate 更新仓库时，更新后附加组件都将被触发，请参见：附加组件。

避免合并冲突

当同一文件在 Weblate 内外都被更改时，就会出现来自 Weblate 的合并冲突。有两种方法可以解决这个问题——避免在 Weblate 之外进行编辑，或者将 Weblate 整合到您的更新过程中，这样就可以在更新 Weblate 外部的文件之前刷新更改。

对于单语文件来说，第一种方法很简单——您可以在 Weblate 中添加新字符串，并将文件的整个编辑留在那里。对于双语文件，通常存在某种信息提取过程可以从源代码生成可翻译文件。在一些情况下，这可以分成两部分——一部分是提取生成模板（例如使用 **xgettext** 生成 gettex POT ），然后下一步处理将其合并到真正的翻译中（例如使用 **msgmerge** 更新 gettext PO 文件）。您可以在 Weblate 中执行第二步，它将确保在此操作之前包含所有的待处理更改。

第二种方法是使用 *Weblate 的 REST API* 强制 Weblate 推送所有待处理的更改，并在您进行更改时锁定翻译。

进行更新的脚本看起来像这样：

```
# Lock Weblate translation
wlc lock
# Push changes from Weblate to upstream repository
wlc push
# Pull changes from upstream repository to your local copy
git pull
# Update translation files, this example is for Django
./manage.py makemessages --keep-pot -a
git commit -m 'Locale updates' -- locale
# Push changes to upstream repository
git push
# Tell Weblate to pull changes (not needed if Weblate follows your repo
# automatically)
wlc pull
# Unlock translations
wlc unlock
```

如果多个部件分享相同的仓库，需要分别将他们全部锁定：

```
wlc lock foo/bar
wlc lock foo/baz
wlc lock foo/baj
```

备注： 示例使用了*Weblate 客户端*，这需要配置（API 密钥）来远程控制 Weblate。可以通过使用 HTTP 客户端代替 wlc 来实现这一点，例如 curl，请参见*Weblate 的 REST API*。

参见：

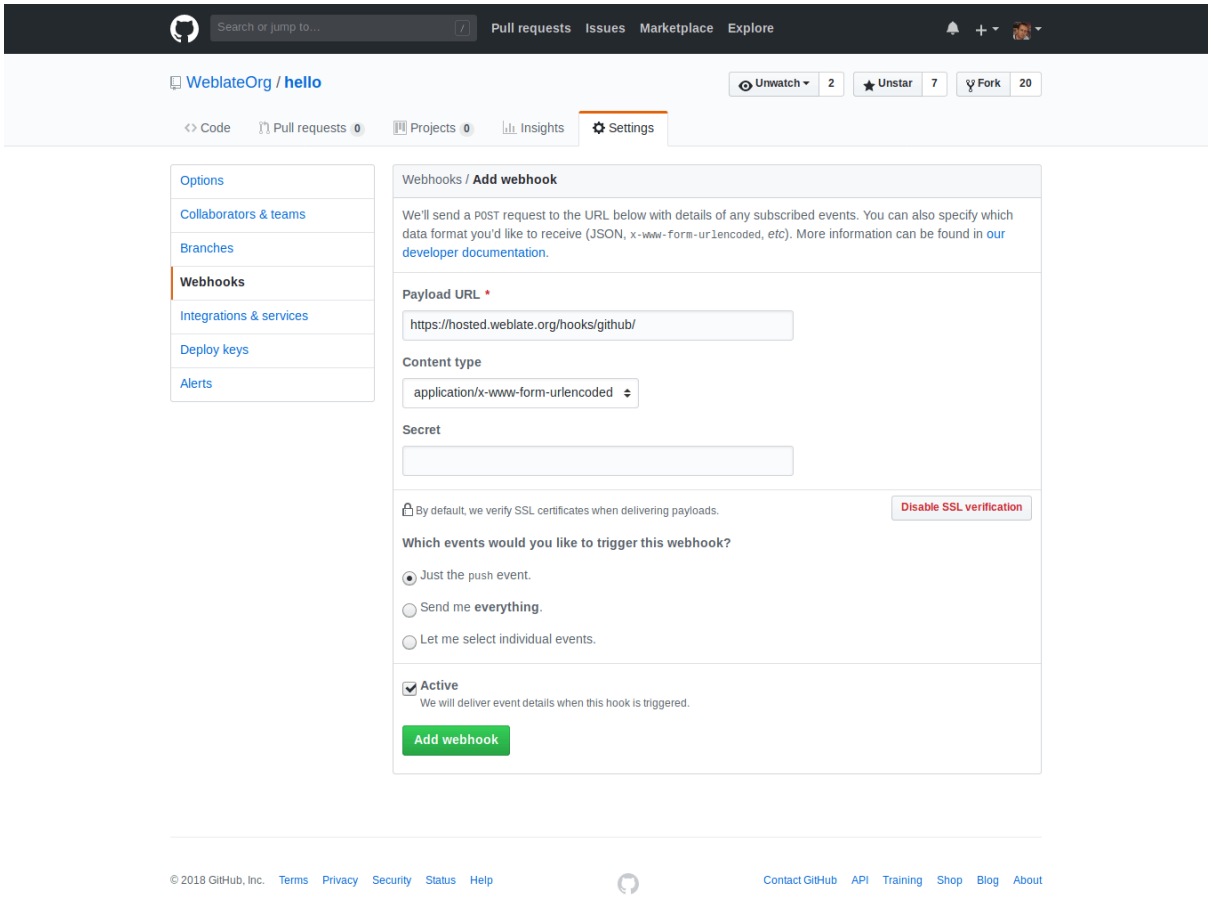
Weblate 客户端

从 GitHub 自动接收更改

Weblate 自带对 GitHub 的原生支持。

如果使用 Hosted Weblate，推荐的方法是安装 Weblate app，该方法能够得到正确的设置，而不必设置很多东西。它还可以用于将更改推送回来。

为了在每次推送到 GitHub 仓库时接收通知，将 Weblate Webhook 添加到仓库设置（*Webhooks*）中，如下图所示：



对于负载 URL，将 `/hooks/github/` 增补到你的 Weblate URL 中，例如对于 Hosted Weblate 服务，这是 `https://hosted.weblate.org/hooks/github/`。

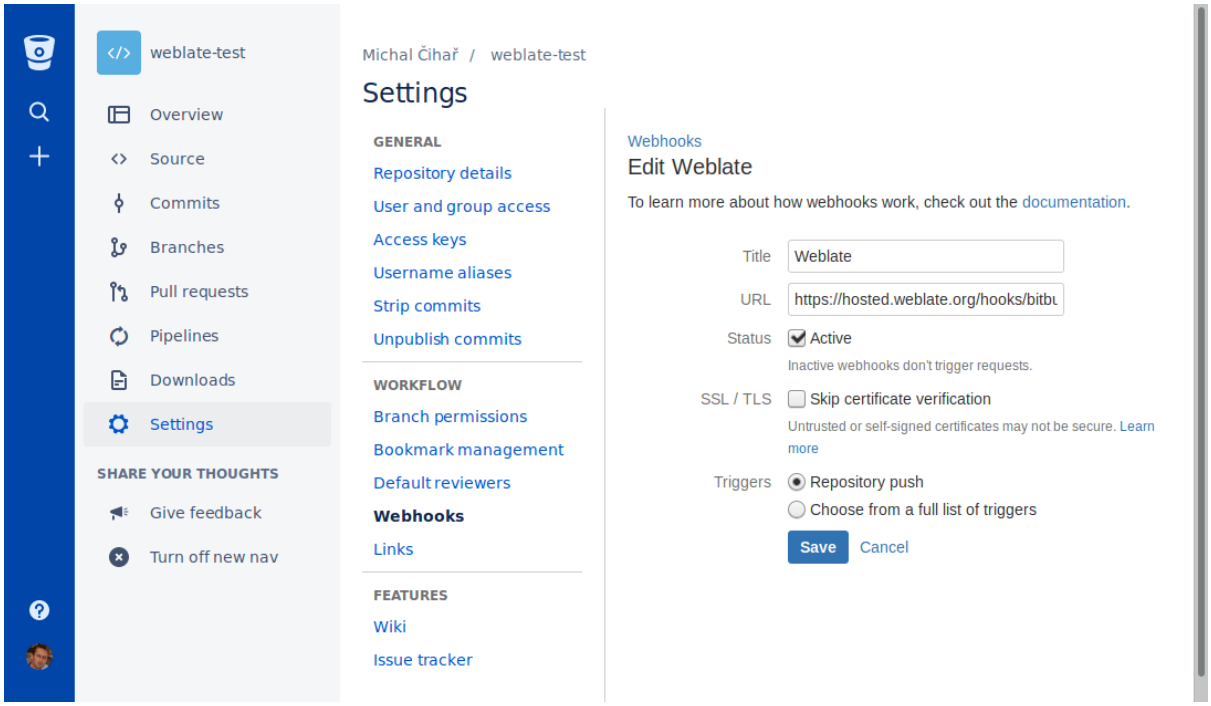
可以将其他设置保留为默认值（Weblate 可以处理内容类型，并只消费 *push* 事件）。

参见：

`POST /hooks/github/`, 从 Hosted Weblate 访问仓库

从 Bitbucket 自动接收更改

Weblate 已经支持 Bitbucket webhooks，添加仓库推送时触发的 webhook，目的地为你的 Weblate 安装上的 `/hooks/bitbucket/` （例如 `https://hosted.weblate.org/hooks/bitbucket/` ）。



参见：

`POST /hooks/bitbucket/`, 从 *Hosted Weblate* 访问仓库

从 GitLab 自动接收更改

Weblate 已经支持 GitLab hooks，添加项目的 webhook，目的地为你的 Weblate 安装上的 `/hooks/gitlab/` （例如 `https://hosted.weblate.org/hooks/gitlab/` ）。

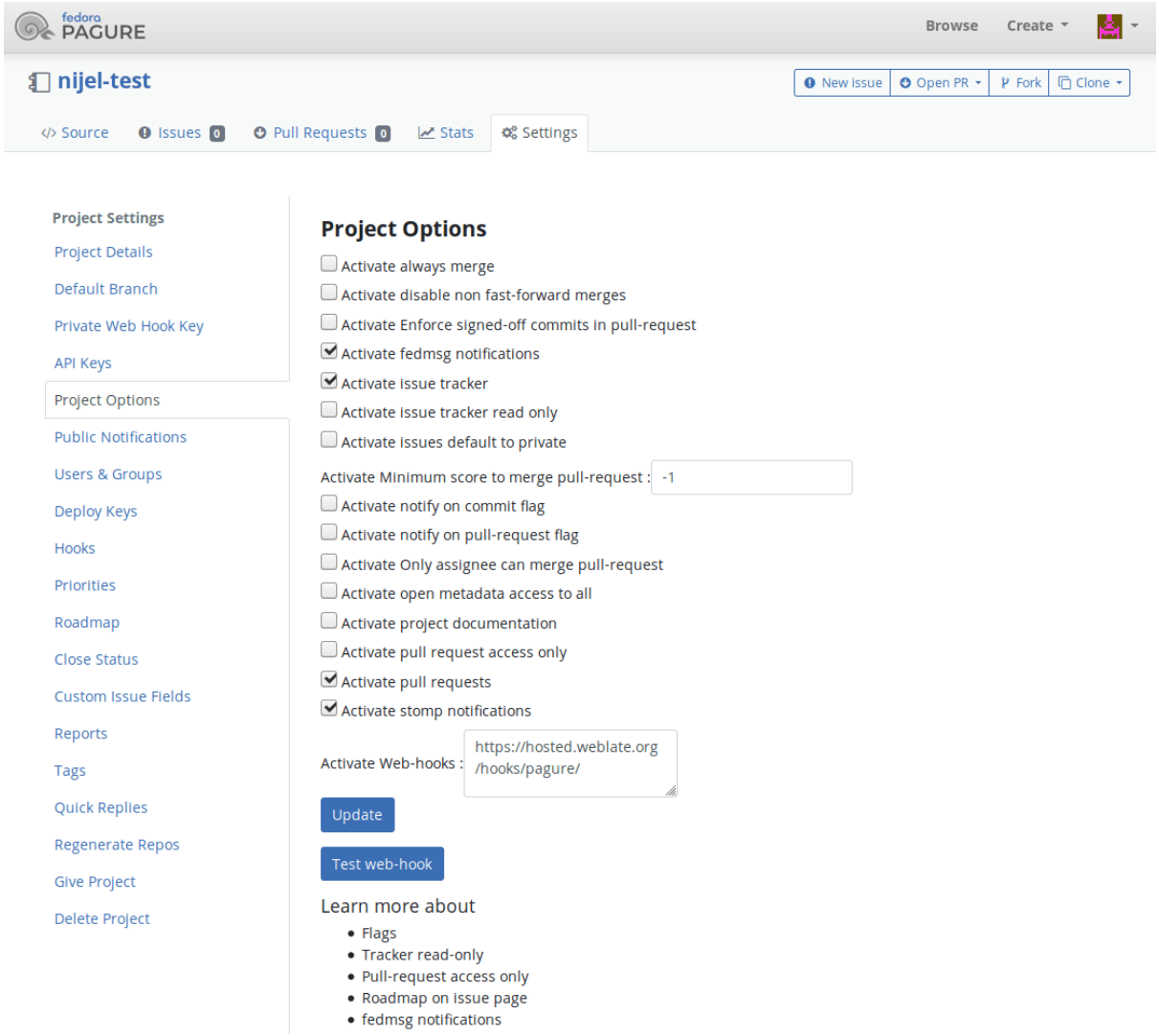
参见：

`POST /hooks/gitlab/`, 从 *Hosted Weblate* 访问仓库

从 Pagure 自动接受更改

在 3.3 版本加入.

Weblate 已经支持 Pagure hooks，添加项目的 webhook，目的地为你的 Weblate 安装上的 `/hooks/pagure/` （例如 `https://hosted.weblate.org/hooks/pagure/` ）。这可以在 *Project options* 之下的 *Activate Web-hooks* 中完成：



参见:

POST /hooks/pagure/, 从 *Hosted Weblate* 访问仓库

从 Azure Repos 自动接收更改

在 3.8 版本加入.

Weblate 已经支持 Azure Repos web hooks，为 *Code pushed* 事件添加 webhook，目的地为你的 Weblate 安装上的 `/hooks/azure/` URL （例如 `https://hosted.weblate.org/hooks/azure/` ）。这可以在 *Project settings* 之下的 *Service hooks* 中完成。

参见:

Azure DevOps 手册中的 Web hooks, *POST /hooks/azure/*, 从 *Hosted Weblate* 访问仓库

从 Gitea Repos 自动接收更改

在 3.9 版本加入.

Weblate 已经支持 Gitea webhooks，为 *Push events* 事件添加 *Gitea Webhook*，目的地为你的 Weblate 安装上的 `/hooks/gitea/` URL（例如 `https://hosted.weblate.org/hooks/gitea/` ）。这可以在 *Settings* 之下的 *Webhooks* 中完成。

参见:

Gitea 手册中的 Webhooks, *POST /hooks/gitea/*, 从 *Hosted Weblate* 访问仓库

从 Gitee Repos 自动接收更改

在 3.9 版本加入.

Weblate 已经支持 Gitee webhooks，为 *Push* 事件添加 *Webhook*，目的地为你的 Weblate 安装上的 `/hooks/gitee/` URL （例如 `https://hosted.weblate.org/hooks/gitee/` ）。这可以在 *Management* 之下的 *Webhooks* 中完成。

参见:

Gitee 手册中的 Webhoks , *POST /hooks/gitee/*, 从 *Hosted Weblate* 访问仓库

每晚自动更新仓库

Weblate 在后面合并更改时，每晚自动获取远程仓库来提高性能。可以选择将其同样转换为进行每晚合并，通过允许*AUTO_UPDATE*。

2.9.2 推送 Weblate 的更改

每个翻译部件可以新建推送 URL（请参见*仓库推送 URL* ），在那种情况下 Weblate 能够将更改推送到远程仓库。Weblate 还可以配置在每次提交时自动推送更改（这是默认的，请参见*提交时推送* ）。如果不想更改自动给推送，可以在 *仓库维护*之下手动进行，或通过*wlc push* 使用 API。

推送选项根据使用的*版本控制集成* 而不同，更多细节可以在那个章节中找到。

如果不想要 Weblate 的直接推送，系统也支持*GitHub 拉取请求* 、*GitLab 合并请求*、*Gitea 拉取请求*、*Pagure 合并请求* 的拉取请求或*Gerrit* 审核。你可以通过在*部件配置* 中选择 *GitHub* 、*GitLab Gitea*、*Gerrit* 或 *Pagure* 作为*版本控制系统（VCS）* 来激活这些。

整体上，Git 、GitHub 和 GitLab 可以具有后面的选项：

需要的设置	版本控制系统（VCS）	仓库推送 URL	推送分支
不推送	*Git*	空	空
直接推送	*Git*	SSH URL	空
推送到单独的分支	*Git*	SSH URL	分支名称
来自派生的 GitHub 拉取请求	*GitHub 拉取请求*	空	空
来自分支的 GitHub 拉取请求	*GitHub 拉取请求*	SSH URL[1]	分支名称
来自派生的 GitLab 合并请求	*GitLab 合并请求*	空	空
来自分支的 GitLab 结合请求	*GitLab 合并请求*	SSH URL[1]	分支名称
来自分叉的 Gitea 合并请求	*Gitea 拉取请求*	空	空
来自分支的 Gitea 合并请求	*Gitea 拉取请求*	SSH URL[1]	分支名称
来自派生的 Pargue 合并请求	*Pagure 合并请求*	空	空
来自分支的 Pagure 合并请求	*Pagure 合并请求*	SSH URL[1]	分支名称

[1] 在*源代码仓库* 支持推送的情况下可以为空。

备注： 还可以允许 Weblate 提交后更改的自动推送，这可以在提交时推送 中完成。

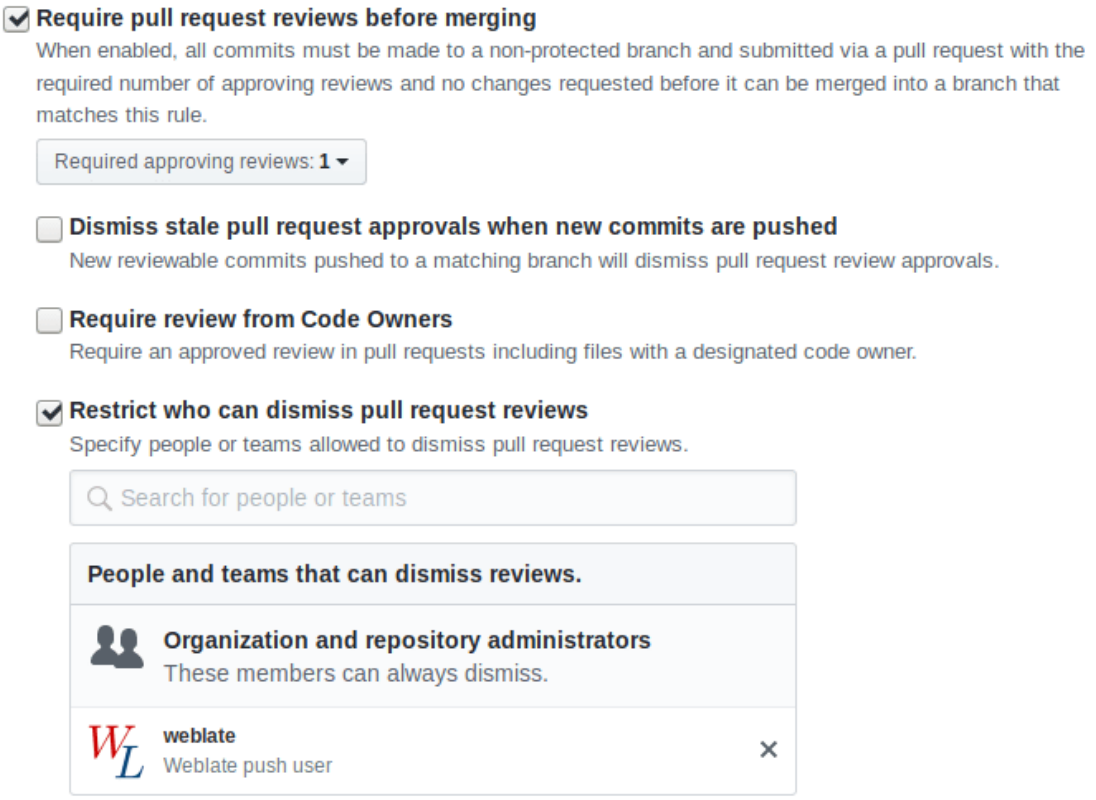
参见：

请参见访问仓库 来设置 SSH 密钥，和惰性提交 获得关于 Weblate 决定提交更改的信息。

受保护的分支

如果在受保护的分支上使用 Weblate，可以配置使用拉取请求，并执行翻译的实际复查（对你不知道的语言可能有问题）。另一个方法是去掉对 Weblate 推送用户的这个限制。

例如在 GitHub，这可以在仓库配置中进行：



2.9.3 与他人交互

Weblate 通过使用它的 API，使与他人的交流更容易。

参见：

Weblate 的 REST API

2.9.4 惰性提交

Weblate 会尽可能将同一作者的提交分组到一个提交中。这大大减少了提交的数量，但是如果你想同步版本控制系统（VCS）仓库，例如合并，你可能需要明确地告诉它去做提交（这对 管理者组是默认允许的，参见权限和内置角色列表）。

一旦后面的任何条件满足，这种模式的更改将被提交：

- 某人另外更改了已经被更改的字符串。
- 来自上游的结合发生了。
- 明确地请求了提交。
- 已请求文件下载。
- 更改比部件配置 上定义为对更改进行提交的延时时间 的时间段更陈旧。

提示： 每个部件都会创建提交。所以，如果你有很多部件，仍然会看到很多提交。在这种情况下，你可以使用挤压 *Git* 提交 附加组件。

如果你想更频繁地提交更改而无需检查存在时间，你可以设置一个定时任务来执行提交。方法是使用*Django* 管理界面 中的 *Periodic Tasks*。首先，创建期望的 *Interval*（比如 120 秒）。接着，添加新的定时任务并选择 `weblate.trans.tasks.commit_pending` 为 *Task*，`{"hours": 0}` 为 *Keyword Arguments* 和想要的间隔。

2.9.5 用脚本处理仓库

定制 Weblate 与仓库交互的方式是附加组件。关于如何通过附加组件执行外部脚本的信息，请咨询从附加组件执行脚本。

2.9.6 跨部件保持翻译一致

一旦具有多个翻译部件，你会想要确保相同的字符串具有相同的翻译。这可以在几个层次实现。

翻译宣传

当允许同步翻译 处于启用状态时（这是默认的，请参见部件配置），在所有的部件中字符串匹配时，所有新的翻译自动进行。在所有的部件中这样的翻译都适当地归功于当前翻译的用户。

备注： 翻译宣传需要密钥来匹配单语言翻译格式，因此在建立翻译密钥匙请记住。

一致性检查

只要字符串存在差异，就会触发不一致的 检查。可以利用此检查来手动复查这些差异，并选择正确的翻译。

自动翻译

基于不同部件的自动翻译，可以是跨部件同步翻译的方式。可以或者手动触发（请参见自动翻译），或者使用附加组件（自动翻译）在仓库更新时自动运行。

2.10 翻译许可

您可以指定翻译输入哪种授权方式。当翻译对公众公开时这特别重要，这样明确规定如何使用。

您应该指定部件配置许可证信息。您应该避免要求与贡献者签订许可协议，尽管这是可能的。

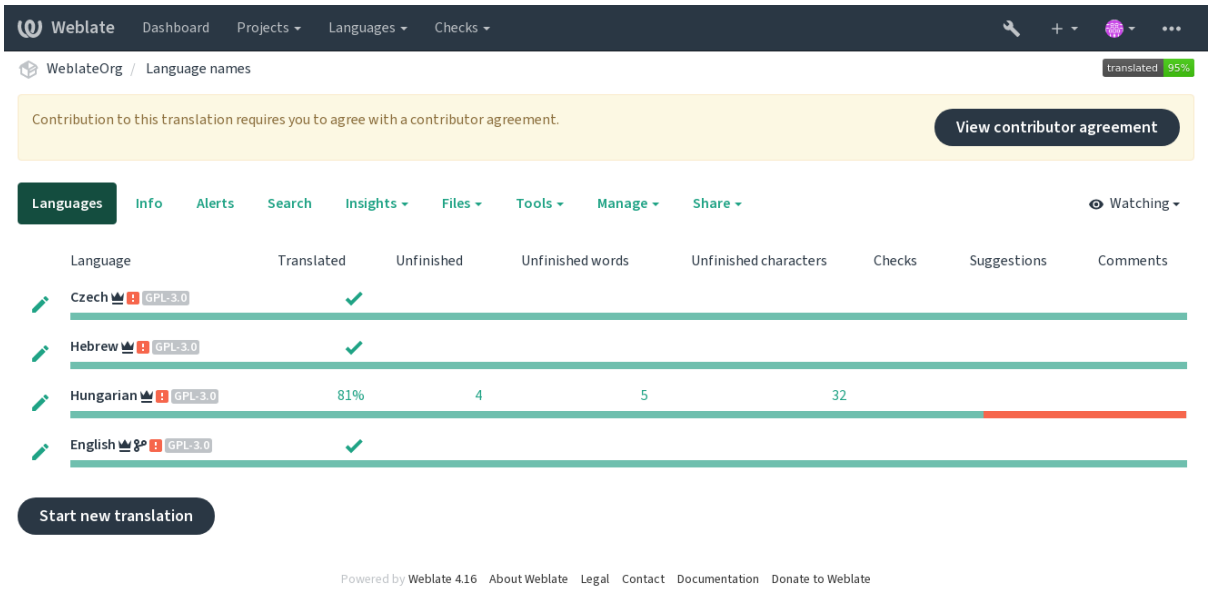
2.10.1 许可证信息

在指定许可证信息的时候（许可证名称和 URL）后，这个信息将显示在各个部件配置的翻译信息部分。

如果不需要特别同意的话，这通常是放置许可信息的最佳位置。如果您的项目或翻译不是自由的，您很可能需要事先同意。

2.10.2 贡献者协议

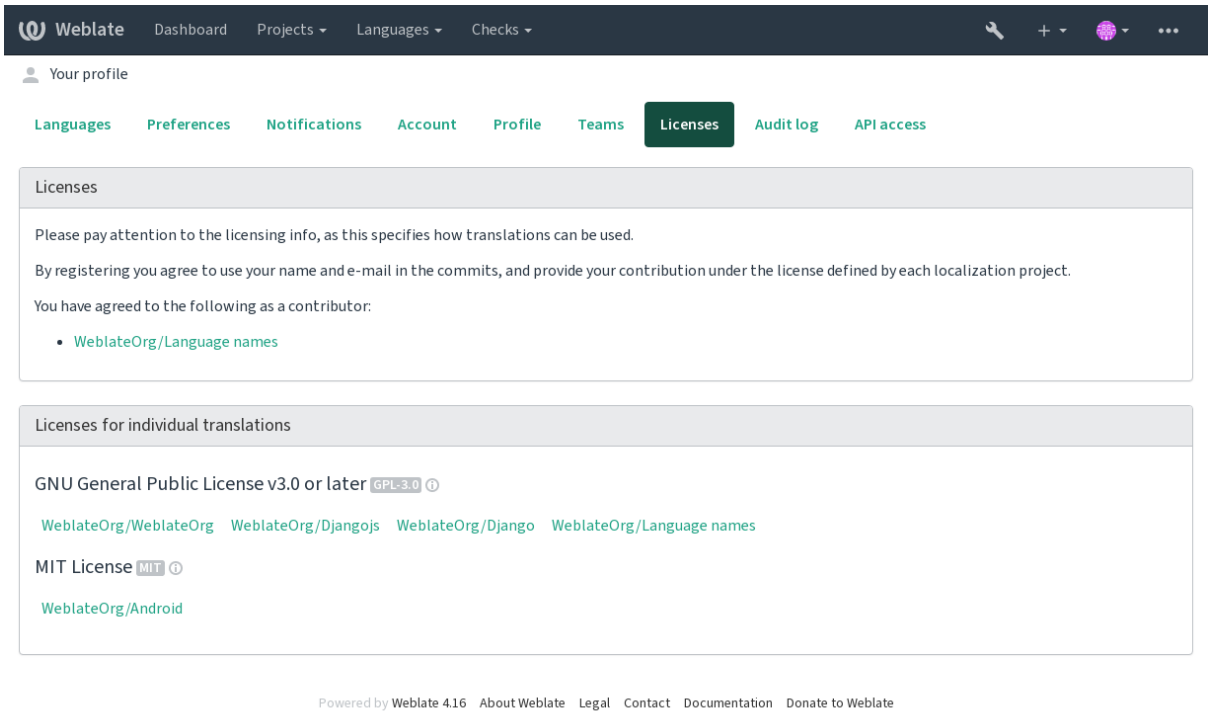
如果您指定了贡献者许可协议，那么只有同意该协议的用户能够作贡献。在访问翻译时，这是一个清晰可见的步骤：



输入的文本被格式化为段落，并且可以包括外部链接。不能使用 HTML 标记。

2.10.3 用户许可证

任何用户都可以在其个人资料中查看实例上所有公共项目的所有翻译许可证：



2.11 翻译进程

2.11.1 建议投票

每个人可以默认添加建议，由登录用户来接受。当超过一名登录用户同意时，建议投票结果可以使用字符串，通过 *建议投票设置*部件配置，来打开投票，并且通过 *自动接受建议*来设置接受建议的阈值（如果投票的话，这也包括来自提出建议的用户的投票）。

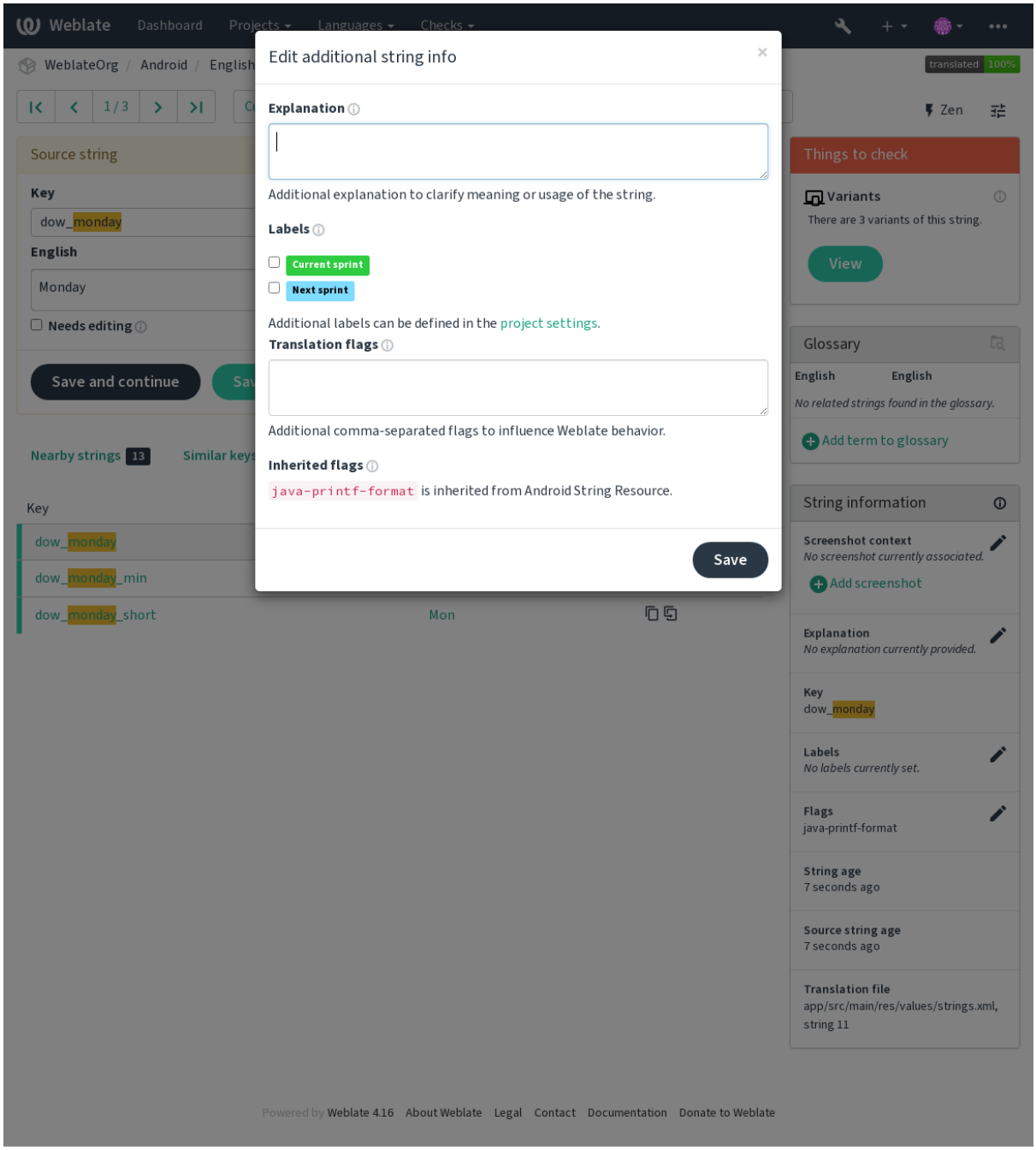
备注： 一旦设置了自动接受，普通用户会失去直接保存翻译或接受建议的权限。要绕过这一点，可以通过设置 *当建议强制时编辑字符串*权限。

您可以将这些与访问控制 结合起来，形成下列设置之一：

- 用户提出建议并对建议进行投票，由一个有限的小组控制采纳的内容。——打开投票。——关闭自动接受。——不允许用户保存翻译。
- 用户提出建议并对建议进行投票，一旦有规定数量的用户同意，就自动接受。——打开投票。——设置自动接受所需要的投票数量。
- 对建议的可选投票。（当用户对做出的多个建议不确定时，可以由用户可选地使用。）——只打开投票。

2.11.2 源字符串另外的信息

通过向字符串添加额外的信息来增强翻译过程，这些信息包括解释、字符串优先级、检查标记和可视化上下文。有些信息可以从翻译文件中提取，有些可以通过编辑额外的字符串信息添加:



直接在翻译界面点击 截图上下文或 标记旁的“编辑”图标即可进入此界面。


Weblate
Dashboard
Projects
Languages
Checks
WeblateOrg / Django / Czech / Translate
translated 96%
11 / 26
All strings
Position and priority
Zen
Translation
Explanation
Help text for automatic translation tool
English
Automatic translation via machine translation uses active machine translation engines to get the best possible translations and applies them in this project.
Czech
Automatický překlad prostřednictvím strojového překladu používá aktivní enginy strojového překladu pro získání nejlepších možných překladů a použije je na tento projekt.
Needs editing
169/1570 · 157
Save and continue
Save and stay
Suggest
Skip
Nearby strings 26
Comments
Automatic suggestions
Other languages 4
History
Context
English
Czech
Actions
Files
Soubory
Automatic translation
Automatický překlad
Add new translation string
Add new translation string
Translation status
Stav překladu
Singular
%(count)s word
Plural
%(count)s words
One
%(count)s slovo
Few
%(count)s slova
Many
%(count)s slov
Other components
Další součásti
Translation file
Soubor s překladem
Download
Stáhnout
Browse all translation changes
Procházet všechny změny v překladu.
Automatic translation takes existing translations in this project and applies them to the current component. It can be used to push translations to a different branch, to fix inconsistent translations or to translate a new component using translation memory.
Automatický překlad použije stávající překlady v projektu na tuto součást. Může být užitečný pro sloučení překladů z jiné větve, opravu nekonzistentních překladů nebo překlad nové součásti pomocí překladové paměti.
Automatic translation via machine translation uses active machine translation engines to get the best possible translations and applies them in this project.
Automatický překlad prostřednictvím strojového překladu používá aktivní enginy strojového překladu pro získání nejlepších možných překladů a použije je na tento projekt.
You can add new translation string here, it will automatically appear in all translations.
Zde můžete přidat nový řetězec k překladu, automaticky se objeví ve všech jazycích.
The uploaded file will be merged with the current translation. In case you want to overwrite already translated strings, don't forget to enable it.
Nahraný soubor bude sloučen se stávajícími překlady. Pokud chcete přepsat již přeložené řetězce, nezapomeňte to povolit.
The uploaded file will be merged with the current translation.
Nahraný soubor bude sloučen se stávajícími překlady.
The fulltext search might not work properly as the fulltext index for this translation is not yet up to date.
Fulltextové vyhledávání nemusí fungovat správě, protože fulltextový index pro tento překlad ještě není plně zpracován.
Review
Kontrola
Review translations touched by other users.
Zkontrolovat překlady od ostatních uživatelů.
Start review
Začít kontrolu
Percent
Procenta
Total
Celkem
Failing check
Neúspěšných kontrol
Last activity
Poslední aktivita
Last change
Poslední změna
Last author
Poslední autor
Question for a mathematics-based CAPTCHA, the %s is an arithmetic problem
What is %s?
Kolik to je?
The string uses three dots (...) instead of an ellipsis character (…)
Glossary
English
Czech
machine translation
strojový překlad
WeblateOrg
project
projekt
WeblateOrg
Add term to glossary
String information
Screenshot context
No screenshot currently associated.
Add screenshot
Explanation
Help text for automatic translation tool
Labels
No labels currently set.
Flags
No flags currently set.
Source string location
weblate/templates/translation.html:212
String age
2 seconds ago
Source string age
2 seconds ago
Translation file
weblate/locale/cs/LC_MESSAGES/django.po, string 11
Powered by Weblate 4.16
About Weblate
Legal
Contact
Documentation
Donate to Weblate

字符串优先级

在 2.0 版本加入.

使用 `priority` 标记可以更改字符串优先级，更高优先级的字符串会更早地被提供以进行翻译。

提示： 这可以用于以合乎逻辑的方式编排翻译流程。

参见：

质量检查

翻译标记

在 2.4 版本加入.

在 3.3 版本发生变更: 之前被称为 质量检查标记，它不再只配置检查了。

质量检查和其他 Weblate 行为的定制，请参见使用标记定制行为。

字符串标记也继承了部件配置 中的翻译标记 和翻译文件中的标记（见支持的文件格式）。

参见：

质量检查, 使用标记定制行为

解释

在 4.1 版本发生变更: 在以前的版本中这被称为 额外上下文。

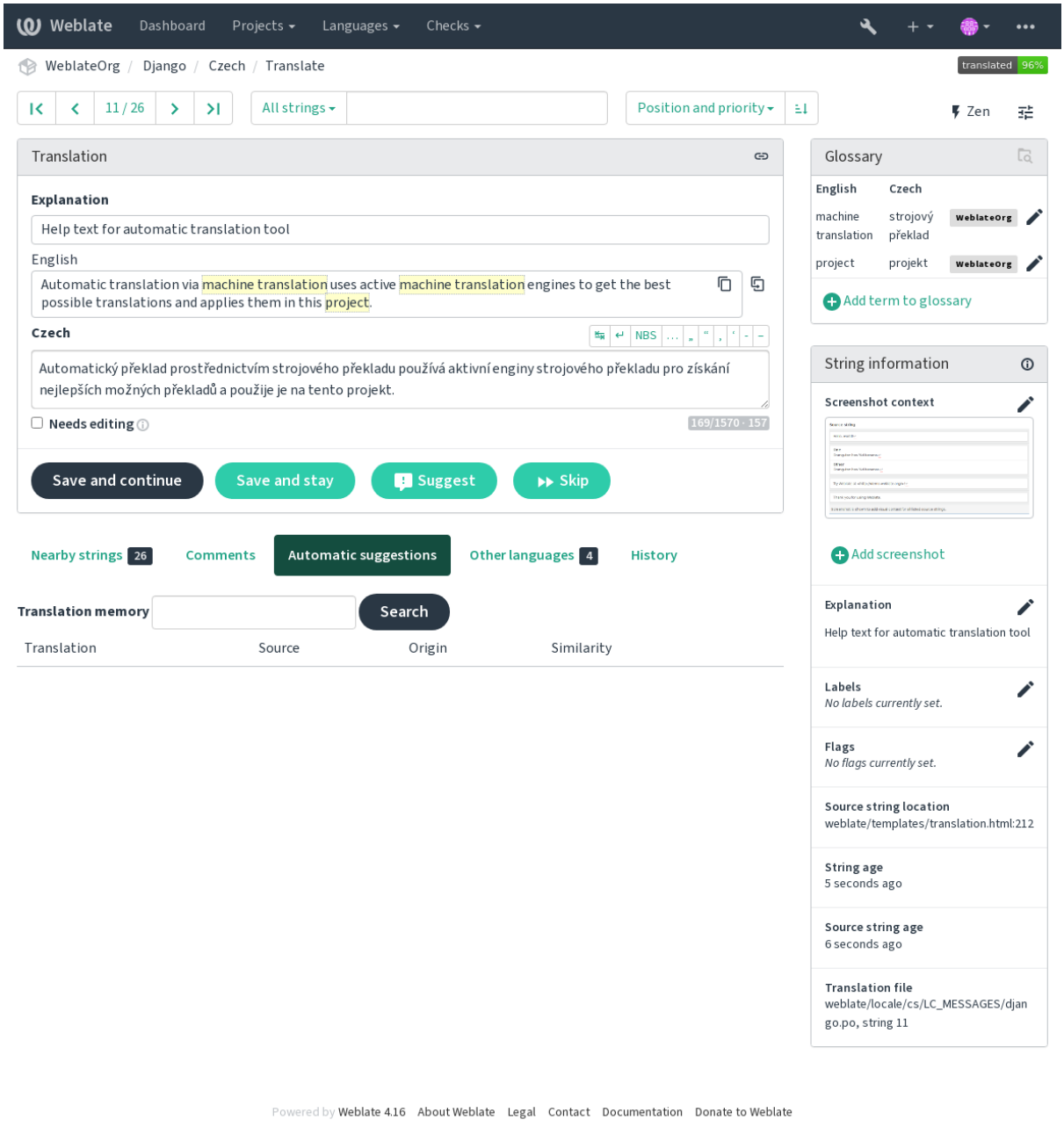
使用解释来阐明翻译的范围或翻译的用法。您可以使用 Markdown 语法来包含链接和其它标记。

字符串的可视化上下文

在 2.9 版本加入.

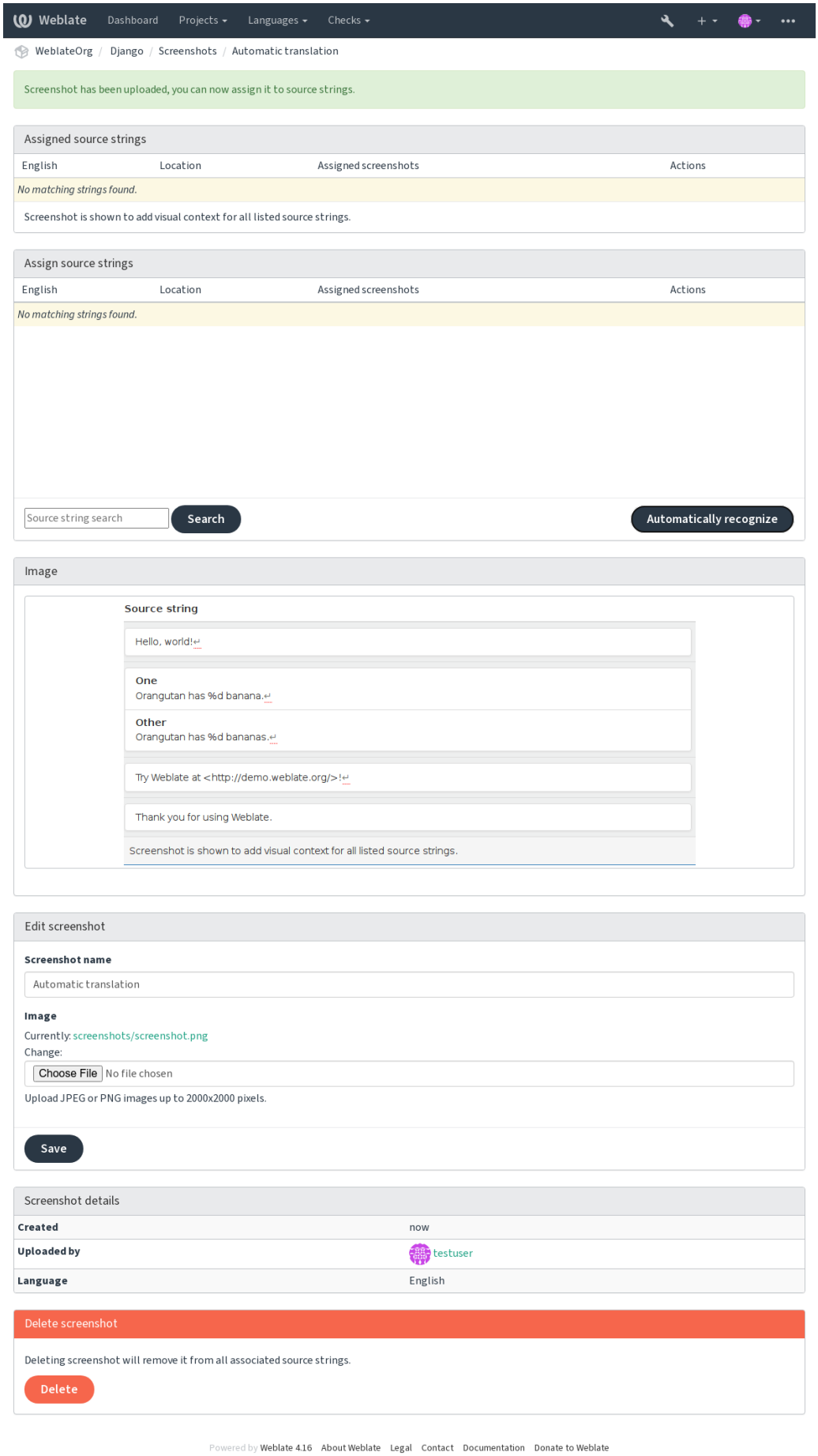
你可以将显示你程序中使用的给定源字符串的截屏上传。这帮助译者理解它用在哪里，并且应该如何翻译。

上传的截屏显示在翻译上下文侧栏中：



除了源字符串另外的信息，截图在 工具菜单下有个单独的管理界面。上传截图，将它们手动分配给源字符串，或者使用光学字符识别（OCR）来进行。

上传截图后，就可以在这个界面进行管理以及和源字符串的关联：


Weblate
Dashboard
Projects
Languages
Checks
WeblateOrg / Django / Screenshots / Automatic translation
Screenshot has been uploaded, you can now assign it to source strings.
Assigned source strings
English
Location
Assigned screenshots
Actions
No matching strings found.
Screenshot is shown to add visual context for all listed source strings.
Assign source strings
English
Location
Assigned screenshots
Actions
No matching strings found.
Source string search
Search
Automatically recognize
Image
Source string
Hello, world!
One
Orangutan has %d banana.
Other
Orangutan has %d bananas.
Try Weblate at <http://demo.weblate.org/>!
Thank you for using Weblate.
Screenshot is shown to add visual context for all listed source strings.
Edit screenshot
Screenshot name
Automatic translation
Image
Currently: screenshots/screenshot.png
Change:
Choose File
No file chosen
Upload JPEG or PNG images up to 2000x2000 pixels.
Save
Screenshot details
Created
now
Uploaded by
testuser
Language
English
Delete screenshot
Deleting screenshot will remove it from all associated source strings.
Delete
Powered by Weblate 4.16
About Weblate
Legal
Contact
Documentation
Donate to Weblate

2.12 检查和修正

2.12.1 定制的自动修正

还可以应用除了自动修正以外自己的自动修正，并将它们包括到*AUTOFIX_LIST*。

自动修复很强大，但可能导致损坏；写脚本的时候要小心。

例如，后面的自动修复会将每次出现的字符串 `foo` 在翻译中替换为 `bar`：

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

from django.utils.translation import gettext_lazy as _

from weblate.trans.autofixes.base import AutoFix


class ReplaceFooWithBar(AutoFix):
    """Replace foo with bar."""

    name = _("Foobar")

    def fix_single_target(self, target, source, unit):
        if "foo" in target:
            return target.replace("foo", "bar"), True
        return target, False
```

为了安装定制的检查，在*AUTOFIX_LIST* 中为 Python 类提供完全合规的路径，请参见定制的质量检查、附加组件和自动修复。

2.12.2 使用标记定制行为

您可以使用标记微调 Weblate 的行为。这可以在源字符串层面进行（请参见源字符串另外的信息），或者在部件配置（翻译标记）中进行。某些文件格式还允许直接在格式中指定标记（请参见支持的文件格式）。

标记用逗号分隔，参数用冒号分隔。可以在字符串中使用引号来包含空格或特定字符。例如：

```
placeholders:"special:value":"other value", regex:.*
```

单引号和双引号都被接受，特殊字符使用反斜杠进行转义：

```
placeholders:"quoted \"string\"":'single \'quoted\''
```

以下是目前接受的标记列表：

rst-text
: 将文本视为 reStructuredText 文档，影响未更改的译文。

dos-eol
: 使用 DOS 的行尾标记，而不使用 Unix 的（使用 `\r\n` 而不使用 `\n`）。

read-only
: 这条字符串是只读的，且不应该在 Weblate 中进行编辑。请参见只读字符串。

priority:N
: 字符串的优先级。高优先级的字符串首先出现被翻译。默认的优先级是 100，字符串的优先级越高，就会越早安排翻译。

max-length:N
: 将字符串的最大长度限制为 N 个字符，请参见译文最大长度。

xml-text
将文本看作 XML 文档，影响*XML 语法* 和*XML 标记*。

font-family:NAME
定义 font-family 来提供检查，请参见管理字型。

font-weight:WEIGHT
定义 font-weight 来提供检查，请参见管理字型。

font-size:SIZE
定义 font-size 来提供检查，请参见管理字型。

font-spacing:SPACING
定义渲染检查的字母间隔，请参见管理字型。

icu-flags:FLAGS
指定自定义*ICU MessageFormat* 质量检查行为的标记。

icu-tag-prefix:PREFIX
为*ICU MessageFormat* 质量检查设置必需的 XML 标签前缀。

placeholders:NAME:NAME2:...
译文中需要的占位符字符串，请参见占位符。

replacements:FROM:TO:FROM2:TO2...
当检查结果文本参数时执行替换（例如在最大译文长度 或译文最大长度 中）。这一典型应用的情况拓展了非译元素，确保匹配那些即使使用了长值的文本，例如 `replacements:%s:"John Doe"`。

variants:SOURCE
将此字符串标记为具有匹配源的字符串的变体。见 variants。

regex:REGEX
用于匹配翻译文件的正则表达式，详见正则表达式。

forbidden
表示术语表中禁止的译文，参见禁止的译文。

strict-same
使“未更改的译文”检查不使用内置单词黑名单。请参见未更改的译文。

check-glossary
启用不遵循术语表 质量检查。

angularjs-format
启用*AngularJS 插值字符串* 质量检查。

c-format
启用*C 格式* 质量检查。

c-sharp-format
启用*C# 格式* 质量检查。

es-format
启用*ECMAScript 模板字面量* 质量检查。

i18next-interpolation
启用*i18next 插值* 质量检查。

icu-message-format
启用*ICU MessageFormat* 质量检查。

java-printf-format
启用*Java 格式* 质量检查。

java-format
启用*Java MessageFormat* 质量检查。

javascript-format
启用*JavaScript 格式* 质量检查。

lua-format
启用*Lua 格式* 质量检查。

object-pascal-format
启用*Object Pascal 格式* 质量检查。

percent-placeholders
启用*百分比占位符* 质量检查。

perl-format
启用*Perl 格式* 质量检查。

php-format
启用*PHP 格式* 质量检查。

python-brace-format
启用*Python brace 格式* 质量检查。

python-format
启用*Python 格式* 质量检查。

qt-format
启用*Qt 格式* 质量检查。

qt-plural-format
启用*Qt 复数格式* 质量检查。

ruby-format
启用*Ruby 格式* 质量检查。

scheme-format
启用*Scheme 格式* 质量检查。

vue-format
启用*Vue I18n 格式化* 质量检查。

md-text
将文本看作 Markdown 文档。启用*Markdown 链接*, *Markdown 引用*, 和*Markdown 语法* 质量检查。

case-insensitive
调整检查行为不区分大小写。目前仅影响*占位符* 质量检查。

safe-html
启用*不安全的 HTML* 质量检查。

url
字符串应该只由一个 URL 组成。启用*URL* 质量检查。

ignore-all-checks
忽略所有质量检查。

ignore-bbcode
跳过*BBCode 标记* 质量检查。

ignore-duplicate
跳过*连续重复的单词* 质量检查。

ignore-check-glossary
跳过*不遵循术语表* 质量检查。

ignore-double-space
跳过*双空格* 质量检查。

ignore-angularjs-format
跳过*AngularJS 插值字符串* 质量检查。

ignore-c-format
跳过*C 格式* 质量检查。

ignore-c-sharp-format
跳过*C# 格式* 质量检查。

ignore-es-format
跳过*ECMAScript 模板字面量* 质量检查。

ignore-i18next-interpolation
跳过*i18next 插值* 质量检查。

ignore-icu-message-format
跳过*ICU MessageFormat* 质量检查。

ignore-java-format
跳过*Java MessageFormat* 质量检查。

ignore-java-printf-format
跳过*Java 格式* 质量检查。

ignore-javascript-format
跳过*JavaScript 格式* 质量检查。

ignore-lua-format
跳过*Lua 格式* 质量检查。

ignore-object-pascal-format
跳过*Object Pascal 格式* 质量检查。

ignore-percent-placeholders
跳过*百分比占位符* 质量检查。

ignore-perl-format
跳过*Perl 格式* 质量检查。

ignore-php-format
跳过*PHP 格式* 质量检查。

ignore-python-brace-format
跳过*Python brace 格式* 质量检查。

ignore-python-format
跳过*Python 格式* 质量检查。

ignore-qt-format
跳过*Qt 格式* 质量检查。

ignore-qt-plural-format
跳过*Qt 复数格式* 质量检查。

ignore-ruby-format
跳过*Ruby 格式* 质量检查。

ignore-scheme-format
跳过*Scheme 格式* 质量检查。

ignore-vue-format
跳过*Vue I18n 格式化* 质量检查。

ignore-translated
跳过*曾被翻译过* 质量检查。

ignore-inconsistent
跳过*不一致的* 质量检查。

ignore-kashida
跳过*使用了 Kashida 字母* 质量检查。

ignore-md-link
跳过*Markdown 链接* 质量检查。

ignore-md-reflink
跳过*Markdown 引用* 质量检查。

ignore-md-syntax
跳过*Markdown 语法* 质量检查。

ignore-max-length
跳过译文最大长度 质量检查。

ignore-max-size
跳过最大译文长度 质量检查。

ignore-escaped-newline
跳过不匹配的 *\n* 质量检查。

ignore-end-colon
跳过不匹配的冒号 质量检查。

ignore-end-ellipsis
跳过不匹配的省略号 质量检查。

ignore-end-exclamation
跳过不匹配的感叹号 质量检查。

ignore-end-stop
跳过不匹配的句号 质量检查。

ignore-end-question
跳过不匹配的问号 质量检查。

ignore-end-semicolon
跳过不匹配的分号 质量检查。

ignore-newline-count
跳过不匹配的换行符 质量检查。

ignore-plurals
跳过缺少复数形式 质量检查。

ignore-placeholders
跳过占位符 质量检查。

ignore-punctuation-spacing
跳过标点间距 质量检查。

ignore-regex
跳过正则表达式 质量检查。

ignore-same-plurals
跳过相同的复数形式 质量检查。

ignore-begin-newline
跳过换行符开头 质量检查。

ignore-begin-space
跳过空格开头 质量检查。

ignore-end-newline
跳过换行符结尾 质量检查。

ignore-end-space
跳过空格结尾 质量检查。

ignore-same
跳过未更改的译文 质量检查。

ignore-safe-html
跳过不安全的 *HTML* 质量检查。

ignore-url
跳过*URL* 质量检查。

ignore-xml-tags
跳过*XML 标记* 质量检查。

ignore-xml-invalid
跳过*XML 语法* 质量检查。

ignore-zero-width-space
跳过零宽空格 质量检查。

ignore-ellipsis
跳过省略号 质量检查。

ignore-icu-message-format-syntax
跳过*ICU MessageFormat 语法* 质量检查。

ignore-long-untranslated
跳过长期未翻译 质量检查。

ignore-multiple-failures
跳过多项未通过的检查 质量检查。

ignore-unnamed-format
跳过多个未命名的变量 质量检查。

ignore-optional-plural
跳过未复数化 质量检查。

备注： 通常，对于任何检查，都可以使用标识符将规则命名为 `ignore-*`，所以您甚至可以将其用于自定义检查。

每个源字符串的设置，在部件配置 设置中，并且在翻译文件自身中（例如在 GNU gettex 中），能够理解这些标记。

2.12.3 强制检查

在 3.11 版本加入.

您可以通过在部件配置 中设置强制检查 来配置不能忽略的检查列表。每个列出的检查都不能在用户界面中关闭，任何未通过此检查的字符串都标记为 需要编辑 ‘（请参阅*:ref:‘states*）。

备注： 启用检查强制不会自动启用它。可以通过将相应的标记添加到字符串或部件标记来打开检查。

参见：

源字符串另外的信息, 翻译标记

2.12.4 管理字型

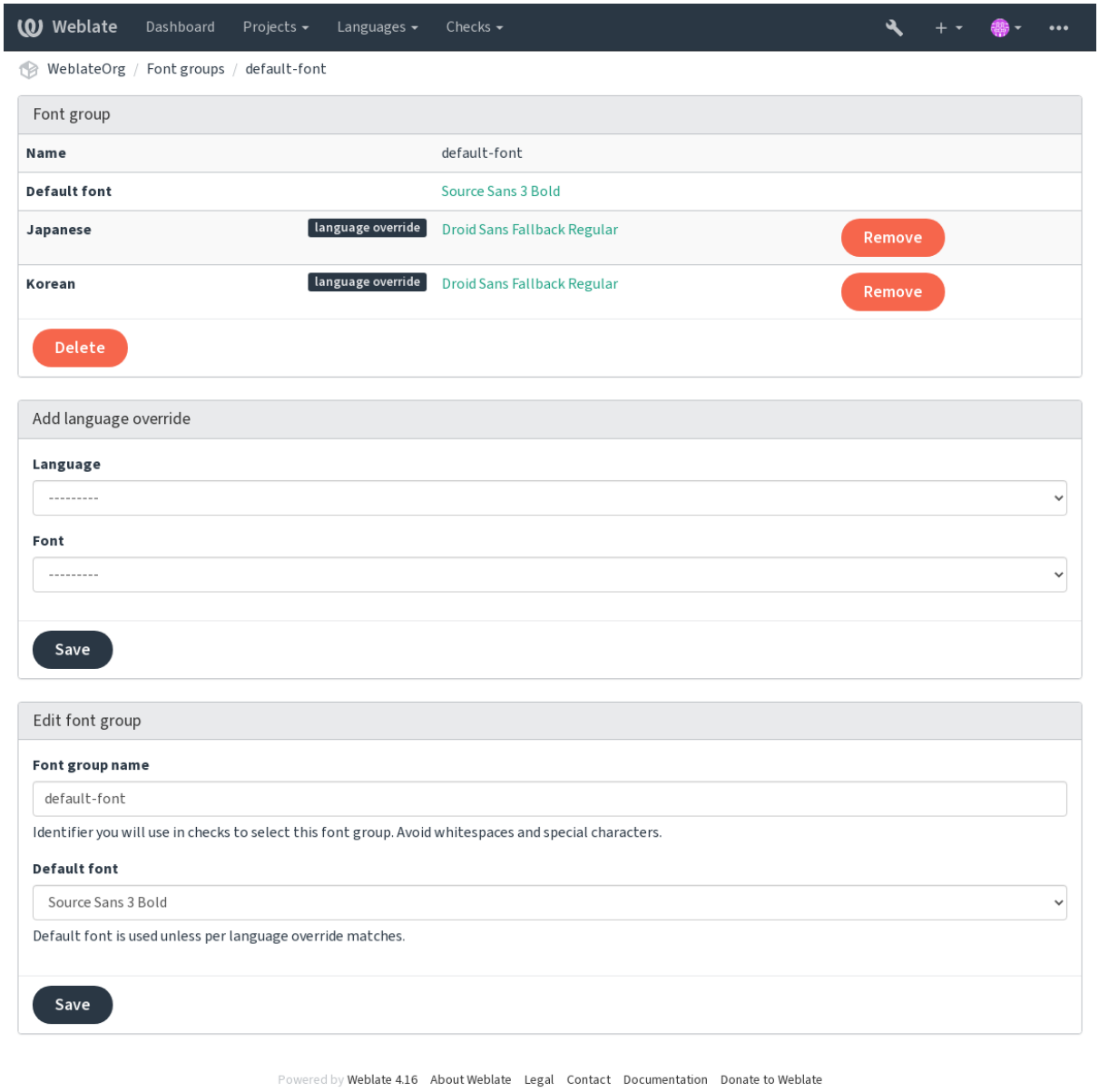
在 3.7 版本加入.

提示： 上传到 Weblate 的字体仅用于最大译文长度 检查，它们对 Weblate 用户界面没有影响。

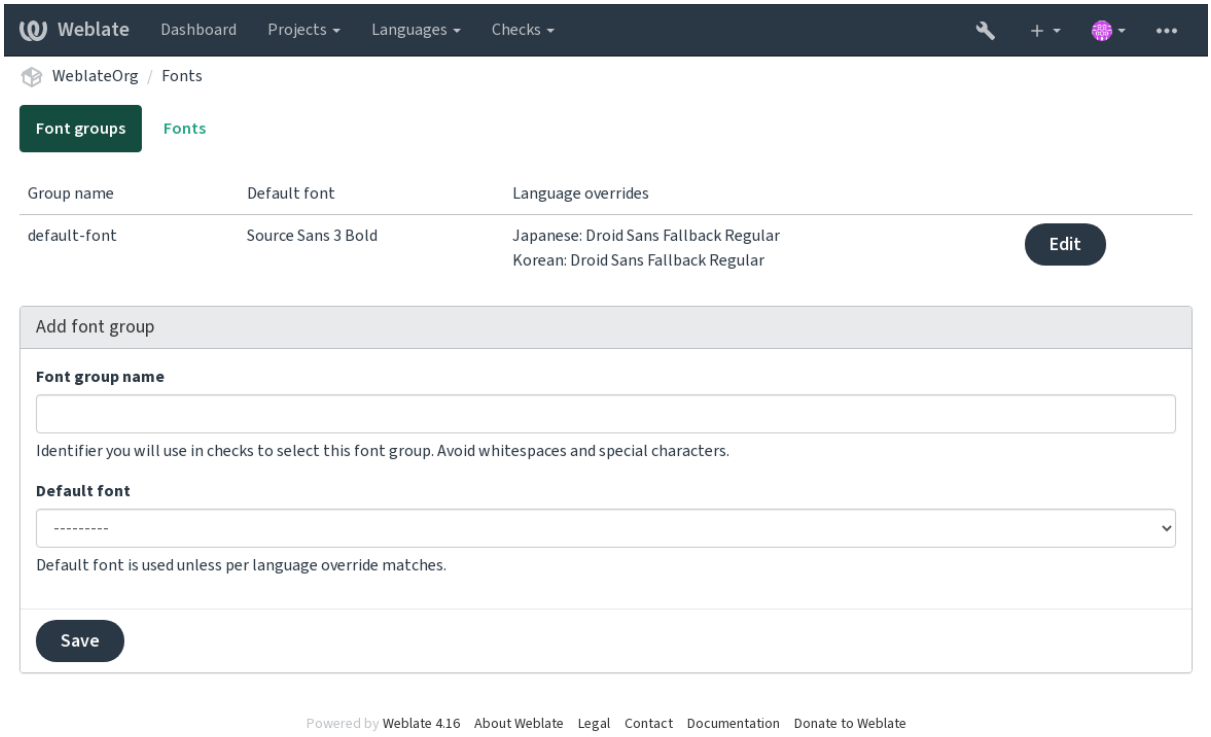
用于计算所呈现文本尺寸的最大译文长度 检查需要将字体加载到 Weblate 中，并使用翻译标记选中（请参见使用标记定制行为）。

在您的翻译项目 管理菜单下 字体中的 Weblate 字体管理工具提供了上传和管理字体的界面。可以上传 TrueType 或 OpenType 字体，设置 font-groups 并在检查中使用它们。

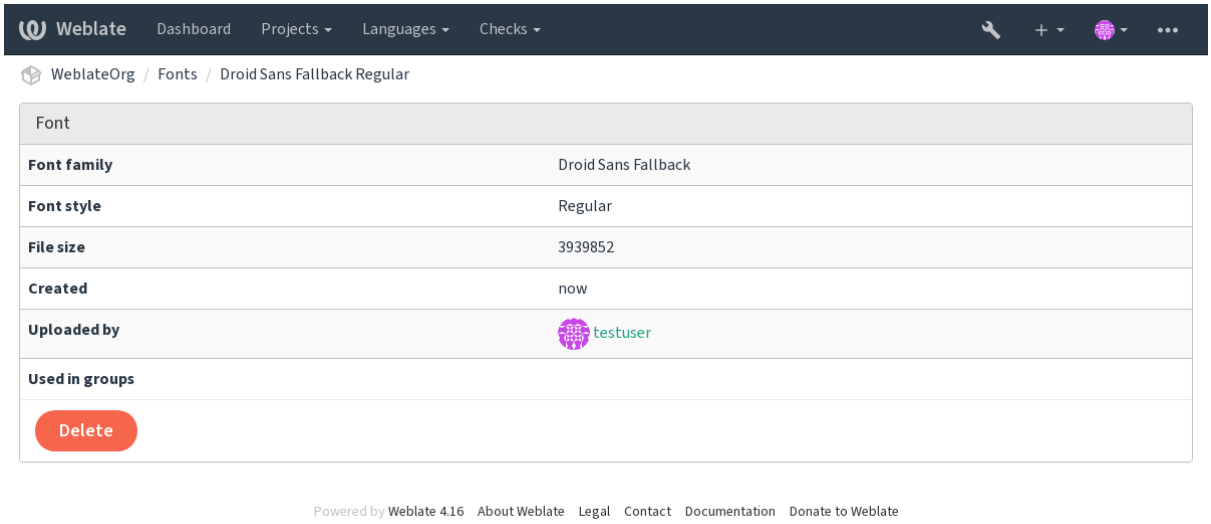
字型组允许为不同语言确定不同字型，这是非拉丁语言中典型需要的：



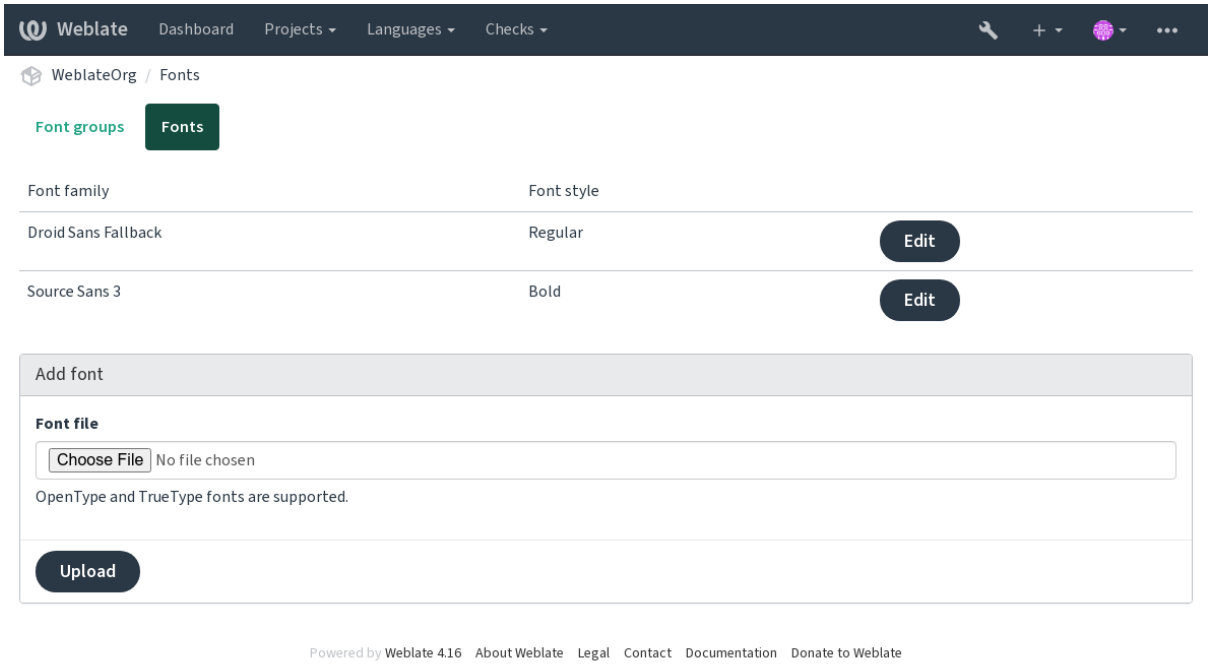
字型组通过名称识别，名称不能包含空格或特殊字符，这使它能够容易地用在检查定义中：



字型族和样式在上传后自动识别：



可以将几种字型加载到 Weblate 中：



为了使用字型来检查字符串长度，将适当的标记传递给它（请参见使用标记定制行为）。可能会需要后面这些：

max-size:500
指定以像素为单位的最大宽度。

font-family:ubuntu
通过指定其标识符来定义要使用的字型组。

font-size:22
指定以像素为单位的字体大小。

2.12.5 编写自己的检查

Weblate 内置了多种多样的质量检查，（请参阅质量检查），尽管它们可能没有涵盖您想要检查的所有东西。可以使用*CHECK_LIST* 来调整执行的检查列表，也可以添加自定义检查。

1. 子类 *weblate.checks.Check*
2. 设置一些属性。
3. 应用 `check` （如果想要处理代码中的复数的话）或 `check_single` 方法（它将为你完成）。

一些示例：

为了安装定制的检查，在*CHECK_LIST* 中为 Python 类提供完全合格的路径，请参见定制的质量检查、附加组件和自动修复。

检查译文文本是否不包含“foo”

这是一个非常简单的检查，只是检查译文中是否缺少了字符串“foo”。

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

"""Simple quality check example."""

from django.utils.translation import gettext_lazy as _

from weblate.checks.base import TargetCheck


class FooCheck(TargetCheck):
    # Used as identifier for check, should be unique
    # Has to be shorter than 50 characters
    check_id = "foo"

    # Short name used to display failing check
    name = _("Foo check")

    # Description for failing check
    description = _("Your translation is foo")

    # Real check code
    def check_single(self, source, target, unit):
        return "foo" in target
```

检查捷克语译文文本复数是否不同

使用语言信息检查，验证捷克语中的两种复数形式不同。

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

"""Quality check example for Czech plurals."""

from django.utils.translation import gettext_lazy as _

from weblate.checks.base import TargetCheck


class PluralCzechCheck(TargetCheck):
    # Used as identifier for check, should be unique
    # Has to be shorter than 50 characters
    check_id = "foo"

    # Short name used to display failing check
    name = _("Foo check")

    # Description for failing check
    description = _("Your translation is foo")

    # Real check code
    def check_target_unit(self, sources, targets, unit):
        if self.is_language(unit, ("cs",)):
            return targets[1] == targets[2]
        return False
```

(续下页)

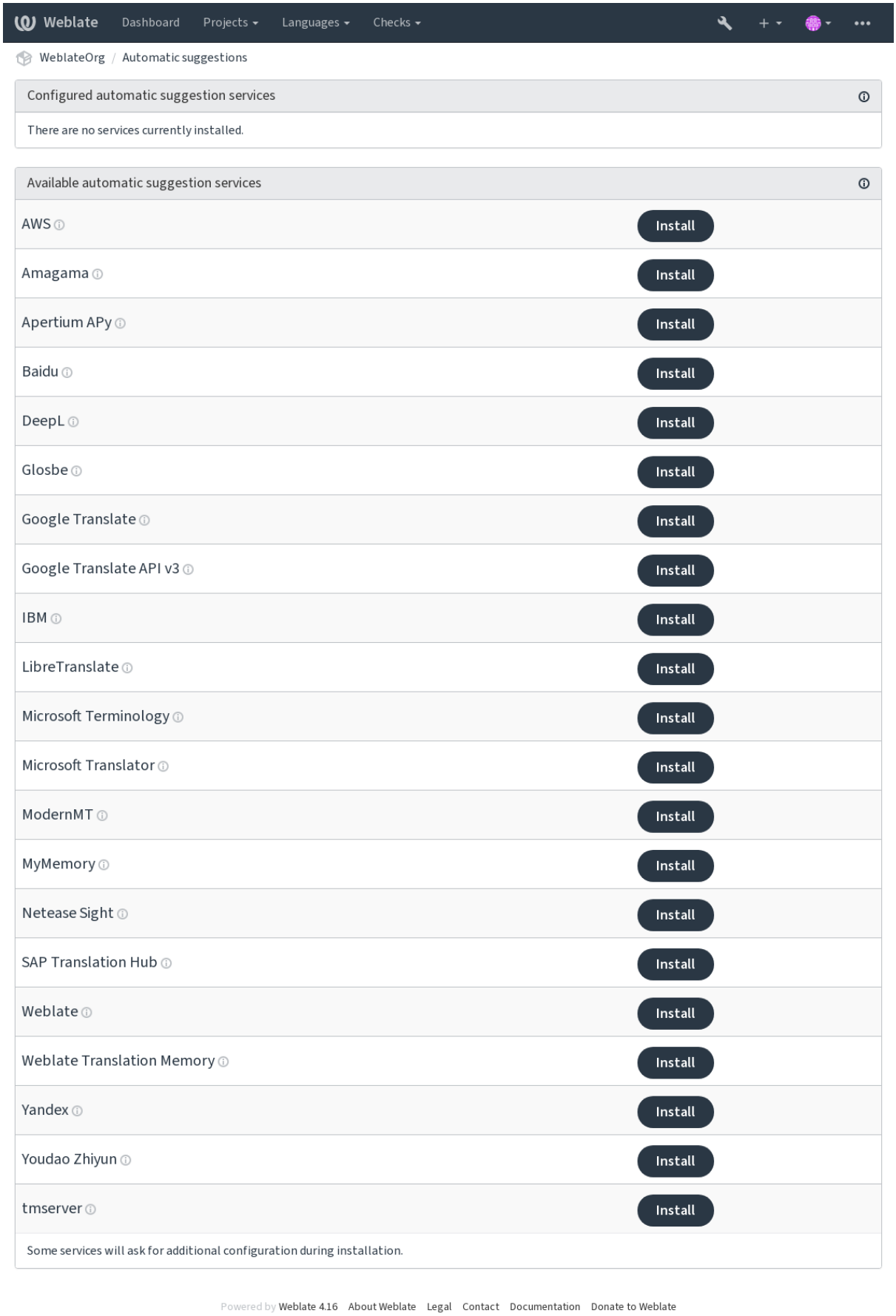
(接上页)

```
    def check_single(self, source, target, unit):
        """We don't check target strings here."""
        return False
```

2.13 配置自动建议

在 4.13 版本发生变更: 在 Weblate 4.13 之前，这些服务是在配置 中配置的。

内置了对几种机器翻译和翻译记忆服务的支持。每个服务都可以由管理员在整个网站或项目设置中打开：


Weblate
Dashboard
Projects
Languages
Checks
WeblateOrg / Automatic suggestions
Configured automatic suggestion services
There are no services currently installed.
Available automatic suggestion services
AWS
Amagama
Apertium APy
Baidu
DeepL
Glosbe
Google Translate
Google Translate API v3
IBM
LibreTranslate
Microsoft Terminology
Microsoft Translator
ModernMT
MyMemory
Netease Sight
SAP Translation Hub
Weblate
Weblate Translation Memory
Yandex
Youdao Zhiyun
tmserver
Install
Some services will ask for additional configuration during installation.
Powered by Weblate 4.16 About Weblate Legal Contact Documentation Donate to Weblate

备注： 它们受到其使用条款的约束，所以确保你被允许以你想要的方式使用它们。

该服务从部件配置 配置的源文件进行翻译，请参考源语言 。

参见：

自动建议

2.13.1 Amagama

服务 ID
`amagama`

配置
此服务无配置。

由 Virtaal 作者运行的*tmserver* 的特殊安装。

参见：

Installing amaGama, Amagama, amaGama 翻译记忆

2.13.2 Apertium APy

服务 ID
`apertium-apy`

配置

`url`	API 网址

一个自由软件机器翻译平台，提供有限语言的翻译。

使用 Apertium 的推荐方式是运行您自己的 Apertium-APy 服务器。

参见：

Apertium 网站, Apertium APy 文档

2.13.3 AWS

在 3.1 版本加入.

服务 ID
`aws`

配置

`key`	访问密钥 ID
`secret`	API 密钥
`region`	区域名

Amazon Translate 是神经机器翻译服务，用于将英语与广泛支持的语言进行互译。

参见：

Amazon 翻译文档

2.13.4 百度

在 3.2 版本加入.

服务 ID
`baidu`

配置

`key`	客户 ID
`secret`	客户端 secret

由百度提供的机器翻译服务。

这项服务使用 API，并且您需要从百度获得 ID 和 API 密钥来使用它。

参见:

百度翻译 API

2.13.5 DeepL

在 2.20 版本加入.

服务 ID
`deepl`

配置

`url`	API 网址
`key`	API 密钥

DeepL 是付费服务，提供一些语言的良好机器翻译。您需要购买 *DeepL API* 订阅，或者您可以使用传统的 *DeepL Pro (classic)* 计划。

用于 DeepL 服务的 API URL。在撰写本文时，有 v1 版 API 以及 v2 版 API 的免费和付费版本。

`https://api.deepl.com/v2/` （Weblate 默认）
用于付费计划的 API 用量，订阅是基于用量的。

`https://api-free.deepl.com/v2/`
用于免费计划的 API 用量，订阅是基于用量的。

`https://api.deepl.com/v1/`
用于计算机辅助翻译工具，可以按用户订阅。

之前 Weblate 被 DeepL 归类为计算机辅助翻译（CAT）工具，所以应该使用 v1 API，但现在应该使用 v2 API。因此它默认为 v2，如果您现在有一个计算机辅助翻译工具（CAT ）订阅，并希望 Weblate 使用它，您可以将其更改为 v1。

最简单的方法就是在浏览器中打开如下网址：

https://api.deepl.com/v2/translate?text=Hello&target_lang=FR&auth_key=XXX

将链接中的 xxx 替换为你自己 deepl 账号的 auth_key。如果你收到了一个包含有”Bonjour”（你好）的 JSON 对象，说明 URL 是正确，否则，试试其他方式。

Weblate 支持 DeepL 形式，它将根据语言选择匹配的形式（例如，有 `de@formal` 和 `de@informal` ）。

参见:

DeepL 网站, DeepL 定价, DeepL API 文档

2.13.6 Glosbe

服务 ID
`glosbe`

配置
此服务无配置。

几乎所有现存语言的免费词典和翻译记忆库。

其 API 可免费使用，但翻译的使用受到所用数据源许可证的约束。为防止滥用，在一段时间内，可以从一个 IP 进行的调用是有限制的。

参见:

Glosbe 网站

2.13.7 Google 翻译

服务 ID
`google-translate`

配置

`key`	API 密钥

Google 提供的机器翻译服务。

这项服务使用了 Google Translation API，您需要得到 API 密钥，并在 Google API 控制台打开记费。

参见:

Google 翻译文档

2.13.8 Google 翻译 API v3

服务 ID
`google-translate-api-v3`

配置

`credentials`	Google 翻译服务账户信息
`project`	Google 翻译项目
`location`	Google 翻译位置

Google 云服务提供的机器翻译服务。

参见:

谷歌翻译文档, 使用客户端库认证云服务, 创建谷歌翻译项目, 谷歌云应用引擎位置

2.13.9 LibreTranslate

在 4.7.1 版本加入.

服务 ID
`libretranslate`

配置

`url`	API 网址
`key`	API 密钥

LibreTranslate 是一项自由开源的机器翻译服务。公共实例需要 API 密钥，但 LibreTranslate 可以自托管，并且有几个镜像可以免费使用 API。

`https://libretranslate.com/` （官方公开实例）
需要一个 API 密钥才能在该网站之外使用。

参见:

LibreTranslate 网站, LibreTranslate 资料库, LibreTranslate 镜像

2.13.10 微软术语服务

在 2.19 版本加入.

服务 ID
`microsoft-terminology`

配置
此服务无配置。

微软术语服务 API 允许您通过网络服务以编程方式访问语言门户中可用的术语、定义和用户界面（UI）字符串。

参见:

微软术语服务 API

2.13.11 微软翻译

在 2.10 版本加入.

服务 ID
`microsoft-translator`

配置

key	API 密钥	
base	应用的基网址	可用选项： api.cognitive.microsofttranslator.com –全球（非区域） api-apc.cognitive.microsofttranslator.com –亚太 api-eur.cognitive.microsofttranslator.com –欧洲 api-nam.cognitive.microsofttranslator.com –北美 api.translator.azure.cn –中国 api.cognitive.microsofttranslator.us –Azure 美国政府云
endp	身份验证服务 URL	可使用下方的区域字段指定区域或多服务。 可用选项： api.cognitive.microsoft.com –全球 api.cognitive.azure.cn –中国 api.cognitive.microsoft.us –Azure 美国政府云
regi	认证服务区域	

由微软在 Azure 门户提供的机器翻译服务，是认知服务的一种。

Weblate 使用了 Translator API V3。

Translator Text API V2

您在 Translator API V2 使用的密钥可以在 API 3 中使用。

Translator Text API V3

您需要在 Azure 门户注册，并使用得到的密钥。您还需要将新的 Azure 密钥的 `region` 设置为您服务的语言环境。

提示： 中国区 Azure 请使用你在 Azure 门户的端点。

参见：

认知服务 - 文本翻译 API，Microsoft Azure 门户，基 URL，“使用多服务资源进行身份验证”，“使用访问令牌进行身份验证”部分

2.13.12 ModernMT

在 4.2 版本加入.

服务 ID
`modernmt`

配置

`url`	API 网址
`key`	API 密钥

参见:

ModernMT API,

2.13.13 MyMemory

服务 ID
`mymemory`

配置

`email`	联系电子邮箱
`username`	用户名
`key`	API 密钥

使用机器翻译的巨量翻译记忆库。

自由匿名使用，当前限制为 100 个请求/天，或者在 `email` 中提供电子邮箱联系地址时限制为 1000 个请求/天。您还可以向他们请求更多。

参见:

MyMemory 网站

2.13.14 网易见外

在 3.3 版本加入.

服务 ID
`netease-sight`

配置

`key`	客户 ID
`secret`	客户端 secret

网易提供的机器翻译服务。

这项服务使用 API，您需要从网易获取密钥和密码。

参见:

网易见外翻译平台

2.13.15 SAP 翻译中心

服务 ID

`sap-translation-hub`

配置

`url`	API 网址	
`key`	API 密钥	
`username`	SAP 用户名	
`password`	SAP 密码	
`enable_m`	启用机器翻译	
`domain`	翻译域	翻译域的 ID，例如 BC。如果您不指定域，该方法会搜索所有可用域中的翻译。

SAP 提供的机器翻一下服务。

你需要有 SAP 账户（且 SAP 云平台中 SAP 翻译中心已启用）来使用这项服务。

除了术语数据库，你还可以配置是否使用机器翻译服务。

备注： 要访问 Sandbox API，您需要设置 `url` 和 `key`.

要访问生产性 API，你需要设置 `url`, `username` 和 `password`.

参见：

SAP 翻译中心 API, Building the Base URL of SAP 翻译中心

2.13.16 tmserver

服务 ID

`tmserver`

配置

`url`	API 网址

您可以通过使用 Translate-toolkit 绑定的一个服务器并与之对话，来运行您自己的翻译服务器。您还可以将它与 amaGama 服务器一起使用，它是 tmserver 的增强版本。

1. 首先您会想要将一些数据导入翻译记忆库：

```
build_tmdb -d /var/lib/tm/db -s en -t cs locale/cs/LC_MESSAGES/django.po
build_tmdb -d /var/lib/tm/db -s en -t de locale/de/LC_MESSAGES/django.po
build_tmdb -d /var/lib/tm/db -s en -t fr locale/fr/LC_MESSAGES/django.po
```

2. 启动 tmserver 来收听您的请求：

```
tmserver -d /var/lib/tm/db
```

3. 配置 Weblate 与服务通信，默认地址为 `http://localhost:8888/tmserver/`.

参见：

tmserver Installing amaGama, Amagama, Amagama 翻译记忆

2.13.17 IBM Watson Language Translator

服务 ID
`ibm`

配置

`url`	API 网址
`key`	API 密钥

BM Watson Language Translator 将文本从一种语言翻译到另一种语言。该服务提供多个域名特定的模型。

参见:

Watson Language Translator, IBM Cloud API 文档

2.13.18 Weblate

服务 ID
`weblate`

配置
此服务无配置。

Weblate 机器翻译服务可以为 Weblate 内部已经翻译过的字符串提供翻译。它寻找现有字符串中的精确匹配。

2.13.19 Weblate 翻译记忆库

在 2.20 版本加入.

服务 ID
`weblate-translation-memory`

配置
此服务无配置。

使用翻译记忆库 作为机器翻译服务。任何过去翻译过的字符串（或上传到翻译记忆库）都可以用这种方式翻译。

2.13.20 Yandex

服务 ID
`yandex`

配置

`key`	API 密钥

Yandex 提供的机器翻译服务。

这项服务使用翻译 API，您需要从 Yandex 得到 API 密钥。

参见:

Yandex 翻译 API, 由 Yandex 翻译驱动

2.13.21 有道智云

在 3.2 版本加入.

服务 ID
youdao-zhiyun

配置

key	客户 ID
secret	客户端 secret

有道提供的机器翻译服务。

这项服务使用 API，您需要从有道获得 ID 和 API 密钥。

参见:

有道智云自然语言翻译服务

2.13.22 定制化的机器翻译

您还可以用几行 Python 代码来实现自己的机器翻译服务。此示例使用 `dictionary` Python 模块实现了一组确定的语言的机器翻译:

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

"""Machine translation example."""

import dictionary

from weblate.machinery.base import MachineTranslation


class SampleTranslation(MachineTranslation):
    """Sample machine translation interface."""

    name = "Sample"

    def download_languages(self):
        """Return list of languages your machine translation supports."""
        return {"cs"}

    def download_translations(
        self,
        source,
        language,
        text: str,
        unit,
        user,
        search: bool,
        threshold: int = 75,
    ):
        """Return tuple with translations."""
        for t in dictionary.translate(text):
            yield {"text": t, "quality": 100, "service": self.name, "source": text}
```

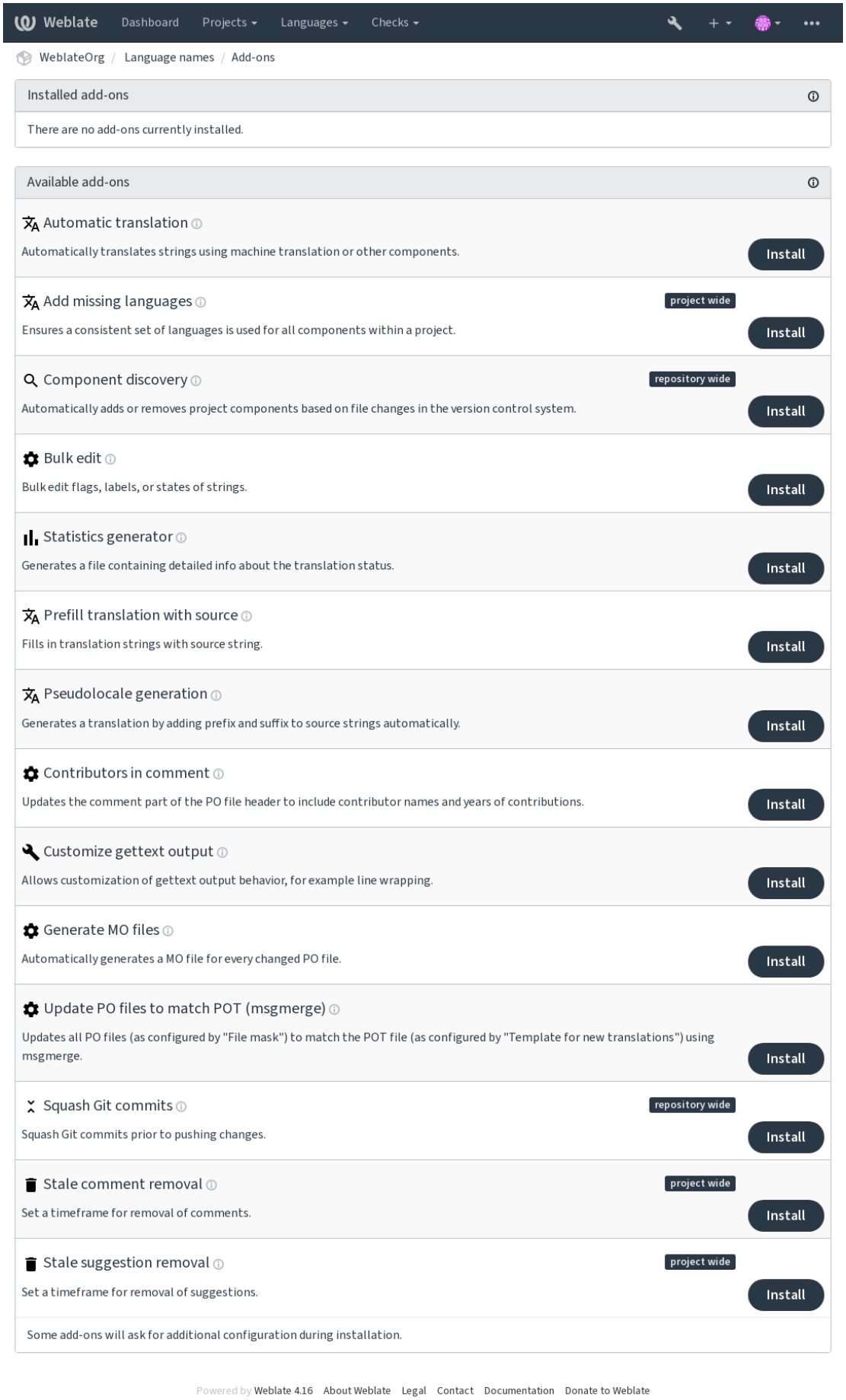
你可以在*WEBLATE_MACHINERY* 中列出自己的类，Weblate 就会开始使用它了。

2.14 附加组件

在 2.19 版本加入.

附加组件提供了自定义和自动化翻译工作流程的方法。管理员可以从每个相应翻译部件的 管理 ↓ 附加组件菜单添加和管理附加组件。

提示： 您也可以使用*API*、*DEFAULT_ADDONS* 或*install_addon* 来配置附加组件。


Weblate
Dashboard
Projects
Languages
Checks
WeblateOrg / Language names / Add-ons
Installed add-ons
There are no add-ons currently installed.
Available add-ons
Automatic translation
Automatically translates strings using machine translation or other components.
Install
Add missing languages
project wide
Ensures a consistent set of languages is used for all components within a project.
Install
Component discovery
repository wide
Automatically adds or removes project components based on file changes in the version control system.
Install
Bulk edit
Bulk edit flags, labels, or states of strings.
Install
Statistics generator
Generates a file containing detailed info about the translation status.
Install
Prefill translation with source
Fills in translation strings with source string.
Install
Pseudolocale generation
Generates a translation by adding prefix and suffix to source strings automatically.
Install
Contributors in comment
Updates the comment part of the PO file header to include contributor names and years of contributions.
Install
Customize gettext output
Allows customization of gettext output behavior, for example line wrapping.
Install
Generate MO files
Automatically generates a MO file for every changed PO file.
Install
Update PO files to match POT (msgmerge)
Updates all PO files (as configured by "File mask") to match the POT file (as configured by "Template for new translations") using msgmerge.
Install
Squash Git commits
repository wide
Squash Git commits prior to pushing changes.
Install
Stale comment removal
project wide
Set a timeframe for removal of comments.
Install
Stale suggestion removal
project wide
Set a timeframe for removal of suggestions.
Install
Some add-ons will ask for additional configuration during installation.
Powered by Weblate 4.16 About Weblate Legal Contact Documentation Donate to Weblate

2.14.1 内置附加组件

自动翻译

在 3.9 版本加入.

附加组件 ID

```
weblate.autotranslate.autotranslate
```

配置

mode	自动翻译模式	可用选项: suggest –添加为建议 translate –添加为翻译 fuzzy–添加为”需要编辑”
filte	搜索筛选器	请注意，翻译所有字符串将丢弃所有现有的译文。 可用选项: all –所有字符串 nottranslated –未翻译的字符串 todo –未完成的字符串 fuzzy –标记为需要编辑的字符串 check:inconsistent –未通过的检查：不一致
auto_	自动翻译的来源	可用选项: others –其他翻译部件 mt –机器翻译
compo	部件	输入要用作来源的部件的标识符，留空则使用当前项目中的所有部件。
engir	机器翻译引擎	
thres	匹配分数阈值	

触发器

部件更新，每天

使用机器翻译或其他部件自动翻译字符串。

触发条件:

- 当部件中出现新字符串时。
- 每个部件每个月一次，可以使用*BACKGROUND_TASKS* 进行配置。

参见:

自动翻译, 跨部件保持翻译一致

JavaScript 本地化 CDN

在 4.2 版本加入.

附加组件 ID

```
weblate.cdn.cdnjs
```

配置

threshold	译文阈值	包含的翻译的阈值。
css_select	CSS 选择器	检测可本地化元素的 CSS 选择器。
cookie_nam	语言 cookie 名称	存储语言偏好的 cookie 的名称。
files	从 HTML 文件中提取字符串	当前仓库或远程 URL 中要解析可翻译字符串的文件名列表。

触发器
每日，仓库提交后，仓库更新后

将译文发布到 JavaScript 或 HTML 本地化中使用的内容分发网络。

可以用于本地化静态 HTML 网页，或者在 JavaScript 代码中加载本地化文件。

为你的部件生成一个唯一的 URL，你可以将其包含在 HTML 页面中以本地化它们。详情见 weblate-cdn。

参见:

cdn-addon-config, weblate-cdn, cdn-addon-extract, cdn-addon-html

移除空白字符串

在 4.4 版本加入.

附加组件 ID
`weblate.cleanup.blank`

配置
此附加组件无配置。

触发器
仓库提交后，仓库更新后

从翻译文件中删除没有译文的字符串.

使用此方法可以使翻译文件中不存在任何空字符串 (例如，如果你的本地化库将它们显示为缺失，而不是退回到源字符串)。

参见:

Weblate 除了更新翻译，还更新翻译文件吗？

清理翻译文件

附加组件 ID
`weblate.cleanup.generic`

配置
此附加组件无配置。

触发器
仓库提交前，仓库更新后

更新所有翻译文件以匹配单语言译文模版文件。对于大多数文件格式来说，这意味着移除译文模版文件中不再出现的旧翻译条目。

参见:

Weblate 除了更新翻译，还更新翻译文件吗？

添加缺少的语言

附加组件 ID
`weblate.consistency.languages`

配置
此附加组件无配置。

触发器
每天，添加仓库后

确保对一个项目内所有部件使用一致的一组语言。

每隔 24 小时，和在 Weblate 中加入新语言时，会检查一次缺失的语言。

不像其他多数附加组件，这个附加组件影响整个项目。

提示： 用自动翻译 自动翻译新添加的字符串。

部件发现

附加组件 ID
`weblate.discovery.discovery`

配置

match	用于匹配翻译文件的正则表达式	
file_format	文件格式	
name_templa	自定义部件名称	
base_file_t	指定单语种译文模版文件名	双语翻译文件请留空。
new_base_te	为新的翻译条目指定翻译模版文件	用于创建新翻译的文件名。对于 gettext 格式，请选择.pot 文件。
intermediat	中间语言文件	中间翻译文件的文件名。在大多数情况下，这是开发者提供的翻译文件，用于创建实际的源字符串。
language_re	语言筛选	扫描文件掩码时用于筛选翻译文件的正则表达式。
copy_addons	将主部件的附加组件克隆到新创建的部件	
remove	删除不存在文件的部件	
confirm	我确认上述匹配是正确的	

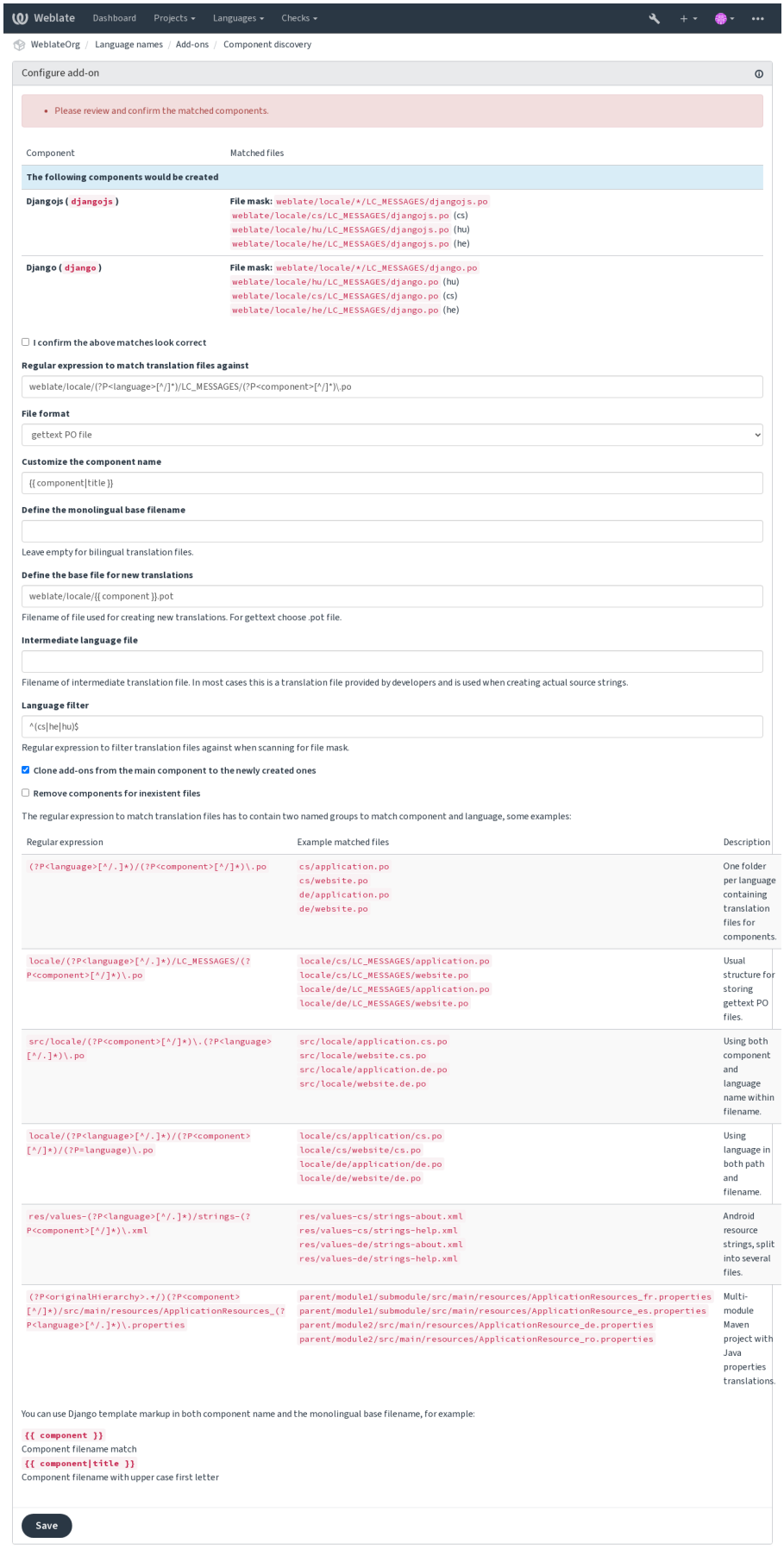
触发器
仓库更新后

根据版本控制系统（VCS）中文件更改的情况来自动添加或删除项目部件。

每次更新版本控制系统（VCS）时触发，在其他方面类似*import_project* 管理命令。通过这种方式，你可以在一个版本控制系统（VCS）中跟踪多个翻译部件。

该匹配是使用允许复杂配置的正则表达式完成的，但这样做需要一些知识。一些常见用例的示例可以在附加组件帮助部分找到。

点击 保存后，将显示匹配部件的预览，您可以从中检查配置是否真正符合您的需求：


Weblate
Dashboard
Projects
Languages
Checks
WeblateOrg / Language names / Add-ons / Component discovery
Configure add-on
Please review and confirm the matched components.
Component
Matched files
The following components would be created
Djangojs (djangojs)
File mask: weblate/locale/*/LC_MESSAGES/djangojs.po
weblate/locale/cs/LC_MESSAGES/djangojs.po (cs)
weblate/locale/hu/LC_MESSAGES/djangojs.po (hu)
weblate/locale/he/LC_MESSAGES/djangojs.po (he)
Django (django)
File mask: weblate/locale/*/LC_MESSAGES/django.po
weblate/locale/hu/LC_MESSAGES/django.po (hu)
weblate/locale/cs/LC_MESSAGES/django.po (cs)
weblate/locale/he/LC_MESSAGES/django.po (he)
I confirm the above matches look correct
Regular expression to match translation files against
weblate/locale/(?P<language>[^/]*)/LC_MESSAGES/(?P<component>[^/]*)\.po
File format
gettext PO file
Customize the component name
{{ component|title }}
Define the monolingual base filename
Leave empty for bilingual translation files.
Define the base file for new translations
weblate/locale/{{ component }}.pot
Filename of file used for creating new translations. For gettext choose .pot file.
Intermediate language file
Filename of intermediate translation file. In most cases this is a translation file provided by developers and is used when creating actual source strings.
Language filter
^(cs|he|hu)$
Regular expression to filter translation files against when scanning for file mask.
Clone add-ons from the main component to the newly created ones
Remove components for inexistent files
The regular expression to match translation files has to contain two named groups to match component and language, some examples:
Regular expression
Example matched files
Description
(?P<language>[^/.]*)/(?P<component>[^/]*)\.po
cs/application.po
cs/website.po
de/application.po
de/website.po
One folder per language containing translation files for components.
locale/(?P<language>[^/.]*)/LC_MESSAGES/(?P<component>[^/]*)\.po
locale/cs/LC_MESSAGES/application.po
locale/cs/LC_MESSAGES/website.po
locale/de/LC_MESSAGES/application.po
locale/de/LC_MESSAGES/website.po
Usual structure for storing gettext PO files.
src/locale/(?P<component>[^/]*)\.(?P<language>[^/.]*)\.po
src/locale/application.cs.po
src/locale/website.cs.po
src/locale/application.de.po
src/locale/website.de.po
Using both component and language name within filename.
locale/(?P<language>[^/.]*)/(?P<component>[^/]*)/(?P=language)\.po
locale/cs/application/cs.po
locale/cs/website/cs.po
locale/de/application/de.po
locale/de/website/de.po
Using language in both path and filename.
res/values-(?P<language>[^/.]*)/strings-(?P<component>[^/]*)\.xml
res/values-cs/strings-about.xml
res/values-cs/strings-help.xml
res/values-de/strings-about.xml
res/values-de/strings-help.xml
Android resource strings, split into several files.
(?P<originalHierarchy>.+/)(?P<component>[^/]*)/src/main/resources/ApplicationResources_(?P<language>[^/.]*)\.properties
parent/module1/submodule/src/main/resources/ApplicationResources_fr.properties
parent/module1/submodule/src/main/resources/ApplicationResource_es.properties
parent/module2/src/main/resources/ApplicationResource_de.properties
parent/module2/src/main/resources/ApplicationResource_ro.properties
Multi-module Maven project with Java properties translations.
You can use Django template markup in both component name and the monolingual base filename, for example:
{{ component }}
Component filename match
{{ component|title }}
Component filename with upper case first letter
Save
Powered by Weblate 4.16 About Weblate Legal Contact Documentation Donate to Weblate

提示： 部件发现附加组件使用*Weblate 内部网址*。这是在多个部件之间共享版本控制系统（VCS）设置的便捷方式。链接的部件使用主部件的本地仓库，其设置方法是将 `weblate://project/main-component` 填入各部件的*源代码仓库* 字段（在 管理 ↓ 设置 ↓ 版本控制系统）。这也节省了配置和系统资源的时间。

参见：

模板标记

批量编辑

在 3.11 版本加入.

附加组件 ID
`weblate.flags.bulk`

配置

`q`	查询	
`state`	要设置的状态	可用选项： `-1` –不要更改 `10` –需要编辑 `20` –已翻译 `30` –已核准
`add_flags`	要添加的翻译标记	
`remove_flags`	要删除的翻译标记	
`add_labels`	要添加的标签	
`remove_labels`	要删除的标签	

触发器
部件更新

批量编辑标记、标签或字符串状态。

从搜索查询 `NOT has:label` 开始，自动添加标签，然后给搜索查询所得字符串添加标签，直到所有字符串都有所需的标签。还可以对 Weblate 元数据进行其他自动化操作。

示例：

表 5: 自动给新字符串添加标签

搜索查询	`NOT has:label`
要添加的标签	近期

表 6: 将应用商店元数据文件所有更新日志字符串标记为只读

搜索查询	`language:en AND key:changelogs/`
要添加的翻译标记	`read-only`

参见：

批量编辑, 使用标记定制行为, labels

将未更改的译文标记为“需要编辑”

在 3.1 版本加入.

附加组件 ID
`weblate.flags.same_edit`

配置
此附加组件无配置。

触发器
单元创建后

每当从版本控制系统（VCS）导入新的可翻译字符串并与源字符串匹配时，就会在 Weblate 中标记为需要编辑。对于包含未翻译字符串的源字符串的文件格式尤其有用。

提示： 您可能还想通过在翻译标记 中添加 `strict-same` 标记来加强未更改的译文 检查。

参见：

翻译状态

将新的源字符串标记为“需要编辑”

附加组件 ID
`weblate.flags.source_edit`

配置
此附加组件无配置。

触发器
单元创建后

每当一个新的源字符串从版本控制系统（VCS）中导入时，它将在 Weblate 中被标记为需要编辑。这样就可以简单地筛选并编辑开发者编写的源字符串。

参见：

翻译状态

将导入的新译文标记为“需要编辑”

附加组件 ID
`weblate.flags.target_edit`

配置
此附加组件无配置。

触发器
单元创建后

每当一个新的可翻译字符串从版本控制系统（VCS）中导入时，它将在 Weblate 中被标记为需要编辑。这样就可以简单地筛选并编辑开发者创建的翻译。

参见：

翻译状态

统计数据生成器

附加组件 ID
`weblate.generate.generate`

配置

`filename`	所生成文件的名称
`template`	所生成文件的内容

触发器
仓库预提交

生成一个包含关于翻译状态详细信息的文件。

你可以在文件名和内容中使用 Django 模板，详细的标记说明请参见模板标记。

例如为每个翻译生成一个摘要文件：

所生成文件的名称
`locale/{{ language_code }}.json`

内容

```
{
    "language": "{{ language_code }}",
    "strings": "{{ stats.all }}",
    "translated": "{{ stats.translated }}",
    "last_changed": "{{ stats.last_changed }}",
    "last_author": "{{ stats.last_author }}",
}
```

参见：

模板标记

用原文预填充译文

在 4.11 版本加入.

附加组件 ID
`weblate.generate.prefill`

配置
此附加组件无配置。

触发器
部件更新，每天

用源字符串填充翻译字符串。

部件中所有未翻译的字符串将被填充为源字符串，并被标记为需要编辑。当你的翻译文件中不能兼容空字符串时请使用这个功能。

假语言环境（Pseudolocale）生成

在 4.5 版本加入.

附加组件 ID

`weblate.generate.pseudolocale`

配置

`source`	源字符串	
`target`	目标译文	此翻译中的所有字符串都将被覆盖
`prefix`	固定的字符串前缀	
`var_prefix`	可变的字符串前缀	
`suffix`	固定的字符串后缀	
`var_suffix`	可变的字符串后缀	
`var_multiplier`	可变部分乘数	可变部分重复多少次取决于源字符串的长度。
`include_readonly`	包含只读字符串	

触发器

部件更新，每天

通过向源字符串添加前缀和后缀自动生成翻译。

伪 locale 对于查找未准备好进行本地化的字符串很有用。这是通过修改所有可翻译的源字符串来实现的，使得在用伪 locale 语言运行应用程序时很容易发现未修改的字符串。

也可以查找其本地化对应项可能不适合布局的字符串。

使用可变部分可以寻找那些在本地化后可能不适合用户界面的字符串-它根据源字符串的长度来扩展文本。可变部分是以文本的长度乘以乘数来重复的。例如 `Hello world` 带有可变后缀 `_``和变量乘数 1 将变为``Hello world_____-` 后缀在源字符串中的每个字符后都重复一次。

将使用以下模式生成字符串：

`固定字符串前缀 可变字符串前缀 源字符串 可变字符串后缀 固定字符串后缀`

提示： 可以使用真正的语言进行检测，但在 Weblate 中有专用的假语言环境——*en_XA* 和 *ar_XB*。

提示： 您可以使用此附加组件来开始翻译到现有语言或类似语言的新语言环境。将翻译添加到部件后，请遵循附加组件。示例：如果您有 *fr* 的翻译并想开始翻译 *fr_CA*，只需将 *fr* 设置为源，将 *fr_CA* 设置为目标，并将前缀和后缀留空。

一旦您有新的翻译填重，请卸载附加组件，以防止 Weblate 更改复制后的翻译。

在注释中添加贡献信息

附加组件 ID
`weblate.gettext.authors`

配置
此附加组件无配置。

触发器
仓库预提交

更新 PO 文件标头的注释部分，以包含贡献者名字和贡献年份。

PO 文件标头如下所示：

```
# Michal Čihař <michal@weblate.org>, 2012, 2018, 2019, 2020.
# Pavel Borecki <pavel@example.com>, 2018, 2019.
# Filip Hron <filip@example.com>, 2018, 2019.
# anonymous <noreply@weblate.org>, 2019.
```

更新“配置文件”中的 ALL_LINGUAS 变量

附加组件 ID
`weblate.gettext.configure`

配置
此附加组件无配置。

触发器
添加仓库后，每天

当新的翻译添加时，更新 `configure` 、`configure.in` 或任何 `configure.ac` 文件中的 ALL_LINGUAS 变量。

自定义 gettext 输出

附加组件 ID
`weblate.gettext.customize`

配置

`widt`	自动换行	默认情况下 gettext 会在每行第 77 个字符处与换行符处换行。添加–no-wrap 参数后，换行只在换行符处发生。 可用选项： `77` –在 77 个字符和换行符处换行（xgettext 默认） `65535` –只在换行符处换行（如’xgettext –no-wrap’） `-1` –不换行

触发器
存储延迟加载

允许自定义 gettext 的输出行为，例如是否启用自动换行。

它提供以下选项：

- 在第 77 个字符处和换行符处换行
- 仅在换行符处换行
- 不换行

备注： 默认情况下 gettext 会在每行第 77 个字符处与换行符处换行。添加 `--no-wrap` 参数后，换行只会在换行符处发生。

更新 LINGUAS 文件

附加组件 ID
`weblate.gettext.linguas`

配置
此附加组件无配置。

触发器
添加仓库后，每天

添加新的翻译时更新 LINGUAS 文件。

生成 MO 文件

附加组件 ID
`weblate.gettext.mo`

配置

`path`	生成的 MO 文件的路径	如未指定，将使用 PO 文件对应的位置。

触发器
仓库预提交

为每个变更的 PO 文件自动生成 MO 文件。

生成的 MO 文件的位置可以定制化，并且其字段使用模板标记。

更新 PO 文件以匹配 POT 文件 (msgmerge)

附加组件 ID
`weblate.gettext.msgmerge`

配置

`previous`	保持已翻译字符串先前的 msgid
`no_location`	删除翻译字符串的位置
`fuzzy`	使用模糊匹配

触发器
仓库更新后

使用 **`msgmerge`** 来更新所有的 PO 文件（如文件掩码 所配置），而与 POT 文件（如新翻译的翻译模版 所配置）匹配。

每当从上游仓库中提取新的更改时都会被触发。多数 msgmerge 命令行选项可通过附加组件配置进行设置。

参见：

Weblate 除了更新翻译，还更新翻译文件吗？

挤压 Git 提交

附加组件 ID
`weblate.git.squash`

配置

`squash`	挤压提交	可用选项： `all` –所有提交并成一个提交 `language` –按语言 `file` –按文件 `author` –按作者
`append_t`	在挤压的提交说明附上尾注	尾注行是类似于 RFC 822 电子邮件标头的行，位于提交说明的其他自由格式部分的末尾，例如 “Co-authored-by …”。
`commit_m`	提交说明	将使用此提交说明来代替挤压提交的组合提交说明。

触发器
仓库提交后

在推送变更之前挤压 Git 提交。

在推送变更之前，可以通过以下任一模式挤压 Git 提交：

- 所有提交成一个
- 每种语言
- 每个文件
- 每位作者

原始提交说明保留，但其作者信息丢失，除非选择 `每位作者`，或者定制提交说明来包括它。

可以选择使用自定义提交说明覆盖原始提交说明。

可以选择从原始提交说明中删除尾注（像 `Co-authored-by: …`这样的提交行），并附在挤压的提交说明的末尾。这也会为每位译者产生合适的 `Co-authored-by:` 记录。

自定义 JSON 输出

附加组件 ID
`weblate.json.customize`

配置

`sort_keys`	对 JSON 键值排序	
`indent`	JSON 缩进	
`style`	JSON 缩进风格	可用选项： `spaces` –空格 `tabs` –制表符

触发器
存储延迟加载

允许调整 JSON 的输出行为，例如缩进或排序。

格式化 Java 属性文件

附加组件 ID
`weblate.properties.sort`

配置
此附加组件无配置。

触发器
仓库预提交

对 Java 属性文件进行格式化和排序。

- 将换行符合并到 Unix 换行符。
- Unicode 转义序列的大写格式（如果存在）。
- 去除空白行和注释。
- 按键对字符串进行排序。
- 删除重复的字符串。

陈旧评论删除

在 3.7 版本加入.

附加组件 ID
`weblate.removal.comments`

配置

`age`	要保持的天数

触发器
每天

设置删除评论的时间。

这可以用于删除可能已经过时的陈旧评论。小心使用，因为评论变旧并不意味着它们已经失去了重要性。

陈旧建议删除

在 3.7 版本加入.

附加组件 ID
`weblate.removal.suggestions`

配置

`age`	要保持的天数	
`votes`	投票阈值	移除阈值。当投票被禁用时该项无效果。

触发器
每天

设置删除建议的时间。

此附加组件在与建议投票一道用来删除在给定的时间内没有得到足够的正面投票的建议方面非常有用。

更新 RESX 文件

在 3.9 版本加入.

附加组件 ID
`weblate.resx.update`

配置
此附加组件无配置。

触发器
仓库更新后

更新所有翻译文件以匹配上游单语言译文模版文件。未使用的字符串将被删除，新字符串将复制源字符串添加。

提示: 如果只想删除陈旧的翻译键，那么使用清理翻译文件。

参见:

Weblate 除了更新翻译，还更新翻译文件吗？

自定义 XML 输出

在 4.15 版本加入.

附加组件 ID
`weblate.xml.customize`

配置

`closing_tags`	包括空白 XML 标记的结束标记

触发器
存储延迟加载

允许调整 XML 输出行为，例如结束标记而不是空标记的自关闭标记。

自定义 YAML 输出

在 3.10.2 版本加入.

附加组件 ID
`weblate.yaml.customize`

配置

`indent`	YAML 缩进	
`width`	自动换行	可用选项: `80` –在 80 个字符处换行 `100` –在 100 个字符处换行 `120` –在 120 个字符处换行 `180` –在 180 个字符处换行 `65535` –不换行
`line_b`	换行符	可用选项: `dos` –DOS (\r\n) `unix` –UNIX (\n) `mac` –MAC (\r)

触发器
存储延迟加载

允许调整 YAML 的输出，例如自定义缩进或自定义换行。

2.14.2 定制附加组件列表

附加组件列表由*WEBLATE_ADDONS*配置。要添加其他附加组件，只需在这个设置中包含绝对类名称即可。

2.14.3 编写附加组件

你也可以编写自己的附加组件，创建一个*weblate.addons.base.BaseAddon*的子类来定义附加组件元数据，然后实现一个回调来执行处理。

参见:

开发附加组件

2.14.4 从附加组件执行脚本

附加组件还可以用于执行外部脚本。这曾经集成在 Weblate 中，但现在必须写一些代码，将脚本包裹在附加组件中。

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

"""Example pre commit script."""


from django.utils.translation import gettext_lazy as _

from weblate.addons.events import EVENT_PRE_COMMIT
from weblate.addons.scripts import BaseScriptAddon


class ExamplePreAddon(BaseScriptAddon):
    # Event used to trigger the script
    events = (EVENT_PRE_COMMIT,)
    # Name of the addon, has to be unique
    name = "weblate.example.pre"
    # Verbose name and long description
    verbose = _("Execute script before commit")
    description = _("This add-on executes a script.")

    # Script to execute
    script = "/bin/true"
    # File to add in commit (for pre commit event)
    # does not have to be set
    add_file = "po/{{ language_code }}.po"
```

安装方法请参见*定制的质量检查、附加组件和自动修复*。

对于任何给定的部件，当前路径设置为版本控制系统（VCS）仓库的根目录时，执行脚本。

此外，以下环境变量是可用的：

WL_VCS
使用的版本控制系统。

WL_REPO
上游仓库 URL。

WL_PATH
版本控制系统（VCS）仓库的绝对路径。

WL_BRANCH
在 2.11 版本加入.

当前部件配置的仓库分支。

WL_FILEMASK
当前部件的文件掩码。

WL_TEMPLATE
单语言翻译模板的文件名（可以为空）。

WL_NEW_BASE
在 2.14 版本加入.

建立新的翻译所使用文件的文件名（可以为空）。

WL_FILE_FORMAT
用于当前部件的文件格式。

WL_LANGUAGE
当前处理的翻译的语言（对于部件级别的钩子不可用）。

WL_PREVIOUS_HEAD
更新后的上个 HEAD (仅在运行更新后钩子后可用)。

WL_COMPONENT_SLUG
在 3.9 版本加入.

部件标识串用于构建 URL。

WL_PROJECT_SLUG
在 3.9 版本加入.

项目标识串用于构建 URL。

WL_COMPONENT_NAME
在 3.9 版本加入.

部件名称。

WL_PROJECT_NAME
在 3.9 版本加入.

项目名称。

WL_COMPONENT_URL
在 3.9 版本加入.

部件 URL。

WL_ENGAGE_URL
在 3.9 版本加入.

项目参与 URL。

参见:

部件配置

更新后仓库处理

可用于在 VCS 上游源更改时更新翻译文件。为了实现这个功能，请记住 Weblate 只能看到提交给版本控制系统（VCS）的文件，所以您需要将更改作为脚本的一部分提交。

例如，使用 Gulp，您可以使用以下代码来实现：

```
#! /bin/sh
gulp --gulpfile gulp-i18n-extract.js
git commit -m 'Update source strings' src/languages/en.lang.json
```

翻译的预提交处理

在将翻译提交到仓库之前，使用 commit 脚本自动更改翻译。

它作为组成当前翻译文件名的单一参数而通过。

2.15 翻译记忆库

在 2.20 版本加入.

Weblate 带有内建的翻译记忆库，包括下面的：

- 手动导入翻译记忆库（请参见*用户界面*）。
- 自动存储 Weblate 中进行的翻译（取决于*翻译记忆库的范围*）。
- 自动导入以前的翻译。

翻译记忆库中的内容可以以两种方式之一来应用：

- 手动，*自动建议* 当翻译时查看。
- 自动，通过使用*自动翻译* 或*自动翻译* 附加组件来翻译字符串。

对于安装提示，请参见*Weblate 翻译记忆库*，它默认是打开的。

2.15.1 翻译记忆库的范围

在 3.2 版本加入: 在较早的版本中，翻译记忆库只能从相应于当前导入的翻译记忆库范围的文件中加载。

翻译记忆库的范围这样允许私有或译者共享，来适应所需要的行为。

导入翻译记忆库

使用 *import_memory* 命令导入任意翻译记忆库数据，使记忆的内容可用于所有的用户和项目。

每名用户的翻译记忆库

在每个单独用户的个人翻译记忆库中自动存储用户的翻译。

每个项目的翻译记忆库

项目内的所有翻译都自动存储在项目翻译记忆库中，这个翻译记忆库只在项目内可用。

共享翻译记忆库

翻译记忆库分享打开的项目的所有翻译，都存储在分享的翻译记忆库中，可用于所有项目。

对于分享的 Weblate 安装，请仔细考虑是否打开这个特性，因为可能导致严重的影响：

- 翻译可以被任何人使用。
- 这会导致泄露秘密信息。

2.15.2 管理翻译记忆库

用户界面

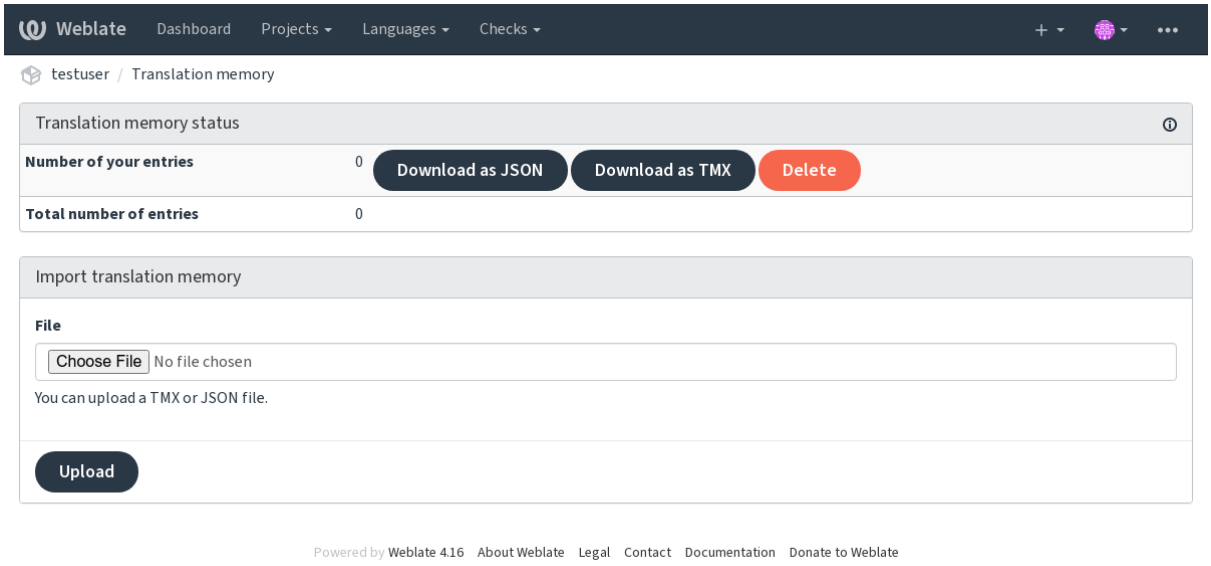
在 3.2 版本加入.

在基本上用户界面上，可以管理每名用户每个项目的项目翻译记忆库。它可以用于下载、消除或导入翻译记忆库。

提示： JSON 的翻译记忆库可以导入 Weblate，提供了 TMX 与其他工具进行互操作。

参见：

Weblate 翻译记忆库概要



管理界面

有几个管理命令来操作翻译记忆库的内容。它们整体操作翻译记忆库，不会被范围来筛选（除非被参数请求）：

dump_memory
将记忆库导入 JSON

import_memory
将 TMX 或 JSON 文件导入翻译记忆库

2.16 配置

所有的设置都存储在 `settings.py` 中（就像 Django 通常那样）。

备注： 更改这些设置中的任意一项后，您需要重新启动 Weblate——包括 WSGI 和 Celery 进程。

在它作为 `mod_wsgi` 运行的情况下，需要重新启动 Apache，来重新加载配置。

参见：

另请查阅 Django 的文档，了解配置 Django 自身的参数。

2.16.1 AKISMET_API_KEY

Weblate 可以使用 Akismet 检查传入的匿名建议是否为垃圾。请访问 akismet.com 来购买 API 密钥，并将它与网站关联。

2.16.2 ANONYMOUS_USER_NAME

未登录用户的用户名。

参见：

访问控制

2.16.3 AUDITLOG_EXPIRY

在 3.6 版本加入.

Weblate 应该将审计日志保存多少天，审计日志包括了账户活动的信息。

默认为 180 天。

2.16.4 AUTH_LOCK_ATTEMPTS

在 2.14 版本加入.

应用速率限制之前，认证尝试失败的最大次数。

当前，这应用在后面的位置：

- 登录。删除账户密码，防止用户不请求新的密码而登录。
- 密码重置。防止发出新的电子邮件，避免向用户发出太多密码重置尝试的垃圾电子邮件。

默认为 10。

参见:

频次限制

2.16.5 AUTO_UPDATE

在 3.2 版本加入.

在 3.11 版本发生变更: 更改原来的开关选项，来区别接受哪条字符串。

以每天的频率更新所有仓库。

提示: 在不使用通知钩子 来自动更新 Weblate 仓库的情况下有用。

备注: 除了字符串选项还存在开关选项，用于向后兼容。

选项有:

"none"
不进行每日更新。

"remote" 也是 False
只进行远程更新。

"full" 也是 True
更新远程，并合并工作副本。

备注: 这需要使用 *Celery* 的后台任务 工作，并在重启后生效。

2.16.6 AVATAR_URL_PREFIX

构成头像 URL 的前缀为: `${AVATAR_URL_PREFIX}/avatar/${MAIL_HASH}?${PARAMS}`。已知后面的服务工作:

Gravatar（默认），根据 https://gravatar.com/

```
AVATAR_URL_PREFIX = 'https://www.gravatar.com/'
```

Libravatar，根据 https://www.libravatar.org/

```
AVATAR_URL_PREFIX = 'https://www.libravatar.org/'
```

参见:

头像缓存, *ENABLE_AVATARS*, 头像

2.16.7 AUTH_TOKEN_VALID

在 2.14 版本加入.

身份验证令牌和密码重置电子邮件中临时密码的有效时间。以秒为单位，默认为 172800（2 天）。

2.16.8 AUTH_PASSWORD_DAYS

在 2.15 版本加入.

Weblate 拒绝用户重复使用之前用过的密码的情况会持续多少天。

检查是基于审计日志进行的，*AUDITLOG_EXPIRY* 需要至少和这个一样。

备注： Weblate 2.15 版本之前的密码更改不遵从这个原则。

默认为 180 天。

2.16.9 AUTOFIX_LIST

当存储字符串时应用自动修复列表。

备注： 为应用自动修复见面的 Python 类提供完全合规的路径。

可用的修复：

weblate.trans.autofixes.whitespace.SameBookendingWhitespace
将字符串开头和结尾处的空格与原文匹配。

weblate.trans.autofixes.chars.ReplaceTrailingDotsWithEllipsis
替换连续的点 (…)，如果源字符串有一个对应的省略号 (…)。

weblate.trans.autofixes.chars.RemoveZeroSpace
如果原文不包含任何零宽空格字符，则删除零宽空格字符。

weblate.trans.autofixes.chars.RemoveControlChars
如果原文不包含任何控制字符，则删除控制字符。

weblate.trans.autofixes.html.BleachHTML
从标记为 `safe-html` 的字符串中去掉不安全的 HTML 标记（请参见不安全的 *HTML* ）。

可以选择使用哪一个：

```
AUTOFIX_LIST = (
    "weblate.trans.autofixes.whitespace.SameBookendingWhitespace",
    "weblate.trans.autofixes.chars.ReplaceTrailingDotsWithEllipsis",
)
```

参见：

自动修正, 定制的自动修正

2.16.10 BACKGROUND_TASKS

在 4.5.2 版本加入.

定义应为部件触发冗长维护任务的频率。

现在这个控件:

- 自动翻译 附加组件
- 检查和修正 的重新计算

可能的选项：

- `monthly` （这是默认设置）
- `weekly`

- daily
- never

备注： 当 Weblate 包含数千个部件时，不建议增加频率。

2.16.11 BASIC_LANGUAGES

在 4.4 版本加入.

提供给用户开始新的翻译的语言列表。当未指定时，使用内建列表，其中包括所有常用的语言，但没有特定国家/地区的变体。

这只是限制了非特权用户将不想要的语言添加进来。项目管理员仍然被给出 Weblate 中定义的所有语言选择。

备注： 这对 Weblate 并不定义新的语言，它只在数据库中筛选了现有的那些。

示例：

```
BASIC_LANGUAGES = {"cs", "it", "ja", "en"}
```

参见：

语言定义

2.16.12 BORG_EXTRA_ARGS

在 4.9 版本加入.

当内置备份被触发时，你可以将额外的参数传递给 **borg create**。

示例：

```
BORG_EXTRA_ARGS = ["--exclude", "vcs/"]
```

参见：

备份和移动 *Weblate*, borg create

2.16.13 CACHE_DIR

在 4.16 版本加入.

Weblate 存放缓存文件的目录。默认为*DATA_DIR* 中的 cache 子目录。

将此更改为本地或临时文件系统，如果*DATA_DIR* 位于网络文件系统上。

Docker 容器为这个使用了一个单独的卷，请参阅*Docker 容器卷*。

2.16.14 CSP_SCRIPT_SRC, CSP_IMG_SRC, CSP_CONNECT_SRC, CSP_STYLE_SRC, CSP_FONT_SRC

为 Weblate 定制 `Content-Security-Policy` 标头。根据允许集成的第三方服务（Matomo、Google Analytics、Sentry……）来自动生成标头。

这些默认为空列表。

示例：

```
# Enable Cloudflare Javascript optimizations
CSP_SCRIPT_SRC = ["ajax.cloudflare.com"]
```

参见：

内容安全政策, 内容安全策略 (CSP)

2.16.15 CHECK_LIST

翻译时执行的质量检查列表。

备注： 提供一个实现了检查接口的 Python 类的完全合规路径。

调整检查列表，来包括与你相关的那些检查。

所有内建的质量检查 默认都打开，可以从那里更改设置。它们在配置的示例 中被默认注释掉，从而使用默认值。然后每个新的 Weblate 版本执行新的检查。

可以关闭所有检查：

```
CHECK_LIST = ()
```

可以只打开一部分检查：

```
CHECK_LIST = (
    "weblate.checks.chars.BeginNewlineCheck",
    "weblate.checks.chars.EndNewlineCheck",
    "weblate.checks.chars.MaxLengthCheck",
)
```

备注： 更改这些设置只影响新更改的翻译，现存的检查仍然存储在数据库中。为了将更改同样应用到存储的翻译中，运行*updatechecks*。

参见：

质量检查, 使用标记定制行为

2.16.16 COMMENT_CLEANUP_DAYS

在 3.6 版本加入.

在给定天数后删除评论。默认为 `None`，表示不删除。

2.16.17 COMMIT_PENDING_HOURS

在 2.10 版本加入.

通过后台任务方式提交待处理更改之间的小时数。

参见:

部件配置, 对更改进行提交的延时时间, 运行维护任务, *commit_pending*

2.16.18 CONTACT_FORM

在 4.6 版本加入.

配置来自联系表单的电子邮件的发送方式。选择与您的电子邮件服务器配置相匹配的配置。

"reply-to"
发送者在 *Reply-To* 中使用，这是默认行为。

"from"
发件人为:mailheader:*From*, 你的电子邮箱服务器需要允许发送此类电子邮件。

2.16.19 DATA_DIR

Weblate 存储所有数据的文件夹。它包含指向版本控制系统（VCS）仓库、全文索引和外部工具的各种配置文件。

后面的子目录通常存在:

home
Home 目录用于调用脚本。

ssh
SSH 密钥和配置。

static
静态 Django 文件的默认位置，由 `django:STATIC_ROOT`指定。见:ref:`static-files。

Docker 容器为这个使用了一个单独的卷，请参阅*Docker 容器卷*。

media
Django 媒体文件的默认位置，由 `django:MEDIA_ROOT`指定。包含已上传的截屏，见:ref:`screenshots。

vcs
翻译的版本控制仓库。

backups
每日备份数据，详情请查阅下载的数据用于备份。

fonts:
用户上传的字体，参见管理字型。

cache
可使用:setting:‘CACHE_DIR‘将多个缓存置于别处。

Docker 容器为这个使用了一个单独的卷，请参阅*Docker 容器卷*。

备注: 这个目录必须由 Weblate 写入。运行作为 uWSGI 意味着 `www-data` 用户应该对它具有写入权限。

实现这个的最简单方式是使用户作为目录的所有者:

```
sudo chown www-data:www-data -R $DATA_DIR
```

默认为 /home/weblate/data, 但预计将进行配置。

参见:

文件系统权限, 备份和移动 *Weblate*, *CACHE_DIR*

2.16.20 DATABASE_BACKUP

在 3.1 版本加入.

数据库备份应该存储为纯文本，压缩还是跳过。授权值为:

- "plain"
- "compressed"
- "none"

参见:

备份和移动 *Weblate*

2.16.21 DEFAULT_ACCESS_CONTROL

在 3.3 版本加入.

新项目的默认访问控制设置:

0
: 公开的

1
: 受保护的

100
: 私有的

200
: 自定义

如果手动管理 ACL 则使用 自定义，这意味着不依赖于 Weblate 内部管理。

参见:

项目访问控制, 访问控制

2.16.22 DEFAULT_AUTO_WATCH

在 4.5 版本加入.

配置是否应该为新用户打开 自动关注作出贡献的项目。默认为 True 。

参见:

通知

2.16.23 DEFAULT_RESTRICTED_COMPONENT

在 4.1 版本加入.

部件限制的默认值。

参见:

受限制的访问, 团队范围

2.16.24 DEFAULT_ADD_MESSAGE, DEFAULT_ADDON_MESSAGE, DEFAULT_COMMIT_MESSAGE, DEFAULT_DELETE_MESSAGE, DEFAULT_MERGE_MESSAGE

不同操作的默认提交说明，请查阅部件配置 了解详情。

参见:

模板标记, 部件配置, 提交、添加、删除、合并、附加组件及合并请求说明

2.16.25 DEFAULT_ADDONS

安装在每个创建的部件上的默认附加组件。

备注: 此设置只影响新创建的部件。

示例:

```
DEFAULT_ADDONS = {
    # Add-on with no parameters
    "weblate.flags.target_edit": {},
    # Add-on with parameters
    "weblate.autotranslate.autotranslate": {
        "mode": "suggest",
        "filter_type": "todo",
        "auto_source": "mt",
        "component": "",
        "engines": ["weblate-translation-memory"],
        "threshold": "80",
    },
}
```

参见:

install_addon, 附加组件, *WEBLATE_ADDONS*

2.16.26 DEFAULT_COMMITER_EMAIL

在 2.4 版本加入.

提交者电子邮箱地址默认为 `noreply@weblate.org`。

参见:

DEFAULT_COMMITER_NAME

2.16.27 DEFAULT_COMMITER_NAME

在 2.4 版本加入.

提交者姓名默认为 `Weblate`。

参见:

DEFAULT_COMMITER_EMAIL

2.16.28 DEFAULT_LANGUAGE

在 4.3.2 版本加入.

例如在源语言 中使用的默认源语言。

默认为 *en*。匹配的语言对象需要存在于数据库中。

参见:

语言定义, 源语言

2.16.29 DEFAULT_MERGE_STYLE

在 3.4 版本加入.

任何新部件的合并风格。

- *rebase* - default
- *merge*

参见:

部件配置, 合并方式

2.16.30 DEFAULT_SHARED_TM

在 3.2 版本加入.

配置使用共享的翻译记忆库 和贡献到共享的翻译记忆库 的默认值。

2.16.31 DEFAULT_TRANSLATION_PROPAGATION

在 2.5 版本加入.

翻译传播的默认设置，默认为 `True`。

参见:

部件配置, 允许同步翻译

2.16.32 DEFAULT_PULL_MESSAGE

配置拉取请求的默认标题和说明。

2.16.33 ENABLE_AVATARS

是否为用户开启基于 Gravatar 的头像。默认开启。

头像取出并缓存在从服务器中，降低了泄漏个人信息的风险，加速了用户体验。

参见：

头像缓存, *AVATAR_URL_PREFIX*, *头像*

2.16.34 ENABLE_HOOKS

是否允许匿名远程钩子。

参见：

通知钩子

2.16.35 ENABLE_HTTPS

将链接作为 HTTPS 还是 HTTP 发送给 Weblate。这个设置影响发送电子邮件，并产生绝对 URL。

在默认配置中，这还用于与 HTTPS 相关的几个 Django 设置——启用安全 Cookie、切换 HSTS 或允许重定向到 HTTPS URL。

在一些情况下 HTTPS 重定向会有问题，您可以在使用反向代理进行 SSL 终端连接的情况下碰到无限重定向的问题，使用反向代理进行 SSL 终端连接不能将协议标头正确地传递给 Django。请调整自己的反向代理配置，去掉 `X-Forwarded-Proto` 或 `Forwarded` 标头，或者配置 `SECURE_PROXY_SSL_HEADER`，让 Django 正确地检测 SSL 状态。

参见：

`SESSION_COOKIE_SECURE`, `CSRF_COOKIE_SECURE`, `SECURE_SSL_REDIRECT`, `SECURE_PROXY_SSL_HEADER` *设置正确的网站域名*

2.16.36 ENABLE_SHARING

打开/关闭 分享菜单，使用户能够将翻译过程分享到社交网络上。

2.16.37 EXTRA_HTML_HEAD

在 4.15 版本加入.

将附加标记插入 HTML 标头。可用于验证站点所有权，示例：

```
EXTRA_HTML_HEAD = '<link href="https://fosstodon.org/@weblate" rel="me">'
```

警告： 不对字符串进行整理，它按原样插入到 HTML 标头中。

2.16.38 GET_HELP_URL

在 4.5.2 版本加入.

可以找到你的 Weblate 实例支持的 URL。

2.16.39 GITEA_CREDENTIALS

在 4.12 版本加入.

Gitea 服务器凭据列表。

```
GITEA_CREDENTIALS = {
    "try.gitea.io": {
        "username": "weblate",
        "token": "your-api-token",
    },
    "gitea.example.com": {
        "username": "weblate",
        "token": "another-api-token",
    },
}
```

参见:

Gitea 拉取请求, 创建 Gitea 个人访问令牌

2.16.40 GITLAB_CREDENTIALS

在 4.3 版本加入.

用于 GitLab 服务器的证明列表。

```
GITLAB_CREDENTIALS = {
    "gitlab.com": {
        "username": "weblate",
        "token": "your-api-token",
    },
    "gitlab.example.com": {
        "username": "weblate",
        "token": "another-api-token",
    },
}
```

参见:

GitLab 合并请求, GitLab: 个人访问令牌

2.16.41 GITHUB_CREDENTIALS

在 4.3 版本加入.

用于 GitHub 服务器的证明列表。

```
GITHUB_CREDENTIALS = {
    "api.github.com": {
        "username": "weblate",
        "token": "your-api-token",
    },
    "github.example.com": {
```

(续下页)

(接上页)

```
        "username": "weblate",
        "token": "another-api-token",
    },
}
```

参见：

GitHub 拉取请求, 创建 GitHub 个人访问令牌

2.16.42 BITBUCKETSERVER_CREDENTIALS

在 4.16 版本加入.

Bitbucket 服务器凭据列表。

```
BITBUCKETSERVER_CREDENTIALS = {
    "git.self-hosted.com": {
        "username": "weblate",
        "token": "http-access-token",
    },
}
```

参见：

Bitbucket 服务器拉取请求, Bitbucket: HTTP 访问令牌

2.16.43 GOOGLE_ANALYTICS_ID

使用 Google Analytics（分析）时，开启对 Weblate 的监测的 Google Analytics（分析）ID。

2.16.44 HIDE_REPO_CREDENTIALS

隐藏仓库凭据避免出现在 web 界面中。在仓库 URL 带有用户名和密码的情况下，Weblate 会在相关信息显示给用户时将其隐藏起来。

例如，它将显示 `https://git.example.com/repo.git` 不会显示 `https://user:password@git.example.com/repo.git`。它也尝试以相似的方式清除版本控制系统（VCS）错误信息。

备注： 默认这是打开的。

2.16.45 HIDE_VERSION

在 4.3.1 版本加入.

对未经身份验证用户隐藏版本信息。这也会使所有文档链接都指向最新的版本，而不是与当前安装版本相匹配的文档。

一些公司推荐在安全实践上推荐隐藏版本，但不能防止攻击者通过试探性为来找出版本。

备注： 默认这是关闭的。

2.16.46 INTERLEDGER_PAYMENT_POINTERS

在 4.12.1 版本加入.

用于网络货币化的分类账间支付指针（ILP）列表。

如果指定了多个，则通过随机选择一个来实现概率收入共享。

请查阅 <https://webmonetization.org/> 了解更多细节。

提示： 默认值让用户为 Weblate 自身提供资金。

2.16.47 IP_BEHIND_REVERSE_PROXY

在 2.14 版本加入.

指示 Weblate 是否在反向代理之后运行。

如果设置为 `True`，Weblate 会从*IP_PROXY_HEADER* 定义的标头中获取 IP 地址。

警告： 确保真正使用反向代理，并且设置了这个标头，否则用户将能够假冒 IP 地址。

备注： 默认这不是打开的。

参见：

在反向代理之后运行, *频次限制*, *IP_PROXY_HEADER*, *IP_PROXY_OFFSET*

2.16.48 IP_PROXY_HEADER

在 2.14 版本加入.

指示当*IP_BEHIND_REVERSE_PROXY* 打开时，Weblate 应该从那个标头得到 IP 地址。

默认为 `HTTP_X_FORWARDED_FOR`。

参见：

在反向代理之后运行, *频次限制*, `SECURE_PROXY_SSL_HEADER`, *IP_BEHIND_REVERSE_PROXY*, *IP_PROXY_OFFSET*

2.16.49 IP_PROXY_OFFSET

在 2.14 版本加入.

指示*IP_PROXY_HEADER* 的哪部分用作客户端 IP 地址。

取决于你的设置，这个标头会包括几个 IP 地址，（例如 `X-Forwarded-For: a, b, client-ip`），并且可以配置标头的哪个地址在这里用作客户端 IP 地址。

警告： 设置这个会影响你安装的安全性，应该只配置它来使用信任的代理来确定 IP 地址。

默认为 0。

参见：

在反向代理之后运行, *频次限制*, SECURE_PROXY_SSL_HEADER, *IP_BEHIND_REVERSE_PROXY*, *IP_PROXY_HEADER*

2.16.50 LEGAL_TOS_DATE

在 4.15 版本加入.

备注： 你需要安装*法律声明* 才能让其生效。

服务条款文件的最后更新日期。每当日期更改时，用户都需要同意服务条款。

```
from datetime import date

LEGAL_TOS_DATE = date(2022, 2, 2)
```

2.16.51 LEGAL_URL

在 3.5 版本加入.

你的 Weblate 实例显示其法律文件的 URL。

提示： 在将你的法律文件保存在 Weblate 之外，而将其嵌入 Weblate 的情况下有用。细节请查看*法律声明*。

示例：

```
LEGAL_URL = "https://weblate.org/terms/"
```

参见：

PRIVACY_URL

2.16.52 LICENSE_EXTRA

包括在许可选择中的其他许可。

备注： 每个许可的定义应该是其短名称、长名称和 URL 的元组。

例如：

```
LICENSE_EXTRA = [
    (
        "AGPL-3.0",
        "GNU Affero General Public License v3.0",
        "https://www.gnu.org/licenses/agpl-3.0-standalone.html",
    ),
]
```

2.16.53 LICENSE_FILTER

在 4.3 版本发生变更: 现在将此设置为空值可禁用许可证警报。

筛选要显示的许可证列表。将此设置为空可禁用许可证警报。

备注: 这个过滤器使用简短的许可证名称。

例如:

```
LICENSE_FILTER = {"AGPL-3.0", "GPL-3.0-or-later"}
```

以下为禁用许可证警报的方法:

```
LICENSE_FILTER = set()
```

参见:

alerts

2.16.54 LICENSE_REQUIRED

定义了是否需要部件配置 中的许可属性。

备注: 默认这是关闭的。

2.16.55 LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH

是否应限制给定翻译的长度。限制为源字符串的长度 × 10 个字符。

提示: 将其设置为 `False` 即可允许更长的翻译（最多 10,000 个字符），而不管源字符串的长度。

备注: 默认为 `True`。

2.16.56 LOCALIZE_CDN_URL 和 LOCALIZE_CDN_PATH

这些设置配置*JavaScript 本地化 CDN* 附加组件。*LOCALIZE_CDN_URL* 定义了本地化 CDN 可用的根 URL，而*LOCALIZE_CDN_PATH* 定义了 Weblate 应把*LOCALIZE_CDN_URL* 将提供的生成文件存储在哪个路径。

提示: 在 Hosted Weblate 中，这使用 `https://weblate-cdn.com/`。

参见:

JavaScript 本地化 CDN

2.16.57 LOGIN_REQUIRED_URLS

你希望需要登录的 URL 列表。（除了 Weblate 内建立的标准规则）。

提示： 这允许密码保护整个安装，通过使用：

```
LOGIN_REQUIRED_URLS = (r"/(.*)$",)
REST_FRAMEWORK["DEFAULT_PERMISSION_CLASSES"] = [
    "rest_framework.permissions.IsAuthenticated"
]
```

提示： 同样想要锁住 API 访问，如上面的示例所示。

参见：

REQUIRE_LOGIN

2.16.58 LOGIN_REQUIRED_URLS_EXCEPTIONS

用于*LOGIN_REQUIRED_URLS* 的例外列表。如果未指定，则允许用户访问登录页。

你可能想要包括的一些例外：

```
LOGIN_REQUIRED_URLS_EXCEPTIONS = (
    r"/accounts/(.*)$",  # Required for sign in
    r"/static/(.*)$",  # Required for development mode
    r"/widgets/(.*)$",  # Allowing public access to widgets
    r"/data/(.*)$",  # Allowing public access to data exports
    r"/hooks/(.*)$",  # Allowing public access to notification hooks
    r"/api/(.*)$",  # Allowing access to API
    r"/js/i18n/$",  # JavaScript localization
)
```

2.16.59 MATOMO_SITE_ID

你想要跟踪的 Matomo（以前的 Piwik）中的网站 ID。

备注： 这个集成不支持 Matomo Tag Manager。

参见：

MATOMO_URL

2.16.60 MATOMO_URL

你想用于跟踪 Weblate 使用情况的 Matomo（以前的 Piwik）安装的完整 URL（包括结尾的斜杠）。请查阅 <https://matomo.org/> 了解详情。

提示： 这个集成不支持 Matomo Tag Manager。

例如：

```
MATOMO_SITE_ID = 1
MATOMO_URL = "https://example.matomo.cloud/"
```

参见:

MATOMO_SITE_ID

2.16.61 NEARBY_MESSAGES

在查看当前翻译字符串时显示多少字符串。这只是默认值，用户可以在:ref:‘user-profile‘中调整。

2.16.62 DEFAULT_PAGE_LIMIT

在 4.7 版本加入.

分页激活时显示的默认元素数量。

2.16.63 PAGURE_CREDENTIALS

在 4.3.2 版本加入.

Pagure 服务器凭据列表。

```
PAGURE_CREDENTIALS = {
    "pagure.io": {
        "username": "weblate",
        "token": "your-api-token",
    },
    "pagure.example.com": {
        "username": "weblate",
        "token": "another-api-token",
    },
}
```

参见:

Pagure 合并请求, Pagure API

2.16.64 PRIVACY_URL

在 4.8.1 版本加入.

你的 Weblate 实例显示隐私政策的 URL。

提示: 在将你的法律文件保存在 Weblate 之外，而将其嵌入 Weblate 的情况下有用。细节请查看*法律声明*。

示例:

```
PRIVACY_URL = "https://weblate.org/terms/"
```

参见:

LEGAL_URL

2.16.65 PRIVATE_COMMIT_EMAIL_OPT_IN

在 4.15 版本加入.

配置专用提交电子邮箱是选择加入还是选择退出（默认情况下是选择退出）。

参见:

个人资料, *PRIVATE_COMMIT_EMAIL_TEMPLATE*

2.16.66 PRIVATE_COMMIT_EMAIL_TEMPLATE

在 4.15 版本加入.

为用户生成专用提交电子邮箱的模板。默认为 `"{username}@users.noreply.{site_domain}"`.

留空禁用该功能。

备注: 除非通过*PRIVATE_COMMIT_EMAIL_OPT_IN* 配置，否则用户可以选择使用不同的提交电子邮件。用户可以在个人资料 中配置提交电子邮件。

2.16.67 PROJECT_BACKUP_KEEP_COUNT

在 4.14 版本加入.

定义每个项目在服务器上保留多少备份，它默认为 3。

参见:

项目级别备份

2.16.68 PROJECT_BACKUP_KEEP_DAYS

在 4.14 版本加入.

定义项目备份将在服务器上保留多长时间，默认值为 30 天。

参见:

项目级别备份

2.16.69 PROJECT_NAME_RESTRICT_RE

在 4.15 版本加入.

定义用于限制项目命名的正则表达式。任何匹配的名称都将被拒绝。

参见:

项目名称

2.16.70 PROJECT_WEB_RESTRICT_HOST

在 4.16.2 版本加入.

在项目网站中拒绝使用特定的主机。匹配任意子域名，因此包含 `example.com` 会同样拦截 `test.example.com`. 此列表只应包含小写字符串，进行匹配前，解析的域名名称会转为小写。

默认配置：

```
PROJECT_WEB_RESTRICT_HOST = {"localhost"}
```

参见：

项目网站 PROJECT_WEB_RESTRICT_NUMERIC, *PROJECT_WEB_RESTRICT_RE*,

2.16.71 PROJECT_WEB_RESTRICT_NUMERIC

在 4.16.2 版本加入.

拒绝在项目网站中使用数字 IP 地址。默认启用。

参见：

项目网站 PROJECT_WEB_RESTRICT_HOST, *PROJECT_WEB_RESTRICT_RE*,

2.16.72 PROJECT_WEB_RESTRICT_RE

在 4.15 版本加入.

定义用于限制项目网站的正则表达式。任何匹配的 URL 都将被拒绝。

参见：

项目网站 PROJECT_WEB_RESTRICT_HOST, *PROJECT_WEB_RESTRICT_NUMERIC*

2.16.73 RATELIMIT_ATTEMPTS

在 3.2 版本加入.

在应用次数限制前，身份认证尝试的最多次数。

默认为 5。

参见：

频次限制, *RATELIMIT_WINDOW*, *RATELIMIT_LOCKOUT*

2.16.74 RATELIMIT_WINDOW

在 3.2 版本加入.

应用次数限制后，可接受多少次身份认证。

秒数默认为 300 （5 分钟）。

参见：

频次限制, *RATELIMIT_ATTEMPTS*, *RATELIMIT_LOCKOUT*

2.16.75 RATELIMIT_LOCKOUT

在 3.2 版本加入.

在应用次数限制后，身份认证锁定多久。

秒数默认为 600（10 分钟）。

参见:

频次限制, *RATELIMIT_ATTEMPTS*, *RATELIMIT_WINDOW*

2.16.76 REGISTRATION_ALLOW_BACKENDS

在 4.1 版本加入.

身份验证后端的列表，允许从中注册。这只限制新的注册，用户可以使用配置的身份验证后端，来进行身份验证和添加身份验证。

当限制注册后端时，推荐保持*REGISTRATION_OPEN* 为开启状态，否则用户将能够注册，但 Weblate 不会在用户界面显示注册的链接。

示例:

```
REGISTRATION_ALLOW_BACKENDS = ["azuread-oauth2", "azuread-tenant-oauth2"]
```

提示: 与身份验证 URL 中使用的名称匹配的后端名称。

参见:

REGISTRATION_OPEN, 身份验证

2.16.77 REGISTRATION_CAPTCHA

`True` 或 `False` 的值指示新账户注册是否被 CAPTCHA 保护。这个设置是可选的，默认的 `True` 是假定不提供。

如果打开，CAPTCHA 会添加到用户输入其电子邮箱地址的所有页面中:

- 新账户注册。
- 找回密码。
- 将电子邮箱地址添加到账户中。
- 供未登录用户使用的联系表单。

2.16.78 REGISTRATION_EMAIL_MATCH

在 2.17 版本加入.

允许你筛选哪个电子邮箱地址可以注册。

默认为 `.*`，允许使用任何电子邮箱地址注册。

可以用它限制注册到单一的电子邮箱域名:

```
REGISTRATION_EMAIL_MATCH = r"^.*@weblate\.org$"
```

2.16.79 REGISTRATION_OPEN

是否注册新账户在当前是允许的。这个可选设置可以保持默认为 `True`，或更改为 `False`。

这个设置影响了内建的通过电子邮箱地址或通过 Python 社交认证的身份验证（可以使用*REGISTRATION_ALLOW_BACKENDS* 为适当的后端建立白名单）。

备注： 如果使用第三方身份验证方法，如*LDAP 身份验证*，只是隐藏注册表单，而新用户仍然能够登录并建立账户。

参见：

REGISTRATION_ALLOW_BACKENDS, *REGISTRATION_EMAIL_MATCH*, *身份验证*

2.16.80 REGISTRATION_REBIND

在 4.16 版本加入.

允许为现有用户重新绑定身份验证后端。在不同身份验证提供者之间迁移时请打开此选项。

备注： 默认禁用，不允许添加其他身份验证后端到现有账户。使用更多第三方身份验证后端时，重新绑定可导致账户沦陷。

2.16.81 REPOSITORY_ALERT_THRESHOLD

在 4.0.2 版本加入.

触发仓库过期警报的阈值，或者包含了太多更改的阈值。默认为 25。

参见：

alerts

2.16.82 REQUIRE_LOGIN

在 4.1 版本加入.

这会启用*LOGIN_REQUIRED_URLS* 并配置 REST 框架来对所有 API 端点都要求登录。

备注： 这实现在 sample-configuration‘中。对于 Docker，使用:envvar:‘WEBLATE_REQUIRE_LOGIN。

2.16.83 SENTRY_DSN

在 3.9 版本加入.

Sentry DSN 用于收集错误报告。

参见：

Sentry 的 Django 集成

2.16.84 SESSION_COOKIE_AGE_AUTHENTICATED

在 4.3 版本加入.

对身份已验证用户设置会话过期时间。这是对用于未经身份验证用户的 SESSION_COOKIE_AGE 的补充。

参见:

SESSION_COOKIE_AGE

2.16.85 SIMPLIFY_LANGUAGES

对于默认语言/国家组合使用简单语言代码。例如，fr_FR 翻译将使用 fr 语言代码。这通常是受欢迎的特性，因为它简化了这些默认群组合列出的语言。

如果对每种不同的变化想要不同的翻译，那么请将其关闭。

2.16.86 SITE_DOMAIN

配置网站域名。这对于在很多地方获得正确的绝对链接是必要的（例如激活电子邮箱、通知或 RSS 订阅源）。

在 Weblate 运行在非标准端口时，这里同样要包括它。

示例:

```
# Production site with domain name
SITE_DOMAIN = "weblate.example.com"

# Local development with IP address and port
SITE_DOMAIN = "127.0.0.1:8000"
```

备注: 这个设置应该只包含域名。对于配置协议，（允许或强制 HTTPS ）使用*ENABLE_HTTPS*；对于修改 URL，使用*URL_PREFIX*。

提示: 关于 Docker 容器，网站域名通过 *WEBLATE_ALLOWED_HOSTS* 来配置。

参见:

设置正确的网站域名, 允许主机设置, 正确配置 *HTTPS* *WEBLATE_SITE_DOMAIN*, *ENABLE_HTTPS*

2.16.87 SITE_TITLE

用于网站和发送电子邮件的网站名称。

2.16.88 SPECIAL_CHARS

屏幕键盘中包括的另外的字符，虚拟键盘。

默认值为：

```
SPECIAL_CHARS = ("\t", "\n", "\u00a0", "…")
```

2.16.89 SINGLE_PROJECT

在 3.8 版本加入.

将用户直接重定向到项目或部件，而不是显示操作面板。可以将其设置为 `True`，在这种情况下，只在 Weblate 实际只有单一的项目时有用。另外可以设置项目标识串，将无条件地重定向到这个项目。

在 3.11 版本发生变更: 设置只接受项目标识串，来强制显示那个单一项目。

示例：

```
SINGLE_PROJECT = "test"
```

2.16.90 SSH_EXTRA_ARGS

在 4.9 版本加入.

允许在 Weblate 调用 SSH 时添加自定义参数。这会在连接到使用传统加密或其他非标准功能的服务器时帮助巨大。

例如，当 Weblate 的 SSH 连接失败并提示 'Unable to negotiate with legacyhost: no matching key exchange method found. Their offer: diffie-hellman-group1-sha1'时，你可以用以下方式启用：

```
SSH_EXTRA_ARGS = "-oKexAlgorithms=+diffie-hellman-group1-sha1"
```

提示： 该字符串会用于 shell 计算，因此请确保任何空格和特殊字符都用引号包起来。

参见：

OpenSSH 老旧选项

2.16.91 STATUS_URL

你的 Weblate 实例报告其状态的 URL。

2.16.92 SUGGESTION_CLEANUP_DAYS

在 3.2.1 版本加入.

在给定天数后自动删除建议。默认为 `None`，表示不删除。

2.16.93 UPDATE_LANGUAGES

在 4.3.2 版本加入.

控制在运行数据库迁移时是否应该更新语言数据库，这在默认情况下是启用的。这个设置对*setuplang*的调用没有影响。

警告： 语言显示可能会与此不一致。Weblate 语言定义会随着时间的推移而扩展，并且它不会显示所定义语言的语言代码。

参见：

内置语言定义

2.16.94 URL_PREFIX

这个设置允许在一些路径下运行 Weblate （否则它依赖于从 web 服务器根目录运行）。

备注： 为了使用这个设置，还需要配置服务器来去掉这个前缀。例如 WSGI，可以通过设置 `WSGIScriptAlias` 来实现。

提示： 前缀应该以 / 开始。

示例：

```
URL_PREFIX = "/translations"
```

备注： 这个设置在 Django’s 内建服务器中不起作用，必须调整 `urls.py` 来包含这个前缀。

2.16.95 VCS_API_DELAY

在 4.15.1 版本加入.

在*GitHub 拉取请求* 、*GitLab 合并请求* 、*Gitea 拉取请求* 和*Pagure 合并请求* 中配置第三方 API 调用之间的最小延迟（以秒为单位）。

这个速率限制 Weblate 对这些服务的调用，避免服务过载。

如果你正受限于 Github 的二级限速器，提升这个值可能会有帮助。

默认值为 10。

2.16.96 VCS_BACKENDS

可用的版本控制系统（VCS）后端的配置。

备注： Weblate 尝试使用你所有工具支持的后端。

提示： 可以使用这个来限制选择或添加定制版本控制系统（VCS）后端。

```
VCS_BACKENDS = ("weblate.vcs.git.GitRepository",)
```

参见:

版本控制集成

2.16.97 VCS_CLONE_DEPTH

在 3.10.2 版本加入.

配置 Weblate 应该对仓库进行深度为多少的克隆。

备注: 当前这只在*Git* 中支持。默认 Weblate 进行仓库的浅克隆，使克隆更快并节省磁盘空间。根据应用（例如当使用定制的附加组件 时），你可能想要增加深度，或通过设置为 0 来完全关闭浅克隆。

提示: 在从 Weblate 推送而得到 `fatal: protocol error: expected old/new/ref, got 'shallow <commit hash>'` 错误的情况下，通过设置来完全关闭浅克隆：

```
VCS_CLONE_DEPTH = 0
```

2.16.98 WEBLATE_ADDONS

可供使用的附加组件列表。为了使用，必须为给定的翻译部件启用它们。默认包括了所有内建附加组件，当扩展列表时您可能希望保持现有的附加组件处于启用状态，例如：

```
WEBLATE_ADDONS = (
    # Built-in add-ons
    "weblate.addons.gettext.GenerateMoAddon",
    "weblate.addons.gettext.UpdateLinguasAddon",
    "weblate.addons.gettext.UpdateConfigureAddon",
    "weblate.addons.gettext.MsgmergeAddon",
    "weblate.addons.gettext.GettextCustomizeAddon",
    "weblate.addons.gettext.GettextAuthorComments",
    "weblate.addons.cleanup.CleanupAddon",
    "weblate.addons.consistency.LangaugeConsistencyAddon",
    "weblate.addons.discovery.DiscoveryAddon",
    "weblate.addons.flags.SourceEditAddon",
    "weblate.addons.flags.TargetEditAddon",
    "weblate.addons.flags.SameEditAddon",
    "weblate.addons.flags.BulkEditAddon",
    "weblate.addons.generate.GenerateFileAddon",
    "weblate.addons.json.JSONCustomizeAddon",
    "weblate.addons.xml.XMLCustomizeAddon",
    "weblate.addons.properties.PropertiesSortAddon",
    "weblate.addons.git.GitSquashAddon",
    "weblate.addons.removal.RemoveComments",
    "weblate.addons.removal.RemoveSuggestions",
    "weblate.addons.resx.ResxUpdateAddon",
    "weblate.addons.autotranslate.AutoTranslateAddon",
    "weblate.addons.yaml.YAMLCustomizeAddon",
    "weblate.addons.cdn.CDNJSAddon",
    # Add-on you want to include
    "weblate.addons.example.ExampleAddon",
)
```

备注： 从列表中删除附加组件并不能将其从部件中卸载。Weblate 在这种情况下会崩溃。请先从所有部件中卸载附加组件，然后再将其从此列表中删除。

参见：

附加组件, *DEFAULT_ADDONS*

2.16.99 WEBLATE_EXPORTERS

在 4.2 版本加入.

可用的导出程序列表，提供下载各种文件格式的翻译或术语表。

参见：

支持的文件格式

2.16.100 WEBLATE_FORMATS

在 3.0 版本加入.

可供使用的文件格式列表。

备注： 默认列表已经具有了常见格式。

参见：

支持的文件格式

2.16.101 WEBLATE_MACHINERY

在 4.13 版本加入.

可供使用的机器翻译服务列表。

参见：

配置自动建议

2.16.102 WEBLATE_GPG_IDENTITY

在 3.1 版本加入.

Weblate 对 Git 提交进行签名的身份，例如：

```
WEBLATE_GPG_IDENTITY = "Weblate <weblate@example.com>"
```

在 Weblate GPG 密钥环中搜索匹配的密钥（*DATA_DIR* 下的 `home/.gnupg`）。如果找不到，则会生成一个密钥，请查阅使用 *GnuPG* 为 *Git* 提交签名 了解详情。

参见：

使用 *GnuPG* 为 *Git* 提交签名

2.16.103 WEBSITE_REQUIRED

定义了在创建项目时是否需要指定项目网站。默认打开，因为这适合公共服务器的设置。

2.17 配置的示例

后面的示例作为 `weblate/settings_example.py` 与 Weblate 一起上市：

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

import os
import platform
from logging.handlers import SysLogHandler

# Title of site to use
SITE_TITLE = "Weblate"

# Site domain
SITE_DOMAIN = ""

# Whether site uses https
ENABLE_HTTPS = False

#
# Django settings for Weblate project.
#

DEBUG = True

ADMINS = (
    # ("Your Name", "your_email@example.com"),
)

MANAGERS = ADMINS

DATABASES = {
    "default": {
        # Use "postgresql" or "mysql".
        "ENGINE": "django.db.backends.postgresql",
        # Database name.
        "NAME": "weblate",
        # Database user.
        "USER": "weblate",
        # Name of role to alter to set parameters in PostgreSQL,
        # use in case role name is different than user used for authentication.
        # "ALTER_ROLE": "weblate",
        # Database password.
        "PASSWORD": "",
        # Set to empty string for localhost.
        "HOST": "127.0.0.1",
        # Set to empty string for default.
        "PORT": "",
        # Customizations for databases.
        "OPTIONS": {
            # In case of using an older MySQL server,
            # which has MyISAM as a default storage
            # "init_command": "SET storage_engine=INNODB",
            # Uncomment for MySQL older than 5.7:
```

(续下页)

(接上页)

```
            # "init_command": "SET sql_mode='STRICT_TRANS_TABLES'",
            # Set emoji capable charset for MySQL:
            # "charset": "utf8mb4",
            # Change connection timeout in case you get MySQL gone away error:
            # "connect_timeout": 28800,
        },
        # Persistent connections
        "CONN_MAX_AGE": 0,
        # Disable server-side cursors, might be needed with pgbouncer
        "DISABLE_SERVER_SIDE_CURSORS": False,
    }
}

# Data directory, you can use following for the development purposes:
# os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), "data")
DATA_DIR = "/home/weblate/data"
CACHE_DIR = f"{DATA_DIR}/cache"

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# In a Windows environment this must be set to your system time zone.
TIME_ZONE = "UTC"

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = "en-us"

LANGUAGES = (
    ("ar", "ةيبرعلا"),
    ("az", "Azərbaycan"),
    ("be", "Беларуская"),
    ("be@latin", "Biełaruskaja"),
    ("bg", "Български"),
    ("br", "Brezhoneg"),
    ("ca", "Català"),
    ("cs", "Čeština"),
    ("da", "Dansk"),
    ("de", "Deutsch"),
    ("en", "English"),
    ("el", "Ελληνικά"),
    ("en-gb", "English (United Kingdom)"),
    ("es", "Español"),
    ("fi", "Suomi"),
    ("fr", "Français"),
    ("gl", "Galego"),
    ("he", "עברית"),
    ("hu", "Magyar"),
    ("hr", "Hrvatski"),
    ("id", "Indonesia"),
    ("is", "Íslenska"),
    ("it", "Italiano"),
    ("ja", "日本語"),
    ("kab", "Taqbaylit"),
    ("kk", "Қазақ тілі"),
    ("ko", "🄵🄵🄵"),
    ("nb", "Norsk bokmål"),
    ("nl", "Nederlands"),
    ("pl", "Polski"),
    ("pt", "Português"),
    ("pt-br", "Português brasileiro"),
```

(续下页)

(接上页)

```
    ("ro", "Română"),
    ("ru", "Русский"),
    ("sk", "Slovenčina"),
    ("sl", "Slovenščina"),
    ("sq", "Shqip"),
    ("sr", "Српски"),
    ("sr-latn", "Srpski"),
    ("sv", "Svenska"),
    ("th", "ไทย"),
    ("tr", "Türkçe"),
    ("uk", "Українська"),
    ("zh-hans", "简体中文"),
    ("zh-hant", "正體中文"),
)

SITE_ID = 1

# If you set this to False, Django will make some optimizations so as not
# to load the internationalization machinery.
USE_I18N = True

# If you set this to False, Django will not format dates, numbers and
# calendars according to the current locale.
USE_L10N = True

# If you set this to False, Django will not use timezone-aware datetimes.
USE_TZ = True

# Type of automatic primary key, introduced in Django 3.2
DEFAULT_AUTO_FIELD = "django.db.models.AutoField"

# URL prefix to use, please see documentation for more details
URL_PREFIX = ""

# Absolute filesystem path to the directory that will hold user-uploaded files.
MEDIA_ROOT = os.path.join(DATA_DIR, "media")

# URL that handles the media served from MEDIA_ROOT. Make sure to use a
# trailing slash.
MEDIA_URL = f"{URL_PREFIX}/media/"

# Absolute path to the directory static files should be collected to.
# Don't put anything in this directory yourself; store your static files
# in apps' "static/" subdirectories and in STATICFILES_DIRS.
STATIC_ROOT = os.path.join(CACHE_DIR, "static")

# URL prefix for static files.
STATIC_URL = f"{URL_PREFIX}/static/"

# Additional locations of static files
STATICFILES_DIRS = (
    # Put strings here, like "/home/html/static" or "C:/www/django/static".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
)

# List of finder classes that know how to find static files in
# various locations.
STATICFILES_FINDERS = (
    "django.contrib.staticfiles.finders.FileSystemFinder",
    "django.contrib.staticfiles.finders.AppDirectoriesFinder",
```

(续下页)

(接上页)

```
    "compressor.finders.CompressorFinder",
)

# Make this unique, and don't share it with anybody.
# You can generate it using weblate-generate-secret-key
SECRET_KEY = ""

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "OPTIONS": {
            "context_processors": [
                "django.contrib.auth.context_processors.auth",
                "django.template.context_processors.debug",
                "django.template.context_processors.i18n",
                "django.template.context_processors.request",
                "django.template.context_processors.csrf",
                "django.contrib.messages.context_processors.messages",
                "weblate.trans.context_processors.weblate_context",
            ],
        },
        "APP_DIRS": True,
    }
]


# GitHub username and token for sending pull requests.
# Please see the documentation for more details.
GITHUB_CREDENTIALS = {}

# GitLab username and token for sending merge requests.
# Please see the documentation for more details.
GITLAB_CREDENTIALS = {}

# Bitbucket username and token for sending merge requests.
# Please see the documentation for more details.
BITBUCKETSERVER_CREDENTIALS = {}

# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.email.EmailAuth",
    # "social_core.backends.google.GoogleOAuth2",
    # "social_core.backends.github.GithubOAuth2",
    # "social_core.backends.bitbucket.BitbucketOAuth2",
    # "social_core.backends.suse.OpenSUSEOpenId",
    # "social_core.backends.ubuntu.UbuntuOpenId",
    # "social_core.backends.fedora.FedoraOpenId",
    # "social_core.backends.facebook.FacebookOAuth2",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Custom user model
AUTH_USER_MODEL = "weblate_auth.User"

# Social auth backends setup
SOCIAL_AUTH_GITHUB_KEY = ""
SOCIAL_AUTH_GITHUB_SECRET = ""
SOCIAL_AUTH_GITHUB_SCOPE = ["user:email"]

SOCIAL_AUTH_GITHUB_ORG_KEY = ""
SOCIAL_AUTH_GITHUB_ORG_SECRET = ""
```

(续下页)

(接上页)

```
SOCIAL_AUTH_GITHUB_ORG_NAME = ""

SOCIAL_AUTH_GITHUB_TEAM_KEY = ""
SOCIAL_AUTH_GITHUB_TEAM_SECRET = ""
SOCIAL_AUTH_GITHUB_TEAM_ID = ""

SOCIAL_AUTH_BITBUCKET_OAUTH2_KEY = ""
SOCIAL_AUTH_BITBUCKET_OAUTH2_SECRET = ""
SOCIAL_AUTH_BITBUCKET_OAUTH2_VERIFIED_EMAILS_ONLY = True

SOCIAL_AUTH_FACEBOOK_KEY = ""
SOCIAL_AUTH_FACEBOOK_SECRET = ""
SOCIAL_AUTH_FACEBOOK_SCOPE = ["email", "public_profile"]
SOCIAL_AUTH_FACEBOOK_PROFILE_EXTRA_PARAMS = {"fields": "id,name,email"}

SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = ""
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = ""

# Social auth settings
SOCIAL_AUTH_PIPELINE = (
    "social_core.pipeline.social_auth.social_details",
    "social_core.pipeline.social_auth.social_uid",
    "social_core.pipeline.social_auth.auth_allowed",
    "social_core.pipeline.social_auth.social_user",
    "weblate.accounts.pipeline.store_params",
    "weblate.accounts.pipeline.verify_open",
    "social_core.pipeline.user.get_username",
    "weblate.accounts.pipeline.require_email",
    "social_core.pipeline.mail.mail_validation",
    "weblate.accounts.pipeline.revoke_mail_code",
    "weblate.accounts.pipeline.ensure_valid",
    "weblate.accounts.pipeline.remove_account",
    "social_core.pipeline.social_auth.associate_by_email",
    "weblate.accounts.pipeline.reauthenticate",
    "weblate.accounts.pipeline.verify_username",
    "social_core.pipeline.user.create_user",
    "social_core.pipeline.social_auth.associate_user",
    "social_core.pipeline.social_auth.load_extra_data",
    "weblate.accounts.pipeline.cleanup_next",
    "weblate.accounts.pipeline.user_full_name",
    "weblate.accounts.pipeline.store_email",
    "weblate.accounts.pipeline.notify_connect",
    "weblate.accounts.pipeline.password_reset",
)
SOCIAL_AUTH_DISCONNECT_PIPELINE = (
    "social_core.pipeline.disconnect.allowed_to_disconnect",
    "social_core.pipeline.disconnect.get_entries",
    "social_core.pipeline.disconnect.revoke_tokens",
    "weblate.accounts.pipeline.cycle_session",
    "weblate.accounts.pipeline.adjust_primary_mail",
    "weblate.accounts.pipeline.notify_disconnect",
    "social_core.pipeline.disconnect.disconnect",
    "weblate.accounts.pipeline.cleanup_next",
)

# Custom authentication strategy
SOCIAL_AUTH_STRATEGY = "weblate.accounts.strategy.WeblateStrategy"

# Raise exceptions so that we can handle them later
SOCIAL_AUTH_RAISE_EXCEPTIONS = True
```

(续下页)

(接上页)

```
SOCIAL_AUTH_EMAIL_VALIDATION_FUNCTION = "weblate.accounts.pipeline.send_validation"
SOCIAL_AUTH_EMAIL_VALIDATION_URL = f"{URL_PREFIX}/accounts/email-sent/"
SOCIAL_AUTH_LOGIN_ERROR_URL = f"{URL_PREFIX}/accounts/login/"
SOCIAL_AUTH_EMAIL_FORM_URL = f"{URL_PREFIX}/accounts/email/"
SOCIAL_AUTH_NEW_ASSOCIATION_REDIRECT_URL = f"{URL_PREFIX}/accounts/profile/#account
↪"
SOCIAL_AUTH_PROTECTED_USER_FIELDS = ("email",)
SOCIAL_AUTH_SLUGIFY_USERNAMES = True
SOCIAL_AUTH_SLUGIFY_FUNCTION = "weblate.accounts.pipeline.slugify_username"

# Password validation configuration
AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.
↪UserAttributeSimilarityValidator"  # noqa: E501, pylint: disable=line-too-long
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
        "OPTIONS": {"min_length": 10},
    },
    {"NAME": "django.contrib.auth.password_validation.CommonPasswordValidator"},
    {"NAME": "django.contrib.auth.password_validation.NumericPasswordValidator"},
    {"NAME": "weblate.accounts.password_validation.CharsPasswordValidator"},
    {"NAME": "weblate.accounts.password_validation.PastPasswordsValidator"},
    # Optional password strength validation by django-zxcvbn-password
    # {
    #     "NAME": "zxcvbn_password.ZXCVBNValidator",
    #     "OPTIONS": {
    #         "min_score": 3,
    #         "user_attributes": ("username", "email", "full_name")
    #     }
    # },
]

# Password hashing (prefer Argon)
PASSWORD_HASHERS = [
    "django.contrib.auth.hashers.Argon2PasswordHasher",
    "django.contrib.auth.hashers.PBKDF2PasswordHasher",
    "django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher",
    "django.contrib.auth.hashers.BCryptSHA256PasswordHasher",
]

# Allow new user registrations
REGISTRATION_OPEN = True

# Shortcut for login required setting
REQUIRE_LOGIN = False

# Middleware
MIDDLEWARE = [
    "weblate.middleware.RedirectMiddleware",
    "weblate.middleware.ProxyMiddleware",
    "corsheaders.middleware.CorsMiddleware",
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "weblate.accounts.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
    "social_django.middleware.SocialAuthExceptionMiddleware",
    "weblate.accounts.middleware.RequireLoginMiddleware",
```

(续下页)

(接上页)

```
    "weblate.api.middleware.ThrottlingMiddleware",
    "weblate.middleware.SecurityMiddleware",
    "weblate.wladmin.middleware.ManageMiddleware",
]

ROOT_URLCONF = "weblate.urls"

# Django and Weblate apps
INSTALLED_APPS = [
    # Weblate apps on top to override Django locales and templates
    "weblate.addons",
    "weblate.auth",
    "weblate.checks",
    "weblate.formats",
    "weblate.glossary",
    "weblate.machinery",
    "weblate.trans",
    "weblate.lang",
    "weblate_language_data",
    "weblate.memory",
    "weblate.screenshots",
    "weblate.fonts",
    "weblate.accounts",
    "weblate.configuration",
    "weblate.utils",
    "weblate.vcs",
    "weblate.wladmin",
    "weblate.metrics",
    "weblate",
    # Optional: Git exporter
    "weblate.gitexport",
    # Standard Django modules
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "django.contrib.admin.apps.SimpleAdminConfig",
    "django.contrib.admindocs",
    "django.contrib.sitemaps",
    "django.contrib.humanize",
    # Third party Django modules
    "social_django",
    "crispy_forms",
    "crispy_bootstrap3",
    "compressor",
    "rest_framework",
    "rest_framework.authtoken",
    "django_filters",
    "django_celery_beat",
    "corsheaders",
]

# Custom exception reporter to include some details
DEFAULT_EXCEPTION_REPORTER_FILTER = "weblate.trans.debug.
↪WeblateExceptionReporterFilter"

# Default logging of Weblate messages
# - to syslog in production (if available)
# - otherwise to console
# - you can also choose "logfile" to log into separate file
```

(续下页)

(接上页)

```
#   after configuring it below

# Detect if we can connect to syslog
HAVE_SYSLOG = False
if platform.system() != "Windows":
    try:
        handler = SysLogHandler(address="/dev/log", facility=SysLogHandler.LOG_
↪LOCAL2)
        handler.close()
        HAVE_SYSLOG = True
    except OSError:
        HAVE_SYSLOG = False

DEFAULT_LOG = "console" if DEBUG or not HAVE_SYSLOG else "syslog"
DEFAULT_LOGLEVEL = "DEBUG" if DEBUG else "INFO"

# A sample logging configuration. The only tangible logging
# performed by this configuration is to send an email to
# the site admins on every HTTP 500 error when DEBUG=False.
# See http://docs.djangoproject.com/en/stable/topics/logging for
# more details on how to customize your logging configuration.
LOGGING = {
    "version": 1,
    "disable_existing_loggers": True,
    "filters": {"require_debug_false": {"()": "django.utils.log.RequireDebugFalse"}
↪},
    "formatters": {
        "syslog": {"format": "weblate[%(process)d]: %(levelname)s %(message)s"},
        "simple": {"format": "[%(asctime)s: %(levelname)s/%(process)s] %(message)s
↪"},
        "logfile": {"format": "%(asctime)s %(levelname)s %(message)s"},
        "django.server": {
            "()": "django.utils.log.ServerFormatter",
            "format": "[%(server_time)s] %(message)s",
        },
    },
    "handlers": {
        "mail_admins": {
            "level": "ERROR",
            "filters": ["require_debug_false"],
            "class": "django.utils.log.AdminEmailHandler",
            "include_html": True,
        },
        "console": {
            "level": "DEBUG",
            "class": "logging.StreamHandler",
            "formatter": "simple",
        },
        "django.server": {
            "level": "INFO",
            "class": "logging.StreamHandler",
            "formatter": "django.server",
        },
        "syslog": {
            "level": "DEBUG",
            "class": "logging.handlers.SysLogHandler",
            "formatter": "syslog",
            "address": "/dev/log",
            "facility": SysLogHandler.LOG_LOCAL2,
        },
        # Logging to a file
```

(续下页)

(接上页)

```
        # "logfile": {
        #     "level":"DEBUG",
        #     "class":"logging.handlers.RotatingFileHandler",
        #     "filename": "/var/log/weblate/weblate.log",
        #     "maxBytes": 100000,
        #     "backupCount": 3,
        #     "formatter": "logfile",
        # },
    },
    "loggers": {
        "django.request": {
            "handlers": ["mail_admins", DEFAULT_LOG],
            "level": "ERROR",
            "propagate": True,
        },
        "django.server": {
            "handlers": ["django.server"],
            "level": "INFO",
            "propagate": False,
        },
        # Logging database queries
        # "django.db.backends": {
        #     "handlers": [DEFAULT_LOG],
        #     "level": "DEBUG",
        # },
        "weblate": {"handlers": [DEFAULT_LOG], "level": DEFAULT_LOGLEVEL},
        # Logging VCS operations
        "weblate.vcs": {"handlers": [DEFAULT_LOG], "level": DEFAULT_LOGLEVEL},
        # Python Social Auth
        "social": {"handlers": [DEFAULT_LOG], "level": DEFAULT_LOGLEVEL},
        # Django Authentication Using LDAP
        "django_auth_ldap": {"handlers": [DEFAULT_LOG], "level": DEFAULT_LOGLEVEL},
        # SAML IdP
        "djangosaml2idp": {"handlers": [DEFAULT_LOG], "level": DEFAULT_LOGLEVEL},
    },
}

# Remove syslog setup if it's not present
if not HAVE_SYSLOG:
    del LOGGING["handlers"]["syslog"]

# List of machine translations
MT_SERVICES = (
    #     "weblate.machinery.apertium.ApertiumAPYTranslation",
    #     "weblate.machinery.baidu.BaiduTranslation",
    #     "weblate.machinery.deepl.DeepLTranslation",
    #     "weblate.machinery.glosbe.GlosbeTranslation",
    #     "weblate.machinery.google.GoogleTranslation",
    #     "weblate.machinery.googlev3.GoogleV3Translation",
    #     "weblate.machinery.libretranslate.LibreTranslateTranslation",
    #     "weblate.machinery.microsoft.MicrosoftCognitiveTranslation",
    #     "weblate.machinery.microsoftterminology.MicrosoftTerminologyService",
    #     "weblate.machinery.modernmt.ModernMTTranslation",
    #     "weblate.machinery.mymemory.MyMemoryTranslation",
    #     "weblate.machinery.netease.NeteaseSightTranslation",
    #     "weblate.machinery.tmserver.AmagamaTranslation",
    #     "weblate.machinery.tmserver.TMServerTranslation",
    #     "weblate.machinery.yandex.YandexTranslation",
    #     "weblate.machinery.saptranslationhub.SAPTranslationHub",
    #     "weblate.machinery.youdao.YoudaoTranslation",
    "weblate.machinery.weblatetm.WeblateTranslation",
```

(续下页)

(接上页)

```
    "weblate.memory.machine.WeblateMemory",
)

# Machine translation API keys

# URL of the Apertium APy server
MT_APERTIUM_APY = None

# DeepL API key
MT_DEEPL_KEY = None

# LibreTranslate
MT_LIBRETRANSLATE_API_URL = None
MT_LIBRETRANSLATE_KEY = None

# Microsoft Cognitive Services Translator API, register at
# https://portal.azure.com/
MT_MICROSOFT_COGNITIVE_KEY = None
MT_MICROSOFT_REGION = None

# ModernMT
MT_MODERNMT_KEY = None

# MyMemory identification email, see
# https://mymemory.translated.net/doc/spec.php
MT_MYMEMORY_EMAIL = None

# Optional MyMemory credentials to access private translation memory
MT_MYMEMORY_USER = None
MT_MYMEMORY_KEY = None

# Google API key for Google Translate API v2
MT_GOOGLE_KEY = None

# Google Translate API3 credentials and project id
MT_GOOGLE_CREDENTIALS = None
MT_GOOGLE_PROJECT = None

# Baidu app key and secret
MT_BAIDU_ID = None
MT_BAIDU_SECRET = None

# Youdao Zhiyun app key and secret
MT_YOUDAO_ID = None
MT_YOUDAO_SECRET = None

# Netease Sight (Jianwai) app key and secret
MT_NETEASE_KEY = None
MT_NETEASE_SECRET = None

# API key for Yandex Translate API
MT_YANDEX_KEY = None

# tmserver URL
MT_TMSERVER = None

# SAP Translation Hub
MT_SAP_BASE_URL = None
MT_SAP_SANDBOX_APIKEY = None
MT_SAP_USERNAME = None
MT_SAP_PASSWORD = None
```

(续下页)

(接上页)

```
MT_SAP_USE_MT = True

# Use HTTPS when creating redirect URLs for social authentication, see
# documentation for more details:
# https://python-social-auth-docs.readthedocs.io/en/latest/configuration/settings.
↪html#processing-redirects-and-urlopen
SOCIAL_AUTH_REDIRECT_IS_HTTPS = ENABLE_HTTPS

# Make CSRF cookie HttpOnly, see documentation for more details:
# https://docs.djangoproject.com/en/1.11/ref/settings/#csrf-cookie-httponly
CSRF_COOKIE_HTTPONLY = True
CSRF_COOKIE_SECURE = ENABLE_HTTPS
# Store CSRF token in session
CSRF_USE_SESSIONS = True
# Customize CSRF failure view
CSRF_FAILURE_VIEW = "weblate.trans.views.error.csrf_failure"
SESSION_COOKIE_SECURE = ENABLE_HTTPS
SESSION_COOKIE_HTTPONLY = True
# SSL redirect
SECURE_SSL_REDIRECT = ENABLE_HTTPS
SECURE_SSL_HOST = SITE_DOMAIN
# Sent referrrer only for same origin links
SECURE_REFERRER_POLICY = "same-origin"
# SSL redirect URL exemption list
SECURE_REDIRECT_EXEMPT = (r"healthz/$",)  # Allowing HTTP access to health check
# Session cookie age (in seconds)
SESSION_COOKIE_AGE = 1000
SESSION_COOKIE_AGE_AUTHENTICATED = 1209600
SESSION_COOKIE_SAMESITE = "Lax"
# Increase allowed upload size
DATA_UPLOAD_MAX_MEMORY_SIZE = 50000000
# Allow more fields for case with a lot of subscriptions in profile
DATA_UPLOAD_MAX_NUMBER_FIELDS = 2000

# Apply session coookie settings to language cookie as ewll
LANGUAGE_COOKIE_SECURE = SESSION_COOKIE_SECURE
LANGUAGE_COOKIE_HTTPONLY = SESSION_COOKIE_HTTPONLY
LANGUAGE_COOKIE_AGE = SESSION_COOKIE_AGE_AUTHENTICATED * 10
LANGUAGE_COOKIE_SAMESITE = SESSION_COOKIE_SAMESITE

# Some security headers
SECURE_BROWSER_XSS_FILTER = True
X_FRAME_OPTIONS = "DENY"
SECURE_CONTENT_TYPE_NOSNIFF = True

# Optionally enable HSTS
SECURE_HSTS_SECONDS = 31536000 if ENABLE_HTTPS else 0
SECURE_HSTS_PRELOAD = ENABLE_HTTPS
SECURE_HSTS_INCLUDE_SUBDOMAINS = ENABLE_HTTPS

# HTTPS detection behind reverse proxy
SECURE_PROXY_SSL_HEADER = None

# URL of login
LOGIN_URL = f"{URL_PREFIX}/accounts/login/"

# URL of logout
LOGOUT_URL = f"{URL_PREFIX}/accounts/logout/"

# Default location for login
LOGIN_REDIRECT_URL = f"{URL_PREFIX}/"
```

(续下页)

(接上页)

```
# Anonymous user name
ANONYMOUS_USER_NAME = "anonymous"

# Reverse proxy settings
IP_PROXY_HEADER = "HTTP_X_FORWARDED_FOR"
IP_BEHIND_REVERSE_PROXY = False
IP_PROXY_OFFSET = 0

# Sending HTML in mails
EMAIL_SEND_HTML = True

# Subject of emails includes site title
EMAIL_SUBJECT_PREFIX = f"[{SITE_TITLE}] "

# Enable remote hooks
ENABLE_HOOKS = True

# By default the length of a given translation is limited to the length of
# the source string * 10 characters. Set this option to False to allow longer
# translations (up to 10.000 characters)
LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH = True

# Use simple language codes for default language/country combinations
SIMPLIFY_LANGUAGES = True

# Render forms using bootstrap
CRISPY_ALLOWED_TEMPLATE_PACKS = "bootstrap3"
CRISPY_TEMPLATE_PACK = "bootstrap3"

# List of quality checks
# CHECK_LIST = (
#     "weblate.checks.same.SameCheck",
#     "weblate.checks.chars.BeginNewlineCheck",
#     "weblate.checks.chars.EndNewlineCheck",
#     "weblate.checks.chars.BeginSpaceCheck",
#     "weblate.checks.chars.EndSpaceCheck",
#     "weblate.checks.chars.DoubleSpaceCheck",
#     "weblate.checks.chars.EndStopCheck",
#     "weblate.checks.chars.EndColonCheck",
#     "weblate.checks.chars.EndQuestionCheck",
#     "weblate.checks.chars.EndExclamationCheck",
#     "weblate.checks.chars.EndEllipsisCheck",
#     "weblate.checks.chars.EndSemicolonCheck",
#     "weblate.checks.chars.MaxLengthCheck",
#     "weblate.checks.chars.KashidaCheck",
#     "weblate.checks.chars.PunctuationSpacingCheck",
#     "weblate.checks.format.PythonFormatCheck",
#     "weblate.checks.format.PythonBraceFormatCheck",
#     "weblate.checks.format.PHPFormatCheck",
#     "weblate.checks.format.CFormatCheck",
#     "weblate.checks.format.PerlFormatCheck",
#     "weblate.checks.format.JavaScriptFormatCheck",
#     "weblate.checks.format.LuaFormatCheck",
#     "weblate.checks.format.ObjectPascalFormatCheck",
#     "weblate.checks.format.SchemeFormatCheck",
#     "weblate.checks.format.CSharpFormatCheck",
#     "weblate.checks.format.JavaFormatCheck",
#     "weblate.checks.format.JavaMessageFormatCheck",
#     "weblate.checks.format.PercentPlaceholdersCheck",
#     "weblate.checks.format.VueFormattingCheck",
```

(续下页)

(接上页)

```
#     "weblate.checks.format.I18NextInterpolationCheck",
#     "weblate.checks.format.ESTemplateLiteralsCheck",
#     "weblate.checks.angularjs.AngularJSInterpolationCheck",
#     "weblate.checks.icu.ICUMessageFormatCheck",
#     "weblate.checks.icu.ICUSourceCheck",
#     "weblate.checks.qt.QtFormatCheck",
#     "weblate.checks.qt.QtPluralCheck",
#     "weblate.checks.ruby.RubyFormatCheck",
#     "weblate.checks.consistency.PluralsCheck",
#     "weblate.checks.consistency.SamePluralsCheck",
#     "weblate.checks.consistency.ConsistencyCheck",
#     "weblate.checks.consistency.TranslatedCheck",
#     "weblate.checks.chars.EscapedNewlineCountingCheck",
#     "weblate.checks.chars.NewLineCountCheck",
#     "weblate.checks.markup.BBCodeCheck",
#     "weblate.checks.chars.ZeroWidthSpaceCheck",
#     "weblate.checks.render.MaxSizeCheck",
#     "weblate.checks.markup.XMLValidityCheck",
#     "weblate.checks.markup.XMLTagsCheck",
#     "weblate.checks.markup.MarkdownRefLinkCheck",
#     "weblate.checks.markup.MarkdownLinkCheck",
#     "weblate.checks.markup.MarkdownSyntaxCheck",
#     "weblate.checks.markup.URLCheck",
#     "weblate.checks.markup.SafeHTMLCheck",
#     "weblate.checks.placeholders.PlaceholderCheck",
#     "weblate.checks.placeholders.RegexCheck",
#     "weblate.checks.duplicate.DuplicateCheck",
#     "weblate.checks.source.OptionalPluralCheck",
#     "weblate.checks.source.EllipsisCheck",
#     "weblate.checks.source.MultipleFailingCheck",
#     "weblate.checks.source.LongUntranslatedCheck",
#     "weblate.checks.format.MultipleUnnamedFormatsCheck",
#     "weblate.checks.glossary.GlossaryCheck",
# )

# List of automatic fixups
# AUTOFIX_LIST = (
#     "weblate.trans.autofixes.whitespace.SameBookendingWhitespace",
#     "weblate.trans.autofixes.chars.ReplaceTrailingDotsWithEllipsis",
#     "weblate.trans.autofixes.chars.RemoveZeroSpace",
#     "weblate.trans.autofixes.chars.RemoveControlChars",
# )

# List of enabled addons
# WEBLATE_ADDONS = (
#     "weblate.addons.gettext.GenerateMoAddon",
#     "weblate.addons.gettext.UpdateLinguasAddon",
#     "weblate.addons.gettext.UpdateConfigureAddon",
#     "weblate.addons.gettext.MsgmergeAddon",
#     "weblate.addons.gettext.GettextCustomizeAddon",
#     "weblate.addons.gettext.GettextAuthorComments",
#     "weblate.addons.cleanup.CleanupAddon",
#     "weblate.addons.cleanup.RemoveBlankAddon",
#     "weblate.addons.consistency.LangaugeConsistencyAddon",
#     "weblate.addons.discovery.DiscoveryAddon",
#     "weblate.addons.autotranslate.AutoTranslateAddon",
#     "weblate.addons.flags.SourceEditAddon",
#     "weblate.addons.flags.TargetEditAddon",
#     "weblate.addons.flags.SameEditAddon",
#     "weblate.addons.flags.BulkEditAddon",
#     "weblate.addons.generate.GenerateFileAddon",
```

(续下页)

(接上页)

```
#     "weblate.addons.generate.PseudolocaleAddon",
#     "weblate.addons.generate.PrefillAddon",
#     "weblate.addons.json.JSONCustomizeAddon",
#     "weblate.addons.xml.XMLCustomizeAddon",
#     "weblate.addons.properties.PropertiesSortAddon",
#     "weblate.addons.git.GitSquashAddon",
#     "weblate.addons.removal.RemoveComments",
#     "weblate.addons.removal.RemoveSuggestions",
#     "weblate.addons.resx.ResxUpdateAddon",
#     "weblate.addons.yaml.YAMLCustomizeAddon",
#     "weblate.addons.cdn.CDNJSAddon",
# )

# E-mail address that error messages come from.
SERVER_EMAIL = "noreply@example.com"

# Default email address to use for various automated correspondence from
# the site managers. Used for registration emails.
DEFAULT_FROM_EMAIL = "noreply@example.com"

# List of URLs your site is supposed to serve
ALLOWED_HOSTS = ["*"]

# Configuration for caching
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        # If redis is running on same host as Weblate, you might
        # want to use unix sockets instead:
        # "LOCATION": "unix:///var/run/redis/redis.sock?db=1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "PARSER_CLASS": "redis.connection.HiredisParser",
            # If you set password here, adjust CELERY_BROKER_URL as well
            "PASSWORD": None,
            "CONNECTION_POOL_KWARGS": {},
        },
        "KEY_PREFIX": "weblate",
        "TIMEOUT": 3600,
    },
    "avatar": {
        "BACKEND": "django.core.cache.backends.filebased.FileBasedCache",
        "LOCATION": os.path.join(CACHE_DIR, "avatar"),
        "TIMEOUT": 86400,
        "OPTIONS": {"MAX_ENTRIES": 1000},
    },
}

# Store sessions in cache
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
# Store messages in session
MESSAGE_STORAGE = "django.contrib.messages.storage.session.SessionStorage"

# REST framework settings for API
REST_FRAMEWORK = {
    # Use Django's standard `django.contrib.auth` permissions,
    # or allow read-only access for unauthenticated users.
    "DEFAULT_PERMISSION_CLASSES": [
        # Require authentication for login required sites
        "rest_framework.permissions.IsAuthenticated"
```

(续下页)

(接上页)

```
        if REQUIRE_LOGIN
        else "rest_framework.permissions.IsAuthenticatedOrReadOnly"
    ],
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.TokenAuthentication",
        "weblate.api.authentication.BearerAuthentication",
        "rest_framework.authentication.SessionAuthentication",
    ),
    "DEFAULT_THROTTLE_CLASSES": (
        "weblate.api.throttling.UserRateThrottle",
        "weblate.api.throttling.AnonRateThrottle",
    ),
    "DEFAULT_THROTTLE_RATES": {"anon": "100/day", "user": "5000/hour"},
    "DEFAULT_PAGINATION_CLASS": "weblate.api.pagination.StandardPagination",
    "PAGE_SIZE": 50,
    "VIEW_DESCRIPTION_FUNCTION": "weblate.api.views.get_view_description",
    "UNAUTHENTICATED_USER": "weblate.auth.models.get_anonymous",
}

# Fonts CDN URL
FONTS_CDN_URL = None

# Django compressor offline mode
COMPRESS_OFFLINE = False
COMPRESS_OFFLINE_CONTEXT = [
    {"fonts_cdn_url": FONTS_CDN_URL, "STATIC_URL": STATIC_URL, "LANGUAGE_BIDI":␣
↪True},
    {"fonts_cdn_url": FONTS_CDN_URL, "STATIC_URL": STATIC_URL, "LANGUAGE_BIDI":␣
↪False},
]

# Require login for all URLs
if REQUIRE_LOGIN:
    LOGIN_REQUIRED_URLS = (r"/(.*)$",)

# In such case you will want to include some of the exceptions
# LOGIN_REQUIRED_URLS_EXCEPTIONS = (
#     rf"{URL_PREFIX}/accounts/(.*)$",  # Required for login
#     rf"{URL_PREFIX}/admin/login/(.*)$",  # Required for admin login
#     rf"{URL_PREFIX}/static/(.*)$",  # Required for development mode
#     rf"{URL_PREFIX}/widgets/(.*)$",  # Allowing public access to widgets
#     rf"{URL_PREFIX}/data/(.*)$",  # Allowing public access to data exports
#     rf"{URL_PREFIX}/hooks/(.*)$",  # Allowing public access to notification hooks
#     rf"{URL_PREFIX}/healthz/$",  # Allowing public access to health check
#     rf"{URL_PREFIX}/api/(.*)$",  # Allowing access to API
#     rf"{URL_PREFIX}/js/i18n/$",  # JavaScript localization
#     rf"{URL_PREFIX}/contact/$",  # Optional for contact form
#     rf"{URL_PREFIX}/legal/(.*)$",  # Optional for legal app
#     rf"{URL_PREFIX}/avatar/(.*)$",  # Optional for avatars
# )

# Silence some of the Django system checks
SILENCED_SYSTEM_CHECKS = [
    # We have modified django.contrib.auth.middleware.AuthenticationMiddleware
    # as weblate.accounts.middleware.AuthenticationMiddleware
    "admin.E408"
]

# Celery worker configuration for testing
# CELERY_TASK_ALWAYS_EAGER = True
# CELERY_BROKER_URL = "memory://"
```

(续下页)

(接上页)

```
# CELERY_TASK_EAGER_PROPAGATES = True
# Celery worker configuration for production
CELERY_TASK_ALWAYS_EAGER = False
CELERY_BROKER_URL = "redis://localhost:6379"
CELERY_RESULT_BACKEND = CELERY_BROKER_URL

# Celery settings, it is not recommended to change these
CELERY_WORKER_MAX_MEMORY_PER_CHILD = 200000
CELERY_BEAT_SCHEDULER = "django_celery_beat.schedulers:DatabaseScheduler"
CELERY_TASK_ROUTES = {
    "weblate.trans.tasks.auto_translate*": {"queue": "translate"},
    "weblate.accounts.tasks.notify_*": {"queue": "notify"},
    "weblate.accounts.tasks.send_mails": {"queue": "notify"},
    "weblate.utils.tasks.settings_backup": {"queue": "backup"},
    "weblate.utils.tasks.database_backup": {"queue": "backup"},
    "weblate.wladmin.tasks.backup": {"queue": "backup"},
    "weblate.wladmin.tasks.backup_service": {"queue": "backup"},
    "weblate.memory.tasks.*": {"queue": "memory"},
}

# CORS allowed origins
CORS_ALLOWED_ORIGINS = []
CORS_URLS_REGEX = r"^/api/.*$"

# Enable plain database backups
DATABASE_BACKUP = "plain"

# Enable auto updating
AUTO_UPDATE = False

# PGP commits signing
WEBLATE_GPG_IDENTITY = None

# Third party services integration
MATOMO_SITE_ID = None
MATOMO_URL = None
GOOGLE_ANALYTICS_ID = None
SENTRY_DSN = None
SENTRY_ENVIRONMENT = SITE_DOMAIN
AKISMET_API_KEY = None
```

2.18 管理命令

备注： 在与运行 web 服务器的用户不同的用户下运行管理命令，可能会导致文件获得错误的权限，请查阅文件系统权限 了解详情。

您会找到基本的管理命令（作为 Django 源中的 `./manage.py` 来获得它，或者作为可安装在 Weblate 顶层的脚本调用 **weblate** 中的扩展组来获得它）。

2.18.1 调用管理命令

如上面所提到的，如何调用取决于您如何安装 Weblate。

如果使用 Virtualenv 来运行 Weblate，那么您可以指定 **weblate** 的完整路径，或者在调用前激活 virtualenv：

```
# Direct invocation
~/weblate-env/bin/weblate

# Activating virtualenv adds it to search path
. ~/weblate-env/bin/activate
weblate
```

如果您直接使用源代码（来源于 tarball 或 Git checkout），管理脚本可以在 Weblate 源文件的 `./manage.py` 中获得。要运行它：

```
python ./manage.py list_versions
```

如果您使用 pip 安装程序，或使用 `./setup.py` 脚本来安装 Weblate，那么 **weblate** 安装到您的路径下（或者 virtualenv 路径下），您可以从那里用它控制 Weblate：

```
weblate list_versions
```

对于 Docker 镜像，脚本向上面一样安装，您可以使用 **docker exec** 来运行：

```
docker exec --user weblate <container> weblate list_versions
```

对于 **docker-compose**，过程是相似的，您只是必须使用 **docker-compose exec**：

```
docker-compose exec --user weblate weblate weblate list_versions
```

在您需要向它传递文件的情况下，您可以临时添加卷：

```
docker-compose exec --user weblate /tmp:/tmp weblate weblate importusers /tmp/
↪users.json
```

参见：

使用 Docker 安装, *在 Debian 和 Ubantu 上安装*, *在 SUSE 和 openSUSE 上安装*, *在 Redhat、Fedora 和 CentOS 上安装*, *从源码安装*

2.18.2 add_suggestions

weblate add_suggestions <project> <component> <language> <file>

在 2.5 版本加入.

从文件导入翻译，用作给定翻译的建议。它会跳过重复的翻译；只会添加不同的内容。

--author USER@EXAMPLE.COM

建议的作者电子邮箱地址。这个用户必须在导入前存在（您可以根据需要在管理界面建立一个）。

示例：

```
weblate --author michal@cihar.com add_suggestions weblate application cs /tmp/
↪suggestions-cs.po
```

2.18.3 auto_translate

weblate auto_translate <project> <component> <language>

在 2.5 版本加入.

在 4.6 版本发生变更: 添加了翻译模式参数。

根据其他部件翻译进行自动翻译。

--source PROJECT/COMPONENT
: 指定可用作翻译的来源的部件。如果不指定，将使用项目中的所有部件。

--user USERNAME
: 指定列出的用户名，作为翻译的作者。如果不指定那么使用“匿名用户”。

--overwrite
: 是否去覆盖现有的翻译。

--inconsistent
: 是否去覆盖现有的不一致的翻译（请参见不一致的）。

--add
: 如果给定的翻译不存在，自动添加语言。

--mt MT
: 实用机器翻译而不是其他部件作为机器翻译。

--threshold THRESHOLD
: 机器翻译的相似度阈值，默认为 80。

--mode MODE
: 指定翻译模式，默认为 `translate` 也可以使用 `fuzzy` 或 `suggest`。

示例:

```
weblate auto_translate --user nijel --inconsistent --source weblate/application␣
↪weblate website cs
```

参见:

自动翻译

2.18.4 celery_queues

weblate celery_queues

在 3.7 版本加入.

显示 Celery 任务队列的长度。

2.18.5 checkgit

weblate checkgit <project|project/component>

打印后端 Git 仓库的当前状态。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有部件。

2.18.6 commitgit

weblate commitgit <project|project/component>

将任何可能的待处理更改提交到后端 Git 仓库。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有部件，或使用 `--file-format` 基于文件格式进行过滤。

2.18.7 commit_pending

weblate commit_pending <project|project/component>

提交超过给定期限的待处理更改。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有部件。

--age HOURS
: 时间段以小时为单位。如果不指定，则使用在部件配置 中配置的值。

备注： 这由 Weblate 在后台自动执行，所以实际不需要手动调用，除了要强制早于部件配置 指定的执行。

参见：

运行维护任务, *COMMIT_PENDING_HOURS*

2.18.8 cleanuptrans

weblate cleanuptrans

清理无主的检查和翻译建议。这通常不需要手动运行，因为清理在后台自动启动。

参见：

运行维护任务

2.18.9 cleanup_ssh_keys

weblate cleanup_ssh_keys

在 4.9.1 版本加入.

对存储的 SSH 主机密钥进行清理：

- 移除已废弃的 GitHub 的 RSA 密钥，这可能会导致连接到 GitHub 的问题。
- 删除主机钥匙中的重复条目。

参见：

SSH 仓库

2.18.10 createadmin

weblate createadmin

除非指定，否则用随机密码建立 admin 账户。

--password PASSWORD
在命令行提供密码，而不要生成随机的。

--no-password
不要设置密码，这对 '–update'可能有用。

--username USERNAME
使用给定的姓名而不是 admin。

--email USER@EXAMPLE.COM
指定 admin 的电子邮箱地址。

--name
指定 admin 的姓名（可见的）。

--update
更新现有的用户（您可以用这个来更改密码）。

在 2.9 版本发生变更: 添加参数 --username 、--email 、--name 和 --update。

2.18.11 dump_memory

weblate dump_memory

在 2.20 版本加入.

将包含 Weblate 翻译记忆库内容的 JSON 文件导出。

参见:

翻译记忆库, *Weblate* 翻译记忆库概要

2.18.12 dumpuserdata

weblate dumpuserdata <file.json>

通过*importuserdata* 将 userdata 转储到文件供以后使用。

提示: 这在迁移或合并 Weblate 实例时会很方便。

2.18.13 import_demo

weblate import_demo

在 4.1 版本加入.

基于 <https://github.com/WeblateOrg/demo> 创建包含部件的演示项目。运行此命令之前，请确保 celery 任务正在运行。

这在开发 Weblate 时会有用。

2.18.14 import_json

weblate import_json <json-file>

在 2.7 版本加入.

根据 JSON 数据批量导入部件。

导入的 JSON 文件结构非常符合部件对象（请参见*GET /api/components/(string:project)/(string:component)/*）。您必须包括 name 和 filemask 字段。

--project PROJECT
: 指定从哪里导入部件。

--main-component COMPONENT
: 对所有的使用来自这个部件的给定版本控制系统（VCS）仓库。

--ignore
: 跳过（已经）导入的部件。

--update
: 更新（已经）导入的部件。

在 2.9 版本发生变更: 那里的参数 --ignore 和 --update 用于处理已经导入的部件。

JSON 文件的示例：

```
[
  {
    "slug": "po",
    "name": "Gettext PO",
    "file_format": "po",
    "filemask": "po/*.po",
    "new_lang": "none"
  },
  {
    "name": "Android",
    "filemask": "android/values-*/strings.xml",
    "template": "android/values/strings.xml",
    "repo": "weblate://test/test",
    "file_format": "aresource"
  }
]
```

参见:

import_memory

2.18.15 import_memory

weblate import_memory <file>

在 2.20 版本加入.

将 TMX 或 JSON 文件导入 Weblate 翻译记忆库。

--language-map LANGMAP
: 允许将 TMX 的语言映射到 Weblate 翻译记忆库。语言代码通常在 Weblate 进行规范化之后映射。

  例如 --language-map en_US:en 将所有 en_US 字符串作为 en 字符串来导入。

  如果您的 TMX 文件的地区恰好与您在 Weblate 中使用的地区不匹配，这会很有用。

参见:

翻译记忆库, *Weblate 翻译记忆库概要*

2.18.16 import_project

weblate import_project <project> <gitrepo> <branch> <filemask>

在 3.0 版本发生变更: import_project 命令现在基于部件发现 附加组件，导致一些行为的更改，并接受一些参数。

根据文件掩码，将部件批量导入项目。

<project> 将已存在的项目命名，部件将导入其中。

<gitrepo> 确定了要使用的 Git 仓库的 URL，而 *<branch>* 说明了 Git 分支。为了从现有的 Weblate 部件导入另外的翻译部件，使用 *<gitrepo>* 的 *weblate://<project>/<component>* URL。

<filemask> 为仓库定义了文件发现。或者可以使用通配符来使它简单，或者可以使用正则表达式的全部功能。

简单的匹配对部件名称使用 `**`，对语言使用 `*`，例如：`**/*.po`

正则表达式必须包含组命名的 *component* 和 *language*。例如：`(?P<language>[^/]*)/(?P<component>[^-/]*)\.po`

根据文件，导入与现有的部件匹配，并且添加不存在的那些。它不更改已经存在的那些。

--name-template TEMPLATE
: 使用 Django 模板语法来定制部件的名称。

  例如：`Documentation: {{ component }}`

--base-file-template TEMPLATE
: 为单语言翻译定制翻译模板文件。

  例如：`{{ component }}/res/values/string.xml`

--new-base-template TEMPLATE
: 为另外新的翻译定制翻译模板文件。

  例如：`{{ component }}/ts/en.ts`

--file-format FORMAT
: 您还可以使用的文件格式（请参见：支持的文件格式），默认为自动检测。

--language-regex REGEX
: 您可以使用这个参数指定语言过滤器（请参见：部件配置）。它必须是合法的正则表达式。

--main-component
: 你可以指定哪个部件作为主部件——即真正包含版本控制系统（VCS）仓库的部件。

--license NAME
: 指定整体、项目或部件翻译的许可。

--license-url URL
: 指定翻译许可所在的 URL。

--vcs NAME
: 在需要指定使用哪个版本的轻质系统的情况下，您可以在这里进行。默认版本控制是 Git。

为了给出一些示例，让我们导入两个项目。

第一个是 Debian 手册翻译，那里的每种语言都有各自的文件夹，里面有每个章节的翻译：

```
weblate import_project \
    debian-handbook \
    git://anonscm.debian.org/debian-handbook/debian-handbook.git \
    squeeze/master \
    '*/**.po'
```

然后 Tangaguru 工具，那里需要指定文件格式和翻译模板文件，并且指定所有部件和翻译如何位于单一一个文件夹中：

```
weblate import_project \
    --file-format=properties \
    --base-file-template=web-app/tgol-web-app/src/main/resources/i18n/%s-I18N.
↪properties \
    tanaguru \
    https://github.com/Tanaguru/Tanaguru \
    master \
    web-app/tgol-web-app/src/main/resources/i18n/**-I18N_*.properties
```

更复杂的示例是关于解析文件名而从文件名中得到正确的部件和语言，像 `src/security/Numerous_security_holes_in_0.10.1.de.po`：

```
weblate import_project \
    tails \
    git://git.tails.boum.org/tails master \
    'wiki/src/security/(?P<component>.*)\.(?P<language>[^.]*)\.po$'
```

筛选出指定的语言的翻译：

```
./manage import_project \
    --language-regex '^(cs|sk)$' \
    weblate \
    https://github.com/WeblateOrg/weblate.git \
    'weblate/locale/*/LC_MESSAGES/**.po'
```

导入 Sphinx 文档，分成多个文件：

```
$ weblate import_project --name-template 'Documentation: %s' \
    --file-format po \
    project https://github.com/project/docs.git master \
    'docs/locale/*/LC_MESSAGES/**.po'
```

导入 Sphinx 文档，分成多个文件和文件夹：

```
$ weblate import_project --name-template 'Directory 1: %s' \
    --file-format po \
    project https://github.com/project/docs.git master \
    'docs/locale/*/LC_MESSAGES/dir1/**.po'
$ weblate import_project --name-template 'Directory 2: %s' \
    --file-format po \
    project https://github.com/project/docs.git master \
    'docs/locale/*/LC_MESSAGES/dir2/**.po'
```

参见：

更多具体的示例可以在 starting 章节找到，另外您会想要使用 *import_json*。

2.18.17 importuserdata

weblate importuserdata <file.json>

从 *dumpuserdata* 建立的文件中导入用户数据

2.18.18 importusers

weblate importusers --check <file.json>

从 Django auth_users 数据库的 JSON 转储中导入用户。

--check

使用这个选项可以检查给定文件是否可以被导入，并且报告用户名或电子地址可能导致的冲突。

可以从现有的 Django 安装中导出用户，这需要使用：

```
weblate dumpdata auth.User > users.json
```

2.18.19 install_addon

在 3.2 版本加入.

weblate install_addon --addon ADDON <project|project/component>

将一个附加组件安装到一组部件中。

--addon ADDON

要安装的附加组件名称。例如 `weblate.gettext.customize`。

--configuration CONFIG

以 JSON 编码的附加组件配置。

--update

更新现有的附加组件配置。

可以定义将附加组件安装到哪个项目或部件中（例如 `weblate/application`），或者使用 `--all` 来包含所有现有的部件。

为所有部件安装*自定义 gettext 输出*：

```
weblate install_addon --addon weblate.gettext.customize --config '{"width": -1}' --
↪update --all
```

参见:

附加组件

2.18.20 list_languages

weblate list_languages <locale>

列出 MediaWiki 标记中支持的语言——语言代码、英语名称和本地化名称。

这用来生成 <https://wiki.l10n.cz/Slovn%C3%ADk_s_n%C3%A1zvy_jazyk%C5%AF>.

2.18.21 list_translators

weblate list_translators <project|project/component>

按给定项目的语言列出为对应语言作贡献的译者:

```
[French]
Jean Dupont <jean.dupont@example.com>
[English]
John Doe <jd@example.com>
```

--language-code

用语言代码而不是语言来列出名称。

可以或者定义使用哪个项目或部件（例如 `weblate/application`），或者使用 `--all` 从所有现存的部件中列出译者。

2.18.22 list_versions

weblate list_versions

列出所有 Weblate 依赖项及其版本。

2.18.23 loadpo

weblate loadpo <project|project/component>

从磁盘重新加载翻译（例如，您在版本控制系统（VCS）仓库中进行了一些更新）。

--force

强制更新，即使文件应该是更新的。

--lang LANGUAGE

将处理限制为单一语言。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application` ），或者使用 `--all` 来更新所有现有部件。

备注： 你很少需要调用这个命令，Weblate 会在每次版本控制系统（VCS）更新时自动加载更改的文件。如果您手动更改了 Weblate 的底层版本控制系统（VCS）仓库，或者在升级后的某些特殊情况下，则需要调用这个命令。

2.18.24 lock_translation

weblate lock_translation <project|project/component>

防止部件的进一步翻译。

提示： 在您想要对下层仓库进行一些维护时有用。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application` ），或者使用 `--all` 来更新所有现有部件。

参见：

unlock_translation

2.18.25 move_language

weblate move_language source target

在 3.0 版本加入.

允许您合并语言内容。当更新到新版本，这个新版本对使用 *(generated)* 前缀建立的之前未知的语言包含别名时，这会有用。它将所有内容从 *source* 语言移动到 *target* 语言。

示例：

```
weblate move_language cze cs
```

移动内容后，您应该检查是否有什么落下了（这是因为有人在同时更新仓库而导致的竞争情况），并且要删除 *(generated)* 语言。

2.18.26 pushgit

weblate pushgit <project|project/component>

将提交的更改推送到上游的版本控制系统（VCS）仓库。

--force-commit

强制提交任何待处理的更改，然后再推送。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有部件。

备注： 如果开启了部件配置 中的:ref:*component-push_on_commit*，Weblate 会自动推送更改，这是默认的。

2.18.27 unlock_translation

weblate unlock_translation <project|project/component>

将给定的部件解锁，使它能够被翻译。

提示： 在您想要对下层仓库进行一些维护时有用。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有部件。

参见：

lock_translation

2.18.28 setupgroups

weblate setupgroups

配置默认群组，并可选择将所有用户分配到该默认群组。

--no-privs-update

关闭对现有组的自动更新（只添加新的）。

--no-projects-update

防止对现有项目的组的自动更新。这允许将新添加的组加入到现有项目中，请参见项目访问控制。

参见:

权限和内置角色列表

2.18.29 setuplang

weblate setuplang

将 Weblate 中的确定语言的列表更新。

--no-update

关闭现有语言的自动更新（只添加新的）。

2.18.30 updatechecks

weblate updatechecks <project|project/component>

更新对所有字符串的所有检查。

提示: 对检查进行主要更改的更新是有用的。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有部件。

2.18.31 updategit

weblate updategit <project|project/component>

取回远程版本控制系统（VCS）仓库并更新内部缓存。

您可以确定或者哪个项目或部件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有部件。

备注: 通常最好在仓库中配置钩子，来触发通知钩子，而不是常规的通过*updategit* 来投票。

2.19 公告

在 4.0 版本发生变更: 在以前的版本中，这个功能被称为白板消息。

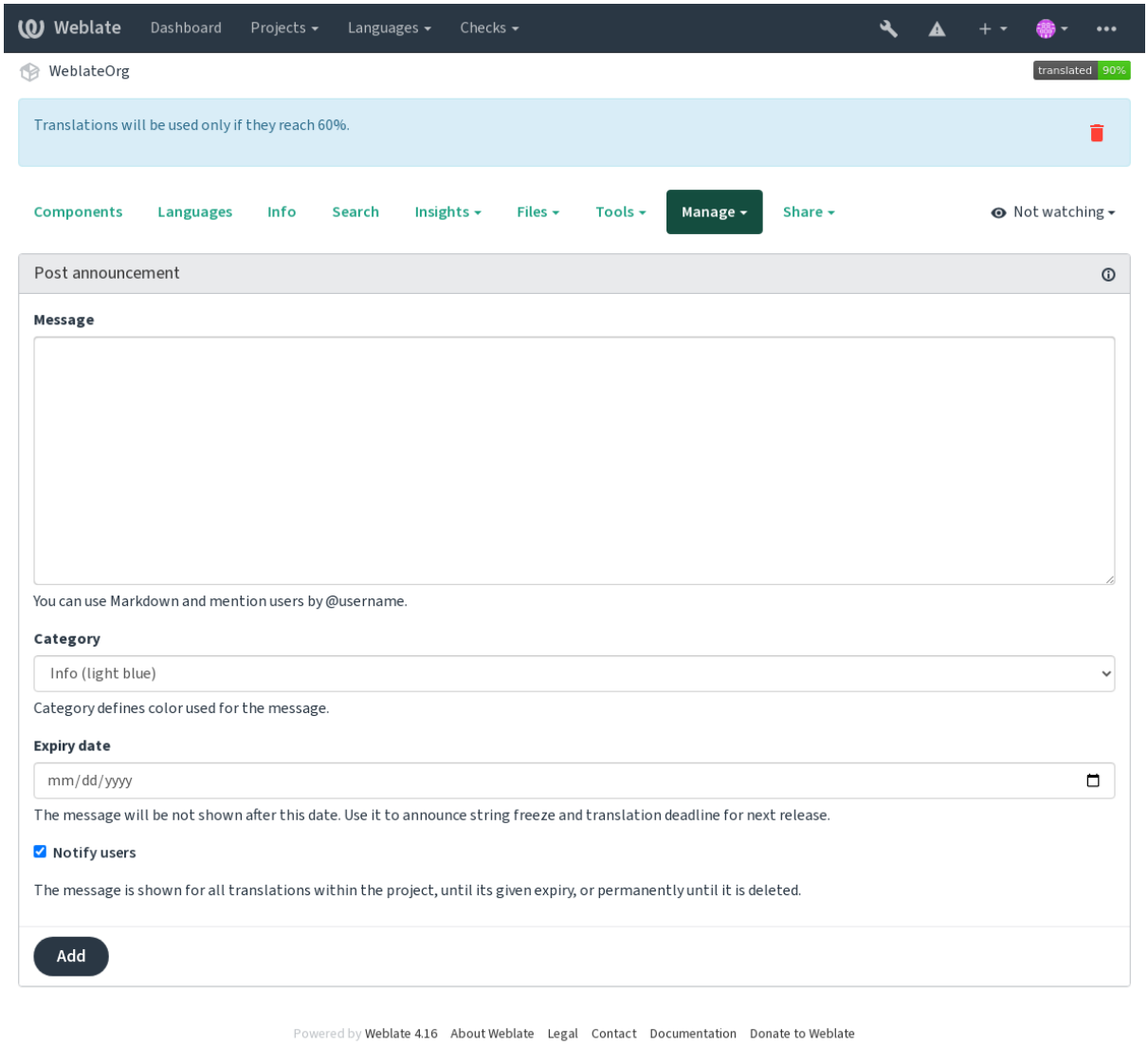
通过发布站点范围、每个项目、部件或语言的公告，向译者提供信息。

宣布目的、截止日期、状态或指定翻译目标。

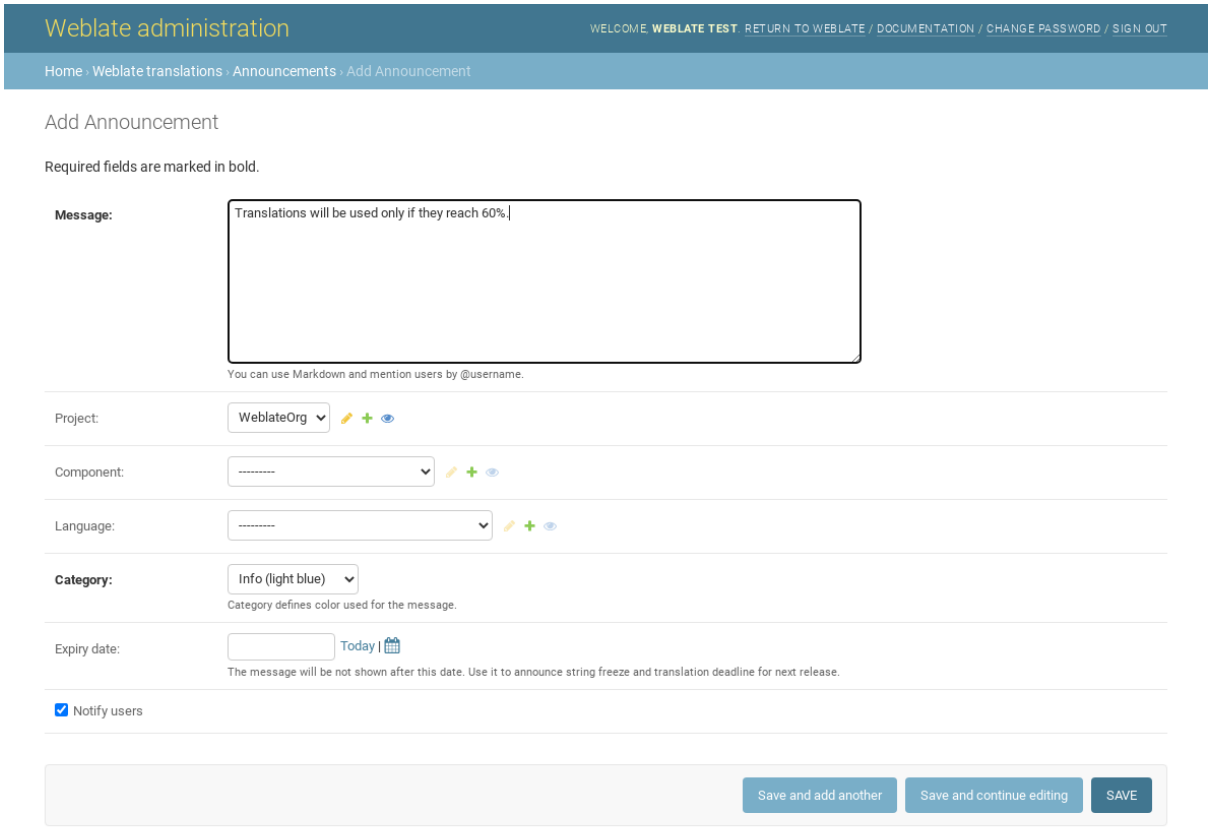
用户将收到关注的项目公告的通知（除非用户选择关闭）。

这对宣布网站的目的和指定翻译目标等各种事情都很有用。

可以通过 管理菜单中的 发布公告在每一层级发布公告:



还可以使用管理界面添加：



然后根据特定的上下文显示公告：

没有特定的上下文

显示在操作面板（着陆页）上。

特定项目

项目内显示，包括其所有的部件和翻译。

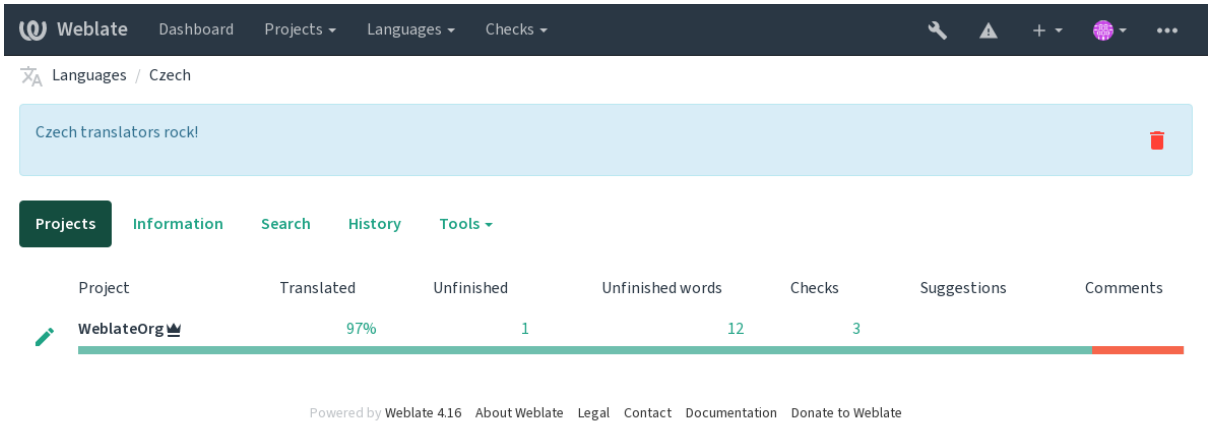
特定部件

对于给定的部件机器翻译来显示。

特定语言

显示在语言的概览和该语言的全部翻译中。

这就是语言概览页面上的样子：



2.20 部件列表

指定多个部件列表，作为显示在用户操作面板上的选项，用户可以从中挑选一个作为默认视图。请参阅操作面板 了解更多信息。

在 2.20 版本发生变更: 操作面板上出现的每个部件列表都会显示一个状态。

可以在管理界面的 部件列表部分指定部件列表的名称和内容。每个部件列表必须有一个显示给用户的名称，和一个在 URL 中代表它的标识串。

在 2.13 版本发生变更: 从管理界面更改匿名用户的操作面板设置，修改操作面板向未经身份验证用户展示的内容。

2.20.1 自动部件列表

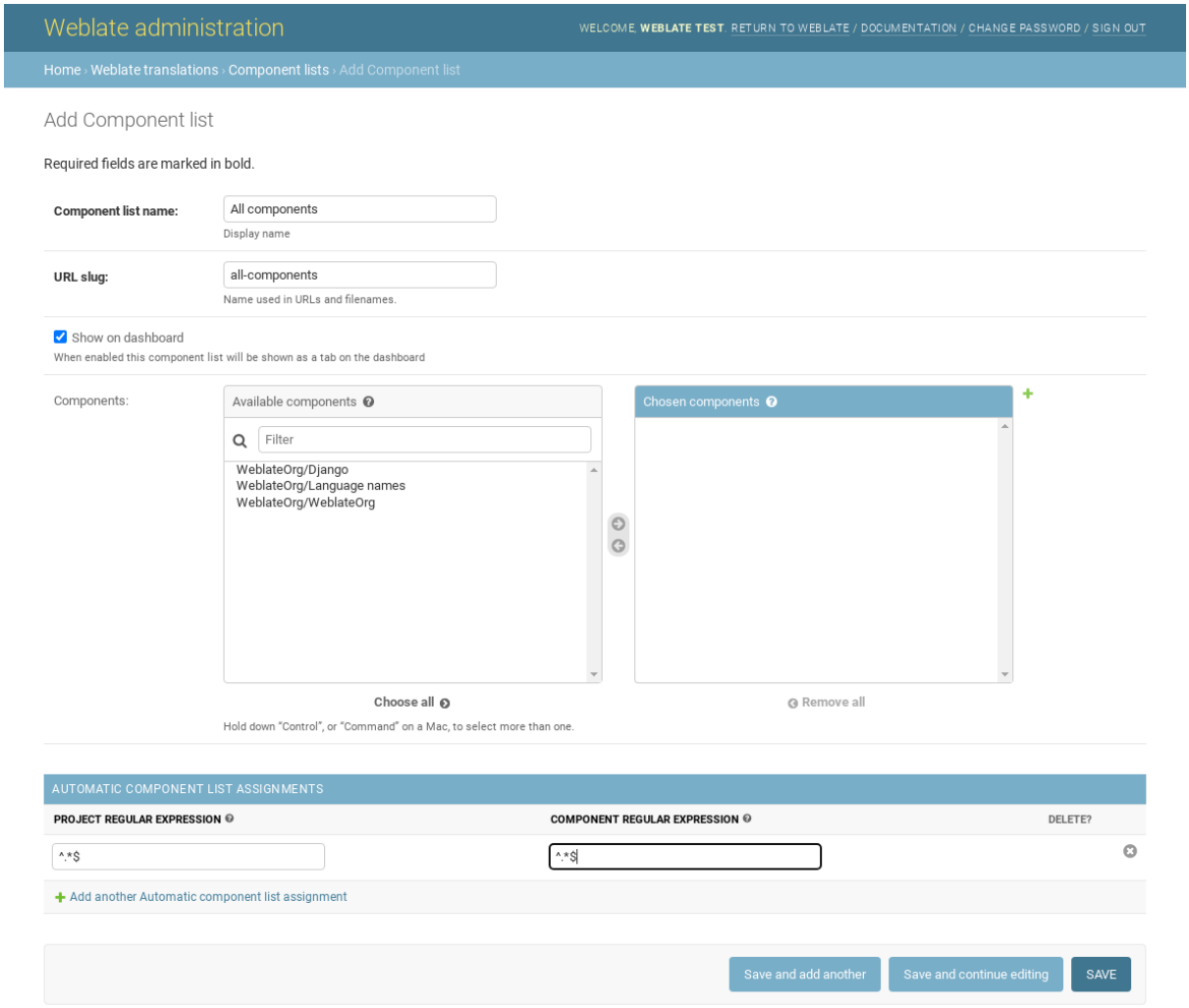
在 2.13 版本加入.

通过建立 自动分配部件列表规则，根据其标识串自动将部件添加到列表中。

- 这对于维护大型安装的部件列表很有用，或者你想在 Weblate 安装中拥有一个包含所有部件的部件列表。

提示: 制作一个包含你的 Weblate 安装时的所有部件的部件列表。

1. Define *Automatic component list assignment* with `^.*$` as regular expression in both the project and the component fields, as shown on this image:



2.21 可选的 Weblate 模块

可以获得几个可选的模块来配置您的设置。

2.21.1 Git 导出器

在 2.10 版本加入.

使用 HTTP(S) 为您提供对底层 Git 仓库的只读访问。

安装

1. 将 `weblate.gitexport` 添加到 `settings.py` 中安装的 apps 中：

```
INSTALLED_APPS += ("weblate.gitexport",)
```

2. 通过安装后迁移数据库，将现有的仓库导出：

```
weblate migrate
```

用法

模块自动钩入 Weblate，并且在*部件配置* 中设置导出仓库 URL。仓库在 Weblate URL 的 `/git/` 部分下是可以访问的，例如 `https://example.org/git/weblate/main/`。

公共可用项目的仓库可以被克隆而无需认证：

```
git clone 'https://example.org/git/weblate/main/'
```

对仓库的受限制的访问（使用 *Private* *访问控制* 或在*REQUIRE_LOGIN* 处于启用状态时）需要一个 API 令牌，获取位置在你的*用户个人资料*:

```
git clone 'https://user:KEY@example.org/git/weblate/main/'
```

提示： 成员或 *用户群组*与匿名用户默认通过 *访问仓库*和 *高级用户*角色访问公开项目。

2.21.2 账单

在 2.4 版本加入.

这在 Hosted Weblate 上用于确定付费套餐、跟踪发票和使用限制。

安装

1. Add `weblate.billing` to installed apps in `settings.py`:

```
INSTALLED_APPS += ("weblate.billing",)
```

2. 运行数据库迁移，来可选地为模块安装另外的数据库结构：

```
weblate migrate
```

用法

安装后您可以在管理界面控制账单。启用了账单模块的用户将在他们的用户个人资料 中看到新的 账单 选项卡。

账单模块额外允许项目管理员不是超级用户的情况下去新建新的项目和部件（请参见添加翻译项目和部件）。当后面的条件满足是这是可能的：

- 账单在其配置的限制下（任何过度的使用都会阻止新建项目/部件），并且被支付（如果价格为非零值的话）
- 用户是现有的带有账单项目的管理员，或者用户是账单的所有者（当为用户新建新的账单，而允许导入新的项目时，后者是必要的）。

在新建项目时，用户在访问多个账单的情况下，能够选择将项目记在哪个账单上。

2.21.3 法律声明

在 2.15 版本加入.

这用在 Hosted Weblate 上，来提供所需的法律文件。它开始时提供空白文档，会希望您填充文档中后面的模板：

legal/documents/tos.html
: 服务条款文档

legal/documents/privacy.html
: 隐私政策文档

legal/documents/summary.html
: 服务条款与隐私政策的简短概况

更改服务条款文档时，请调整*LEGAL_TOS_DATE* 迫使用户接受更新的文档。

备注： 可以在这个 Git 仓库 <https://github.com/WeblateOrg/wllegal/tree/main/wllegal/templates/legal/documents> 中获取 Hosted Weblate 的法律文件。

这些很可能对您没有直接的用处，但如果调整来满足您的需求时，以此作为起点可能会比较方便。

安装

1. Add `weblate.legal` to installed apps in `settings.py`:

```
INSTALLED_APPS += ("weblate.legal",)

# Optional:

# Social auth pipeline to confirm TOS upon registration/subsequent sign in
SOCIAL_AUTH_PIPELINE += ("weblate.legal.pipeline.tos_confirm",)

# Middleware to enforce TOS confirmation of signed in users
MIDDLEWARE += [
    "weblate.legal.middleware.RequireTOSMiddleware",
]
```

2. 运行数据库迁移，来可选地为模块安装另外的数据库结构：

```
weblate migrate
```

3. 编辑 `weblate/legal/templates/legal/` 文件夹中的而法律文件，与您的服务匹配。

用法

安装并编辑后，法律文件显示在 Weblate 界面中。

2.21.4 头像

头像在服务器端下载并缓存，来减少对默认服务的网站的泄露。通过为其配置的电子邮箱地址来取回头像的内建支持，可以使用*ENABLE_AVATARS* 来关闭。

当前的 Weblate 支持：

- Gravatar
- Libravatar

参见：

头像缓存, *AVATAR_URL_PREFIX*, *ENABLE_AVATARS*

2.21.5 针对垃圾电子邮件的保护

您可以使用 Akismet 服务来免受用户发送的垃圾侵扰。

1. 安装 *akismet* Python 模块（这已经包含在官方 Docker 镜像中）。
2. 获取 Akismet API 密钥。
3. 在 Docker 中存储为*AKISMET_API_KEY* 或 *WEBLATE_AKISMET_API_KEY*。

以下内容被发送到 Akismet 进行检查：

- 来自未经身份验证用户的建议
- 项目和部件描述及链接

备注： 这（除了其它事情以外）依赖于客户端的 IP 地址，适当的配置请参见：*在反向代理之后运行*。

参见：

在反向代理之后运行, *AKISMET_API_KEY*, *WEBLATE_AKISMET_API_KEY*

2.21.6 使用 GnuPG 为 Git 提交签名

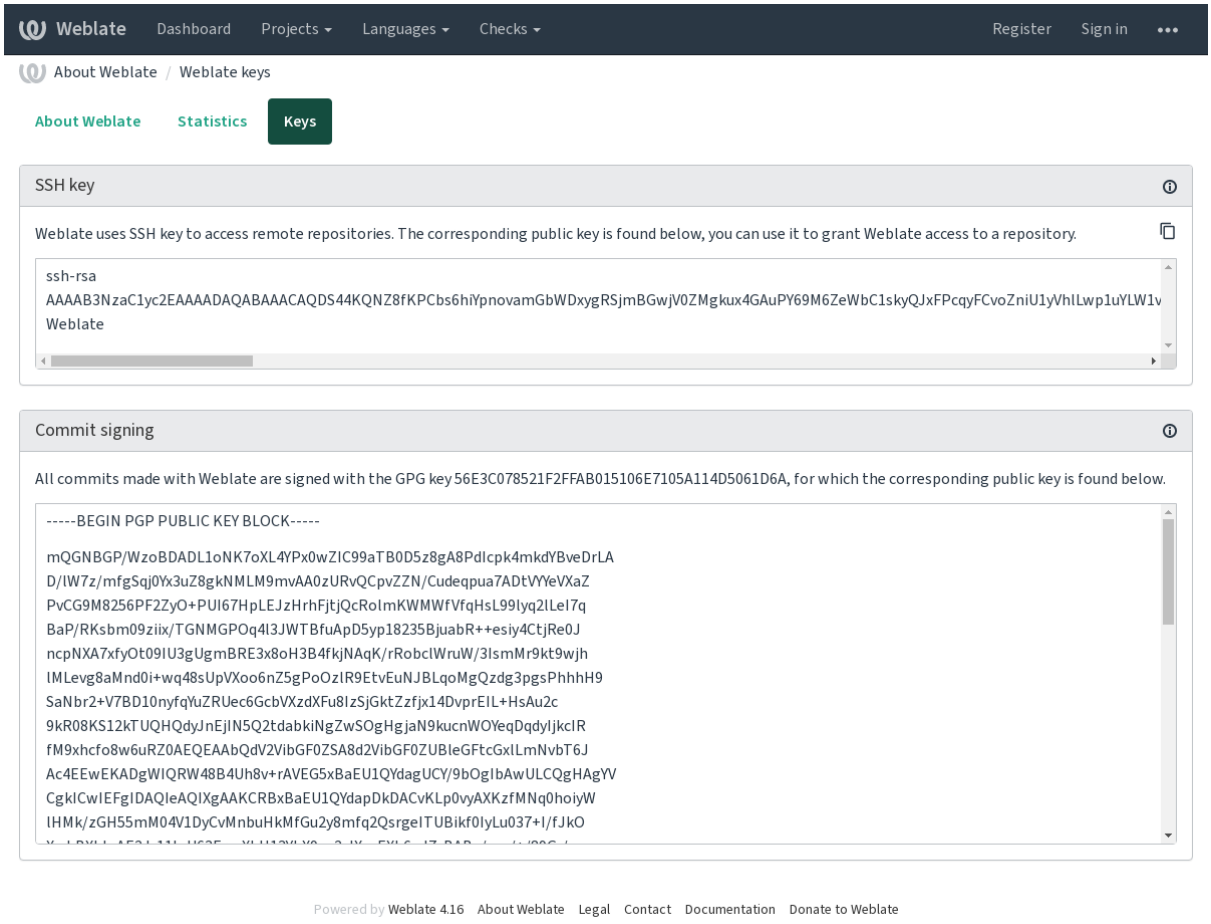
在 3.1 版本加入.

所有的提交均可由 Weblate 实例的 GnuPG 密钥为其签名。

1. Turn on *WEBLATE_GPG_IDENTITY*. (Weblate will generate a GnuPG key when needed and will use it to sign all translation commits.)

此功能需要安装 GnuPG 2.1 或更新版本。

您可以在*DATA_DIR* 中找到密钥，公钥显示在“关于”页面上：



2. Alternatively you can also import existing keys into Weblate, just set `HOME=$DATA_DIR/home` when invoking gpg.

参见:

WEBLATE_GPG_IDENTITY

2.21.7 频次限制

在 3.2 版本发生变更: 频次限制先择接受更细粒度的配置。

在 4.6 版本发生变更: 速率限制不再适用于超级用户。

Weblate 的一些操作受到频次限制。在*RATELIMIT_WINDOW* 的秒数内最多允许*RATELIMIT_ATTEMPTS* 次数的尝试。然后阻止用户*RATELIMIT_LOCKOUT* 时间。还有指定范围的设置，例如 RATELIMIT_CONTACT_ATTEMPTS 或 RATELIMIT_TRANSLATE_ATTEMPTS。下面的表格是可用范围的完整列表。

后面的操作受到频次限制：

名称	范围	允许的尝试	频次限制窗口	锁定时间
注册	REGISTRATION	5	300	600
将消息发送给管理员	MESSAGE	2	300	600
登录时的密码验证	LOGIN	5	300	600
网站范围的搜索	SEARCH	6	60	60
翻译	TRANSLATE	30	60	600
添加到术语表	GLOSSARY	30	60	600
开始翻译到一种新语言	LANGUAGE	2	300	600
创建新项目	PROJECT	5	600	600

如果用户没能在*AUTH_LOCK_ATTEMPTS* 的次数内登录，那么账户的密码验证将关闭，直到完成了重置密码过程为止。

这些设置也可以在 Docker 容器中应用，在设置名称中添加 WEBLATE_ 前缀例如 RATELIMIT_ATTEMPTS`变成:envvar:`WEBLATE_RATELIMIT_ATTEMPTS。

API 具有另外的速率限制设置，请参见*API 频次限制*。

参见:

频次限制, *在反向代理之后运行*, *API 频次限制*

2.21.8 Fedora Messaging 集成

Fedora Messaging 是基于 AMQP 的发布者，用于处理 Weblate 中发生的所有更改。您可以使用它来挂钩 Weblate 中发生的更改的附加服务。

Fedora 消息集成可以作为一个单独的 Python 模块 weblate-fedora-messaging 使用。请参阅 <https://github.com/WeblateOrg/fedora_messaging/> 了解安装说明。

2.22 定制 Weblate

使用 Django 和 Python 进行扩展和定制。将你的更改贡献给上游，让每个人都能受益。这降低了您的维护成本；当改变内部接口或重构代码时，Weblate 的代码会被照顾到。

警告: 内部界面与模板都不被认为是稳定的 API。请对每次升级都复查自己的定制，接口或其语义可能未经通知就进行更改。

参见:

为 Weblate 作贡献

2.22.1 建立 Python 模块

如果不熟悉 Python，你可以查看 适合初学者的 Python（英文），它解释了其基本内容并指向了高级教程。

要写入有定制 Python 代码（被称为模块）的文件，需要一个地方存储它，或者在系统路径（通常像 /usr/lib/python3.9/site-packages/ 的地方），或者在 Weblate 目录下，同样也添加到解释程序搜索路径下。

在 3.8-5 版本加入: 使用*using Docker* 时，你可将 Python 模块置于 /app/data/python/ (见*Docker 容器卷*) 以便它们可以由 Weblate 从:ref:‘settings override file <docker-settings-override>‘等处进行加载。

更好地是，将你的定制化转变为适当的 Python 包:

1. 为你的包建立文件夹（我们会使用 *weblate_customization* ）。

2. 在里面新建 'setup.py' 文件来描述包：

```
from setuptools import setup

setup(
    name="weblate_customization",
    version="0.0.1",
    author="Your name",
    author_email="yourname@example.com",
    description="Sample Custom check for Weblate.",
    license="GPLv3+",
    keywords="Weblate check example",
    packages=["weblate_customization"],
)
```

3. 建立定制代码的 Python 模块（也被成为 `weblate_customization` ）的文件夹。
4. 在里面建立 `__init__.py` 文件来确认 Python 可以导入模块。
5. 现在可以使用 **`pip install -e`** 安装这个包。更多信息可以在 Editable installs 中找到。
6. 模块一旦安装，就可以用在 Webalte 配置中（例如 `weblate_customization.checks.FooCheck` ）。

你的包结构应该看起来像这样：

```
weblate_customization
├── setup.py
└── weblate_customization
    ├── __init__.py
    ├── addons.py
    └── checks.py
```

可以在 <https://github.com/WeblateOrg/customize-example> 找到定制 Weblate 的示例，它涵盖了下面描述的所有题目。

2.22.2 更改标志

1. 建立简单的 Django app 来包含想要覆盖的静态文件（请参见*建立 Python 模块* ）。

   品牌出现在后面的文件中：

   `icons/weblate.svg`
   导航栏中显示的标志。

   `logo-*.png`
   网页图标取决于屏幕分辨率和浏览器。

   `favicon.ico`
   传统浏览器使用的网页图标。

   `weblate-*.png`
   机器人或匿名用户使用的头像。一些浏览器将这些图标用作快捷方式图标。

   `email-logo.png`
   在通知电子邮件中使用。

2. 把它添加到:setting:*django:INSTALLED_APPS*:

```
INSTALLED_APPS = (
    # Add your customization as first
    "weblate_customization",
    # Weblate apps are here…
)
```

3. 运行 `weblate collectstatic --noinput`，来收集提供给客户端的静态文件。

参见:

How to manage static files (e.g. images, JavaScript, CSS), 为静态文件提供服务

2.22.3 定制的质量检查、附加组件和自动修复

要在 Weblate 中安装你的定制的自动修正，编写自己的检查 或编写附加组件 代码:

1. 将文件放进你的包含 Weblate 定制的 Python 模块中（请参见建立 *Python* 模块 ）。
2. 在专用设置（*WEBLATE_ADDONS* 、*CHECK_LIST* 或*AUTOFIX_LIST* ）中将其完全合法的路径添加到 Python 类中:

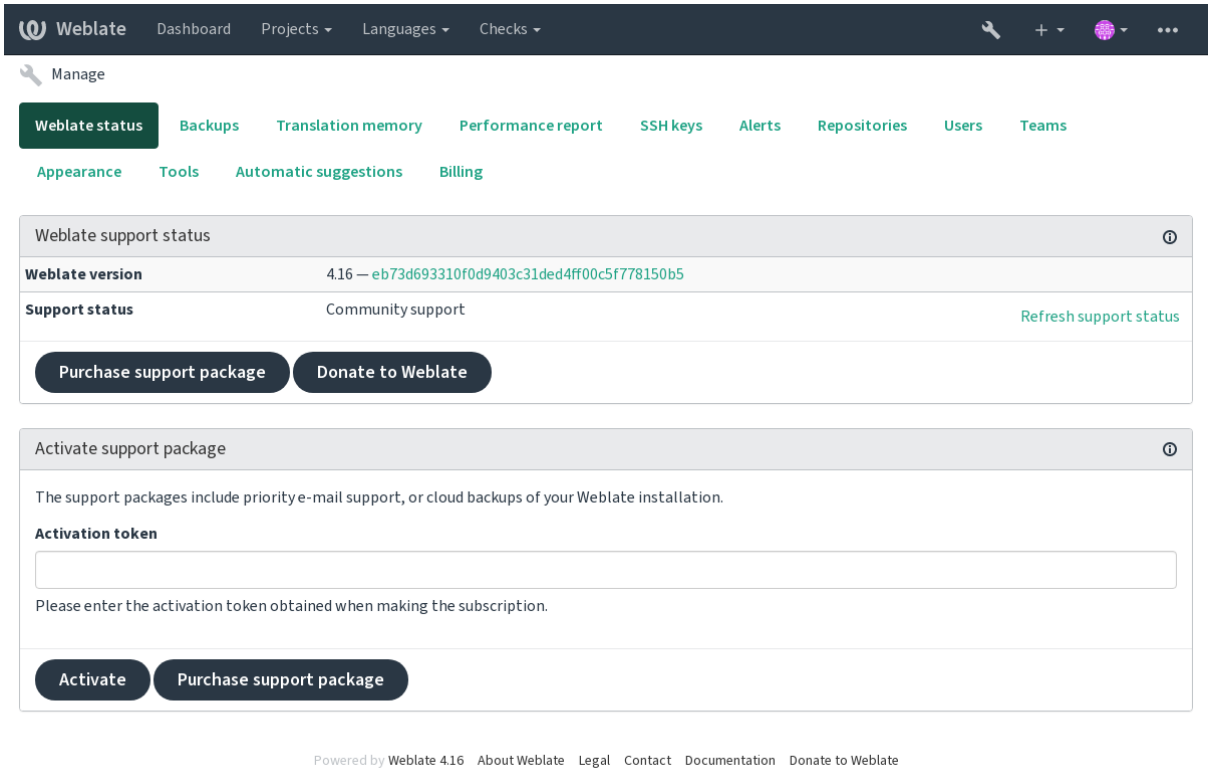
```
# Checks
CHECK_LIST += ("weblate_customization.checks.FooCheck",)

# Autofixes
AUTOFIX_LIST += ("weblate_customization.autofix.FooFixer",)

# Add-ons
WEBLATE_ADDONS += ("weblate_customization.addons.ExamplePreAddon",)
```

参见:

定制的自动修正, 编写自己的检查, 编写附加组件, 从附加组件执行脚本

2.23 管理界面

管理界面在 `/manage/` 下面提供管理设置。它对于具有管理特权的登录用户是可用的，通过使用右上角的扳手图标来访问:



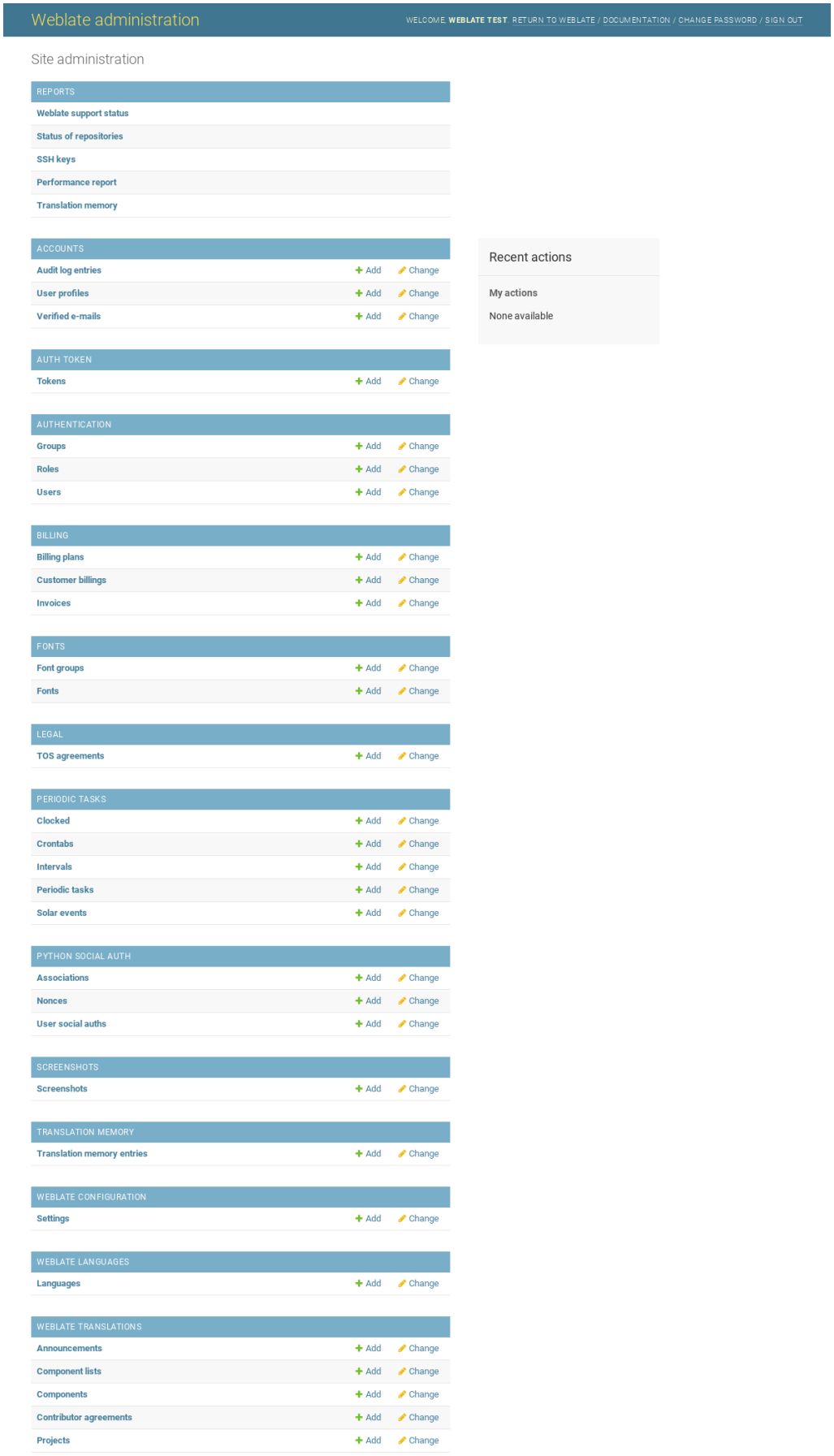
它包括您的 Weblate 的基本概况:

- 支持状态，请参见从 *Weblate* 获取支持
- 备份，请参见备份和移动 *Weblate*
- 共享的翻译记忆库，见翻译记忆库
- 性能报告，来复查 Weblate 的健康状况和 Celery 队列的长度
- SSH 密钥管理，见*SSH* 仓库
- 所有部件的警报概述，见 alerts

2.23.1 Django 管理界面

警告： 请小心使用，因为这是一个底层接口。多数情况下你不应该使用它，因为通过 Weblate 用户界面或 API 你可以舒服地访问多数设置项。

可以在这里管理数据库中存储的对象，如用户、翻译和其他设置：


Weblate administration
WELCOME, WEBLATE TEST. RETURN TO WEBLATE / DOCUMENTATION / CHANGE PASSWORD / SIGN OUT
Site administration
REPORTS
Weblate support status
Status of repositories
SSH keys
Performance report
Translation memory
ACCOUNTS
Audit log entries + Add Change
User profiles + Add Change
Verified e-mails + Add Change
AUTH TOKEN
Tokens + Add Change
AUTHENTICATION
Groups + Add Change
Roles + Add Change
Users + Add Change
BILLING
Billing plans + Add Change
Customer billings + Add Change
Invoices + Add Change
FONTS
Font groups + Add Change
Fonts + Add Change
LEGAL
TOS agreements + Add Change
PERIODIC TASKS
Clocked + Add Change
Crontabs + Add Change
Intervals + Add Change
Periodic tasks + Add Change
Solar events + Add Change
PYTHON SOCIAL AUTH
Associations + Add Change
Nonces + Add Change
User social auths + Add Change
SCREENSHOTS
Screenshots + Add Change
TRANSLATION MEMORY
Translation memory entries + Add Change
WEBLATE CONFIGURATION
Settings + Add Change
WEBLATE LANGUAGES
Languages + Add Change
WEBLATE TRANSLATIONS
Announcements + Add Change
Component lists + Add Change
Components + Add Change
Contributor agreements + Add Change
Projects + Add Change
Recent actions
My actions
None available

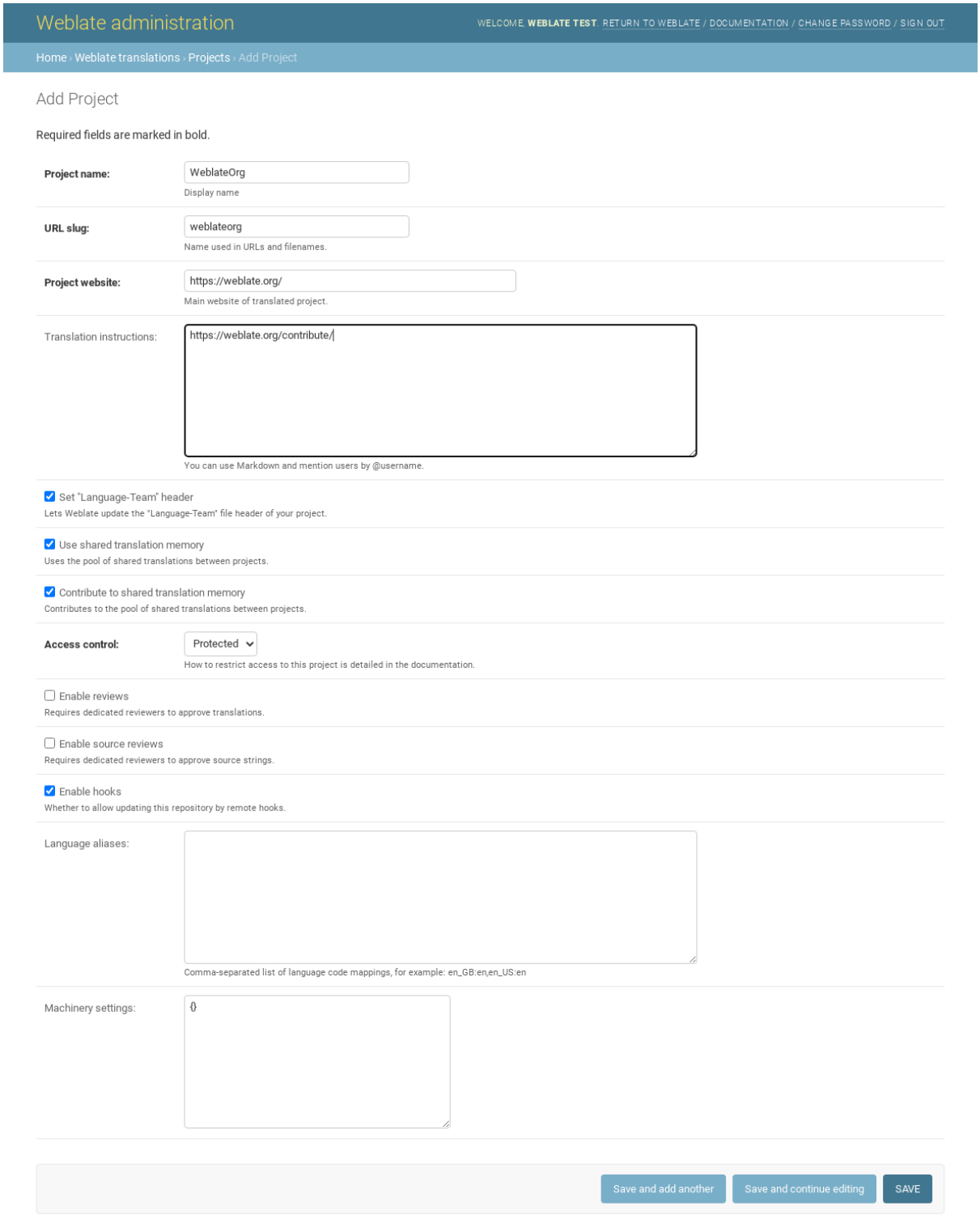
在 报告部分，可以检查网站的状态，为生产设置 进行调整，或者管理用于访问的 SSH 密钥访问仓库。

管理任意部分下的数据库对象。最有趣的也许是 *Weblate* 翻译，你可以在这里管理可翻译的项目，请参见项目配置 和部件配置。

Weblate 语言保持语言定义，在语言定义 中进一步解释。

添加项目

添加项目作为所有部件的容器。通常可以为一部分软件或图书（各自参数的信息请参见项目配置 ）来建立一个项目：


Weblate administration
WELCOME, WEBLATE TEST. RETURN TO WEBLATE / DOCUMENTATION / CHANGE PASSWORD / SIGN OUT
Home › Weblate translations › Projects › Add Project
Add Project
Required fields are marked in bold.
Project name:
WeblateOrg
Display name
URL slug:
weblateorg
Name used in URLs and filenames.
Project website:
https://weblate.org/
Main website of translated project.
Translation instructions:
https://weblate.org/contribute/
You can use Markdown and mention users by @username.
Set "Language-Team" header
Lets Weblate update the "Language-Team" file header of your project.
Use shared translation memory
Uses the pool of shared translations between projects.
Contribute to shared translation memory
Contributes to the pool of shared translations between projects.
Access control:
Protected
How to restrict access to this project is detailed in the documentation.
Enable reviews
Requires dedicated reviewers to approve translations.
Enable source reviews
Requires dedicated reviewers to approve source strings.
Enable hooks
Whether to allow updating this repository by remote hooks.
Language aliases:
Comma-separated list of language code mappings, for example: en_GB:en,en_US:en
Machinery settings:
{}
Save and add another
Save and continue editing
SAVE

参见:

项目配置

双语部件

一旦添加了一个项目，就可以添加翻译部件了。(关于各自参数的信息，请参见部件配置):


Weblate administration
Add Component
Component name:
URL slug:
Project:
Version control system:
Source code repository:
File mask:
File format:
Commit message when translating:
Commit message when adding translation:
Commit message when removing translation:
Commit message when merging translation:
Commit message when add-on makes a change:
Merge request message:
Source language:
Language filter:
Priority:
Share in projects:
Glossary color:
Save and add another
Save and continue editing
SAVE

参见:

部件配置, 双语和单语格式

单语言部件

为了使这些翻译更容易，提供了模板文件，包含了各自源语言的对应信息 ID（通常为英语）。(对于各自参数的信息，请参见部件配置)：


Weblate administration
Add Component
Required fields are marked in bold.
Component name:
URL slug:
Project:
Version control system:
Source code repository:
File mask:
Monolingual base language file:
File format:
Translation flags:
Enforced checks:
Translation license:
Contributor agreement:
Adding new translation:
Merge style:
Commit message when translating:
Commit message when adding translation:
Commit message when removing translation:
Commit message when merging translation:
Commit message when add-on makes a change:
Merge request message:
Age of changes to commit:
Source language:
Language filter:
Priority:
Glossary color:
Save and add another
Save and continue editing
SAVE

参见:

部件配置, 双语和单语格式

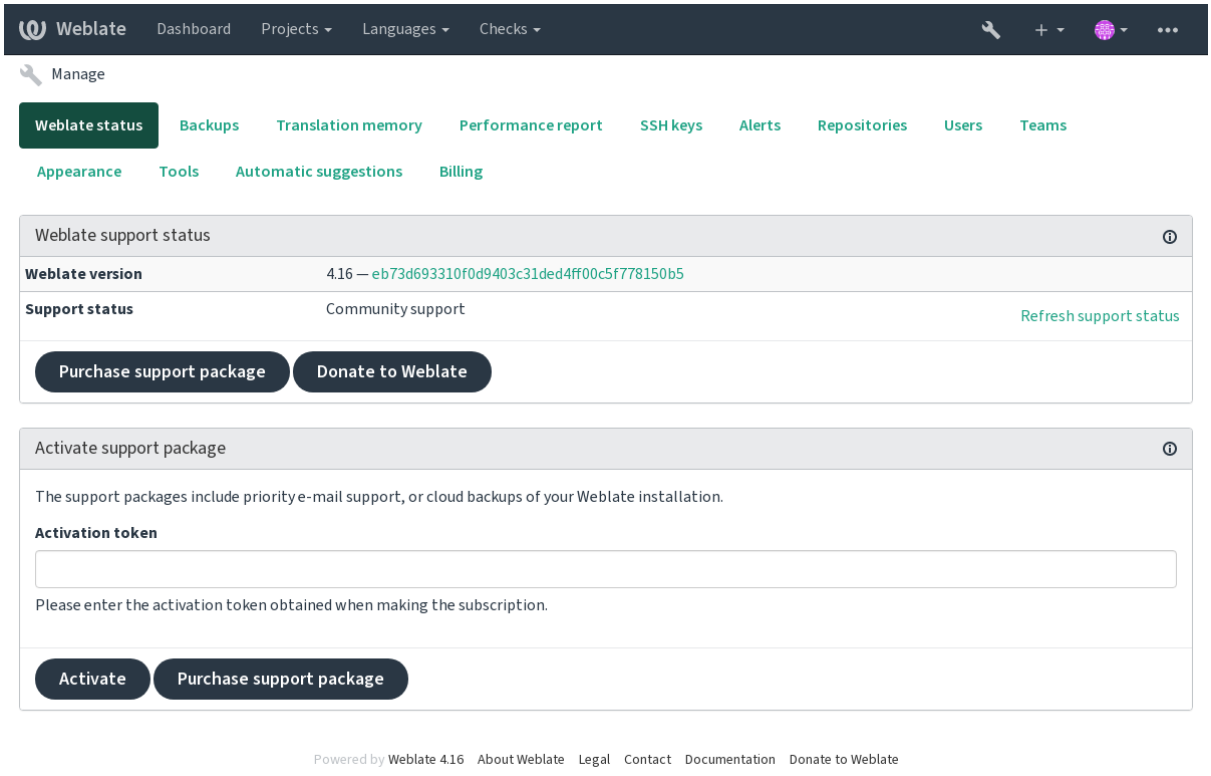
2.24 从 Weblate 获取支持

Weblate 是一款 copylefted 的自由软件，由社区进行支持。订阅者无需支付额外费用便可以获得优先的支持；预付支持服务包可供所有人使用。您可以在 <https://weblate.org/support/> 找到更多关于当前支持服务的信息。

2.24.1 集成支持

在 3.8 版本加入.

你可以视需要将购买的支持包集成进 Weblate 订阅管理 界面，您可以在那里找到所购支持包的链接。关于安装的基本实例细节也会以这种方式报告给 Weblate。



2.24.2 提交给 Weblate 的数据

- 配置 Weblate 实例的 URL
- 您的网站标题
- 您运行的 Weblate 版本
- 您 Weblate 数据库中一些对象的记录（项目、部件、语言、源字符串和用户）
- 您实例的 SSH 公钥

此外，当发现 *Weblate* 开启时:

- 公共项目列表（名称、网址和网站）

没有提交其它数据。

2.24.3 集成服务

- 查看您的支持包是否合法
- *Weblate 提供的备份存储*
- *发现 Weblate*

提示： 购买的支持包在购买时已经激活，并且不必集成它们就可以使用。

2.24.4 发现 Weblate

在 4.5.2 版本加入.

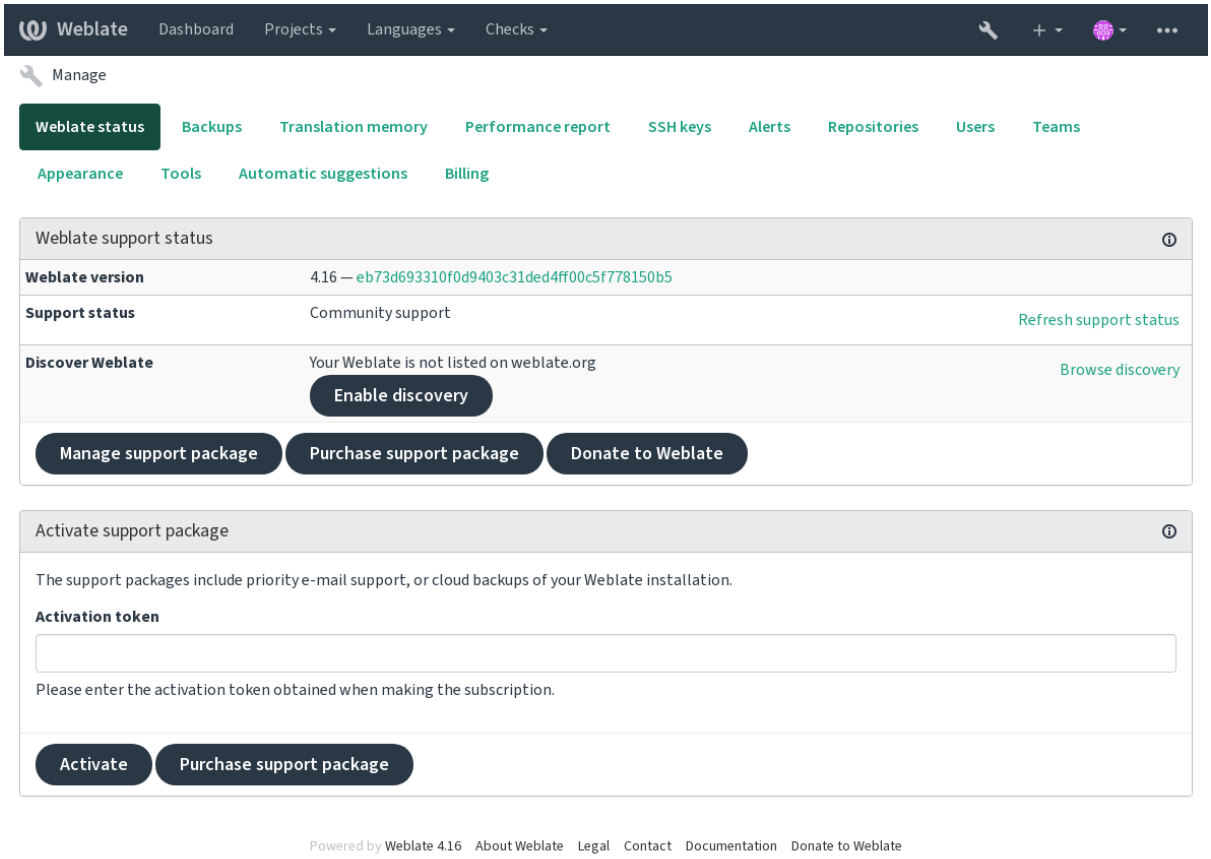
备注： 该功能目前处于早期测试阶段。

Discover Weblate 是一项选择服务，使用户更容易找到 Weblate 服务器和社区。用户可以浏览 <https://weblate.org/discover/> 上的注册服务，并找到那里的项目进行贡献。

列入

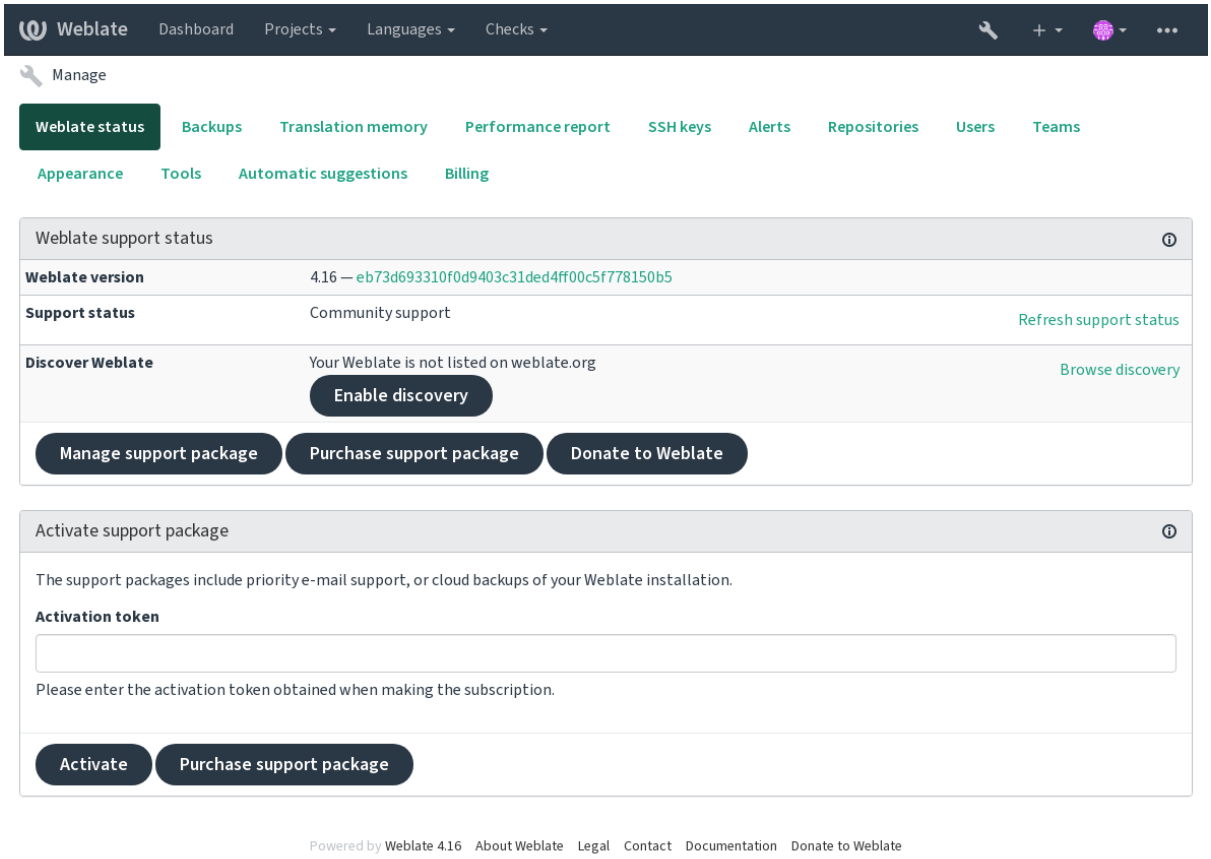
提示： 参与 Discover Weblate 会使 Weblate 提交一些关于你的服务器的信息，请参阅*提交给 Weblate 的数据*。

要在 Discover Weblate 中列出具有有效支持订阅的服务器（请参阅*集成支持*），您要在管理面板中打开它：



在 Discover Weblate 中列出没有订阅支持的服务器：

1. 注册 Weblate，注册地址 <https://weblate.org/user/>
2. 在发现数据库注册你的 Weblate 服务器，注册地址 <https://weblate.org/subscription/discovery/>
3. 在您的 Weblate 中确认服务激活，并在您的 Weblate 管理页面中使用 *Enable discovery* 按钮打开发现列表：



定制列表

您可以在 <https://weblate.org/user/> 上提供文本和图像（570 x 260 像素）来自定义列表。

2.25 法律文件

备注： 在此。您可以找到您在某些法律管辖区操作 Weblate 可能需要的各种法律信息。它是作为一种指导手段提供的，不保证其准确性或正确性。确保您对 Weblate 的使用符合所有适用的法律和法规，最终是您的责任。

2.25.1 许可合规

Weblate 具有符合 REUSE 3.0 的许可证规范。

2.25.2 ITAR 和其他出口管制

Weblate 可以在您自己的数据中心或虚拟私有云中运行。因此，它可用于存储 ITAR 或其他出口管制信息，但最终用户有责任确保此类合规性。

Hosted Weblate 服务未经过 ITAR 或其他出口管制合规性审核，目前不提供按国家/地区限制翻译访问的功能。

2.25.3 美国加密控制

Weblate 不包含任何加密代码，但可能会受到出口管制，因为它使用第三方组件利用加密进行身份验证、数据完整性和机密性。

Weblate 很可能会被归类为 ECCN 5D002 或 5D992，并且作为公开可用的自由软件，它不应受 EAR 约束（请参阅 '加密项目不受 EAR 约束 <https://www.bis.doc.gov/index .php/policy-guidance/encryption/1-encryption-items-not-subject-to-the-ear>'_）。

Weblate 使用的软件部件（仅列出与加密功能相关的组件）：

Python
: 见 https://wiki.python.org/moin/PythonSoftwareFoundationLicenseFaq#Is_Python_subject_to_export_laws.3F

GnuPG
: Weblate 可选择使用

Git
: Weblate 可选择使用

curl
: 由 Git 使用

OpenSSL
: 被 Python 和 curl 使用

加密密钥的强度取决于 Weblate 的配置以及与之交互的第三方组件，但在任何体面的设置中，它将包括所有出口受限的加密功能：

- 对称算法超过 56 位
- 非对称算法超过 512 位的整数因式分解
- 非对称算法在大小大于 512 位的有限域的乘法群中计算离散对数
- 非对称算法中超过 112 位的组中的离散对数

Weblate 没有任何加密激活功能，但可以以不涉及加密代码的方式进行配置。加密功能包括：

- 使用安全传输协议（HTTPS）访问远程服务器
- 为代码的提交生成签名（PGP）

参见：

开源软件出口管制 (EAR)

CHAPTER 3

贡献者文档

3.1 为 Weblate 作贡献

有几十种方法来改进 Weblate。你可以选择一个你觉得舒服的方式，它可以是编程、图形设计、文档、赞助，或一个想法：

- *在 Weblate 中汇报问题*
- *开始为 Weblate 贡献代码*
- *为 Weblate 模块作贡献*
- *翻译 Weblate*
- *为 Weblate 文档作贡献*
- *Weblate 讨论*
- *资助 Weblate 开发*

3.1.1 翻译 Weblate

Weblate 使用 Weblate 持续进行自身的 本地化翻译工作。尽你所能，使 Weblate 能以尽可能多的人类语言提供，让 Weblate 离用户更近！

如果您发现源字符串中可能存在错误，您可以在 Weblate 编辑器中用评论标记它。这样，它可以被讨论和纠正。如果你确定，你也可以点击 源字符串位置部分中的链接并提交一个拉取请求并进行更正。

3.1.2 为 Weblate 文档作贡献

欢迎您改进您选择的文档页面。通过单击页面右上角的 *在 GitHub 上编辑*按钮轻松完成此操作。

请在写作时遵守以下准则：

1. 如果部分文档有效请不要将其删除。
2. 请使用清晰易懂的语言。你正在写技术文档，而不是一首诗。并非所有文档读者都是母语人士，请深思熟虑。
3. 如果你不确定不要害怕询问。如果你在编辑时必须询问某些功能。在你得到答案之前不要改变它的文档。这意味着你要么改变。要么询问。不要同时做这两件事。
4. 在遵循文档的同时通过执行描述的操作来验证您的更改。
5. 发送带有小块更改的 PR 以便更轻松更快速地查看和合并。
6. 如果你想重写和改变一篇大文章的结构，请分两步进行：
    1. 重写
    2. 重写的内容经审校、润色并被合并后，在另一个 PR 中修改段落的结构。

提示： 你可以 翻译文档。

3.1.3 扩展内置语言定义

语言定义在 weblate-language-data 仓库 中。

欢迎您将缺少的语言定义添加到 `languages.csv`, 其他文件是从该文件生成的。

3.1.4 Weblate 讨论

如果你有一个想法，但不确定它是否适合作为一个 issue，不用担心，你可以加入 GitHub discussions 中的社区对其进行讨论。

3.1.5 资助 Weblate 开发

你可以在 捐赠页面 上促进 Weblate 的发展。收集到的资金会用于为自由软件项目提供免费托管服务和支持 Weblate 的进一步开发。具体内容如筹款目标、作为一名自豪的资助者可获得的奖励等选项，请查看**‘捐赠页面 ‘_**。

资助过 Weblate 的支持者

Weblate 支持者名单：

- Yashiro Ccs
- Cheng-Chia Tseng
- Timon Reinhard
- Cassidy James
- Loic Dachary
- Marozed
- https://freedombox.org/
- GNU Solidario (GNU Health)

- BallotReady
- Richard Nespithal
- MyExpenses.Mobi
- Michael Totschnig

想要出现在名单中吗？请查看 捐赠给 Weblate 上的选项。

3.2 开始为 Weblate 贡献代码

要理解 Weblate 源代码，请查看*Weblate 源代码*、*Weblate 前端* 和*Weblate 内部*。

3.2.1 从代码库开始

想熟悉 Weblate 代码库，不如先试试对带有 good first issue 标签的那些 bug 下手。

欢迎你在无人请求的情况下开始处理这些问题。只要在相关 issue 中宣布一下，让别人知道有人正在处理它。

3.2.2 本地运行 Weblate

开始 Weblate 开发的最舒适的方法是按照从源码安装。它将给您一个带有可编辑的 Weblate 源代码的虚拟环境。

1. 克隆 Weblate 源码：

```
git clone https://github.com/WeblateOrg/weblate.git
cd weblate
```

2. 创建一个虚拟环境：

```
virtualenv .venv
.venv/bin/activate
```

3. 安装 Weblate（这需要一些系统依赖项，请参见从源码安装）：

```
pip install -e .
```

3. 安装用于开发的所有依赖项：

```
pip install -r requirements-dev.txt
```

4. 启动一台开发服务器：

```
weblate runserver
```

5. 你可能还想启动 Celery workers，这取决于你的配置：

```
./weblate/examples/celery start
```

6. 若要运行测试（更多细节请参见本地测试）：

```
. scripts/test-database.sh
./manage.py test
```

参见：

从源码安装

3.2.3 在 Docker 中本地运行 Weblate

如果你已经安装了 Docker 和 docker-compose，只需运行以下命令即可启动开发环境：

```
./rundev.sh
```

它将新建一个 Docker 镜像并启动它。Weblate 运行在 <http://127.0.0.1:8080/> 上，并且您可以以用户 `admin` 的身份登录，密码是 `admin`。新安装是什么都没有的，所以接着您可能想添加翻译项目和部件。

相关 `Dockerfile` 和 `docker-compose.yml` 位于的 `dev-docker` 目录中。

脚本还接受一些参数，要执行测试，以 `test` 参数来运行，然后指定任意 `test` 参数，例如，只在 `weblate.machine` 模块中运行测试：

```
./rundev.sh test --failfast weblate.machine
```

备注： 小心在运行测试前您的 Docker 容易活动并运行。您可以通过运行 `docker ps` 命令来检查。

展示日志：

```
./rundev.sh logs
```

为了停止后台容器，运行：

```
./rundev.sh stop
```

运行没有任何参数的脚本将重建 Docker 容器并重启动它。

备注： 这不是用于生产的合适设置，因它包括几个不安全的小技巧，但它们会使开发更容易。

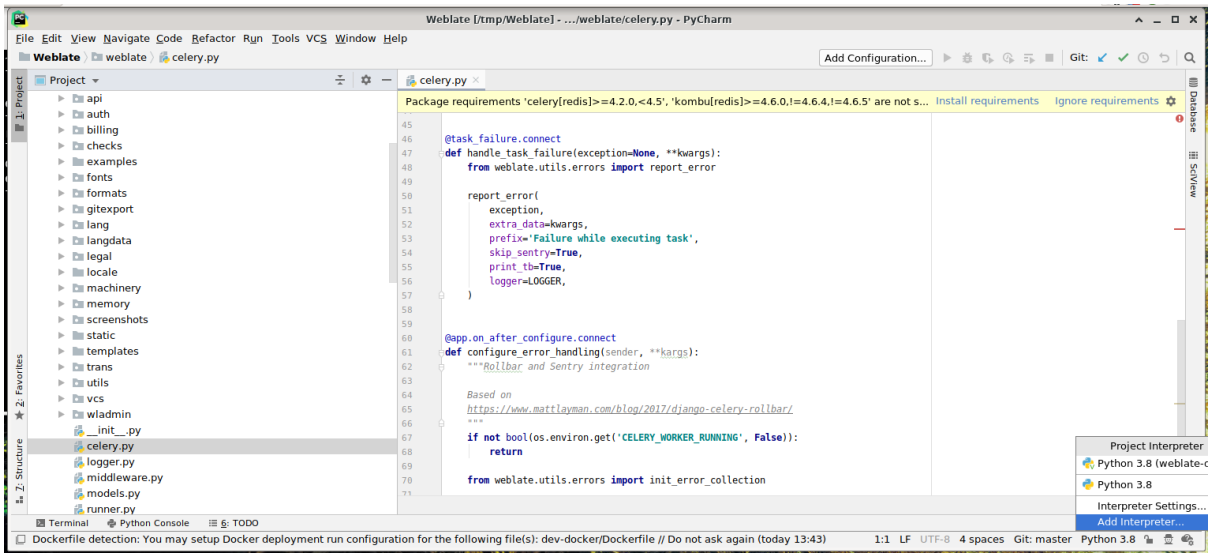
3.2.4 引导你的开发实例

您会想要使用 *`import_demo`* 来新建演示翻译，并且使用 *`createadmin`* 来创建一名管理用户。

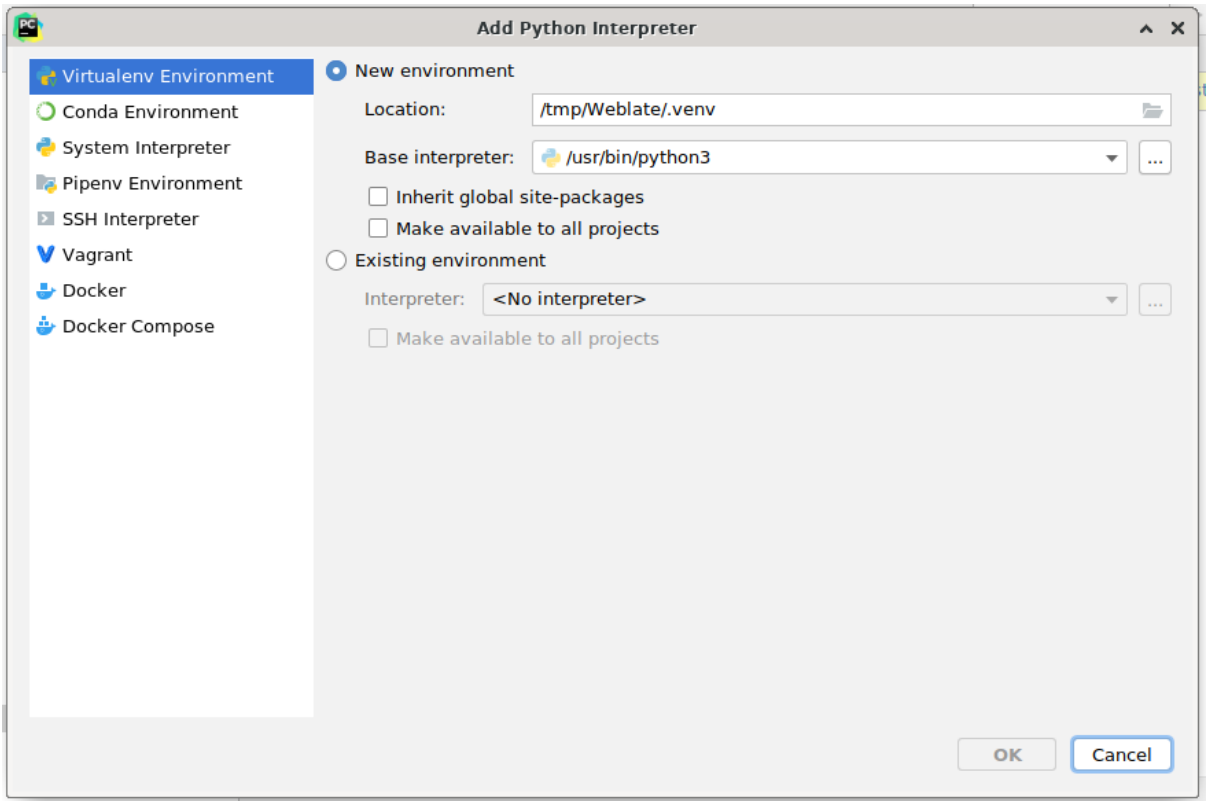
3.2.5 使用 PyCharm 为 Weblate 编写代码

PyCharm 是一个著名的 Python IDE，以下是一些指导准则，可以帮助你在其中设置你的 Weblate 项目。

考虑到你刚刚将 GitHub 仓库复制到一个文件夹中，只需使用 PyCharm 打开它。一旦 IDE 打开，第一步要做的是指定你想要使用的解释器：

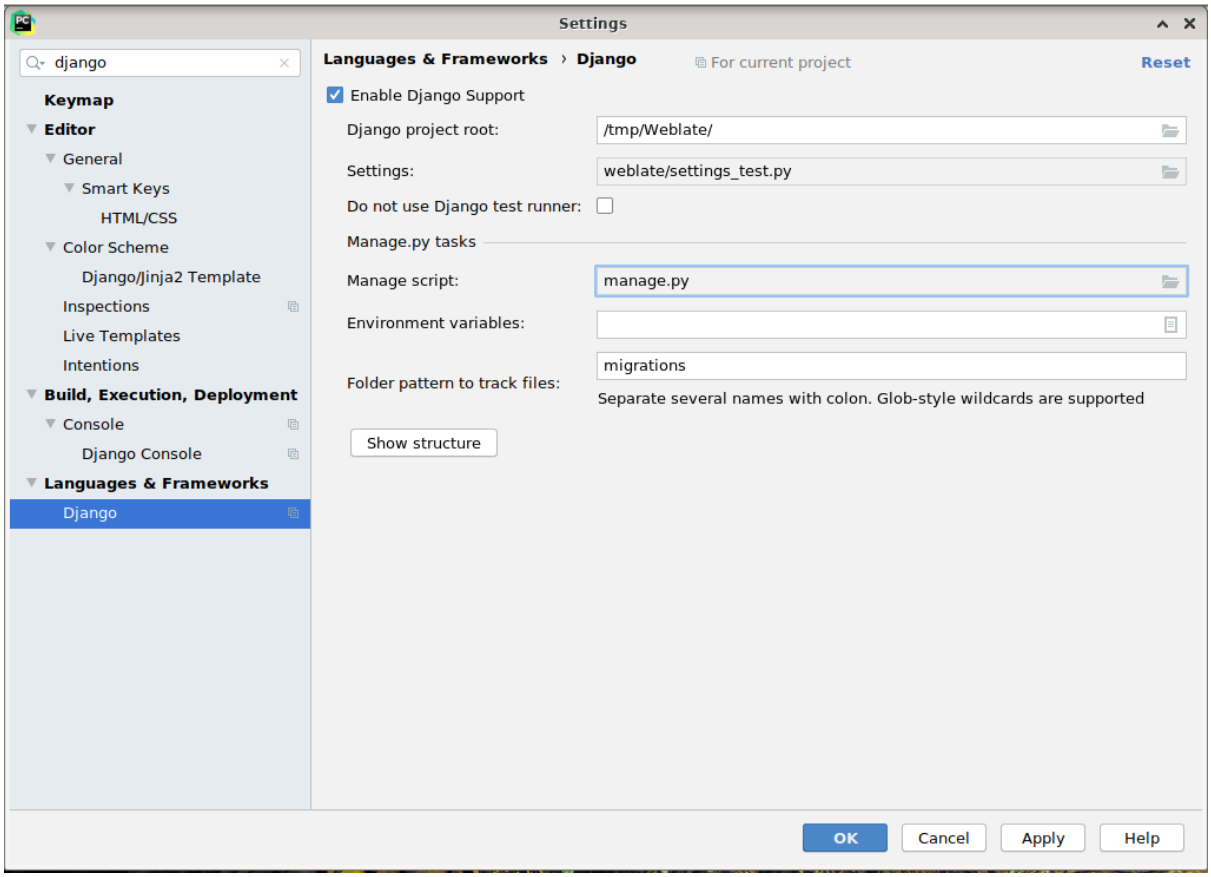


你可以选择让 PyCharm 为您创建 virtualenv（虚拟环境），或者选择一个现有的：



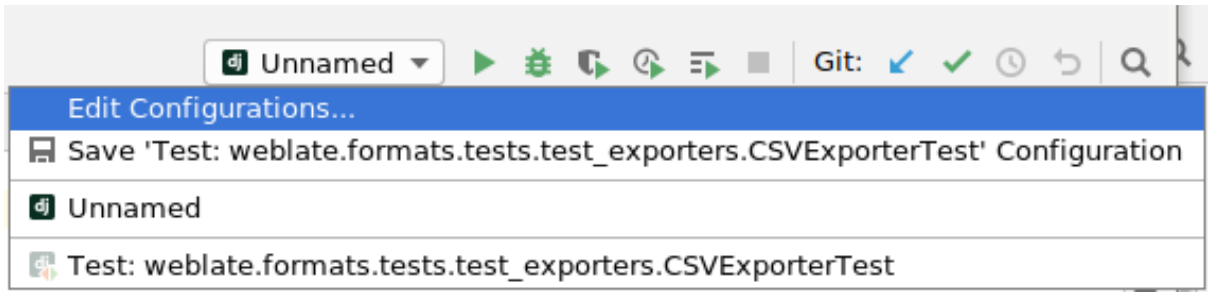
一旦设置了解释程序别忘了安装依赖项：要么通过控制台（IDE 的控制台默认情况下会直接使用你的 virtualenv），或者当你得到一个关于缺少依赖项的警告时通过接口。

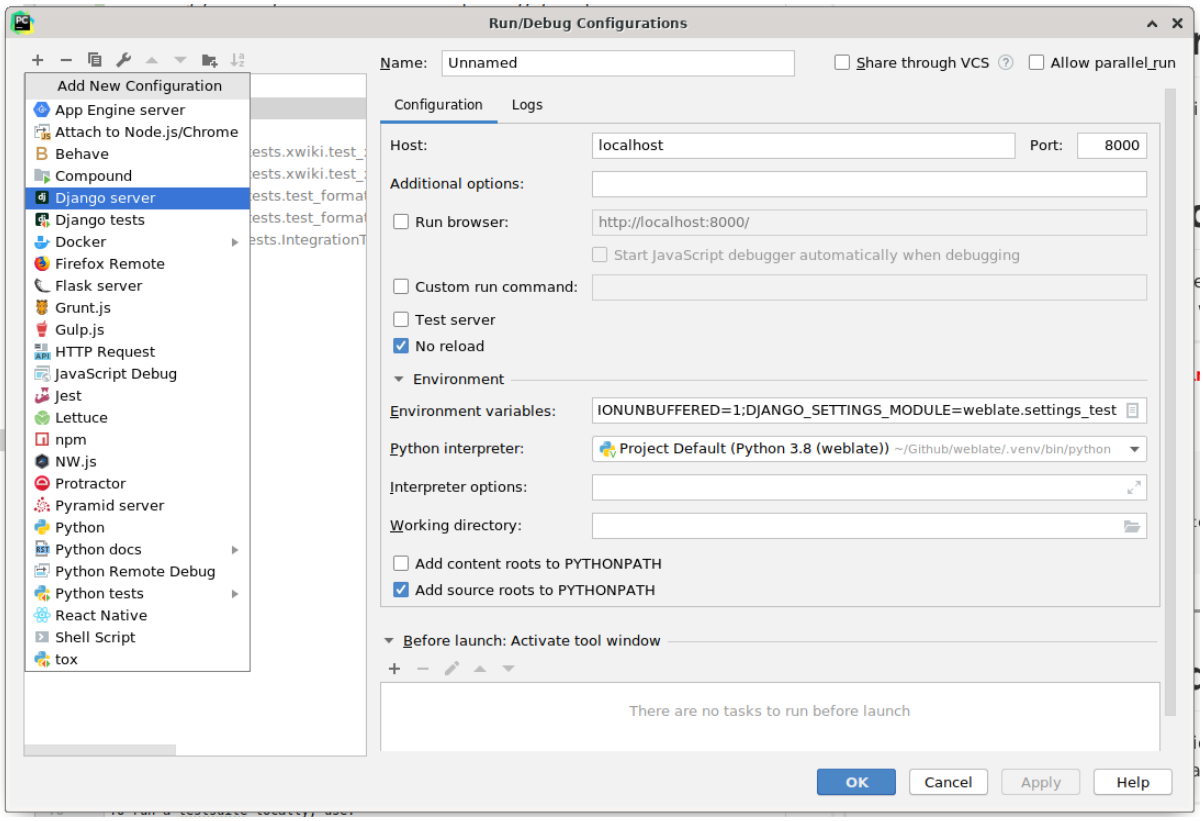
第二步是设置正确的信息来在 PyCharm 中本地使用 Django：目的是能够立即触发 IDE 中的单元测试。为此，你需要指定该 Django 项目的根路径及其设置路径：



请注意，*Django project root* 是仓库的实际根目录，而不是 Weblate 子目录。关于设置，你可以使用仓库中的 `weblate/settings_test.py`，您也可以创建自己的设置并将其设置在那里。

最后一步是运行服务器，并将断点放置在代码中而能够调试它。这通过新建新的 *Django Server* 配置来完成：





提示： 小心被称为:guilabel:'No reload'的属性：如果你修改文件，它会阻止服务器被实时重新加载。这允许保留现有的调试器断点，而它们通常会在重新加载服务器时被丢弃。

3.3 Weblate 源代码

Weblate 在 GitHub 上开发。欢迎您将代码复刻并开一个拉取请求。同样欢迎任何形式的补丁。

参见：

查阅 *Weblate 内部* 来看看 Weblate 内部是什么样子的。

3.3.1 代码编写准则

在为 Weblate 编写任何代码时都应该考虑到 通过设计保证安全原则。

任何代码都应附带解释行为的文档。不要忘记记录方法、复杂的代码块或用户可见的功能。

任何新代码都应使用 **PEP 484** 类型提示。我们尚未在 CI 中检查此项，因为现有代码尚未包含它们。

3.3.2 编码标准和代码检查

代码应该遵循 PEP-8 代码编写准则，并且应该使用 **black** 代码格式化程序进行代码格式化。

为了检查代码质量，可以使用 **flake8**，推荐的插件列在 `.pre-commit-config.yaml` 中，而其配置放置在 `setup.cfg` 中。

将所有这些强制的最简单的方法是安装 pre-commit。仓库为此包含了配置，来确定提交的文件是正常的。安装后（他已经包括在 `requirements-lint.txt` 中了）通过在 Weblate 的付款台运行 `pre-commit install` 来将它打开。通过这种方法，所有更改都将被自动检查。

还能够手动触发检查，来检查所有文件的运行：

```
pre-commit run --all
```

3.4 调试 Weblate

程序缺陷可以表现为应用崩溃或各种不正常行为。欢迎您搜集任何这类问题的信息，并将其提交给 问题跟踪器 。

3.4.1 调试模式

打开调试模式将在网络浏览器中显示异常。对于调试 web 界面中的问题很有用，但不适用于生产环境，因为它会拖累性能，并且可能泄漏私密数据。

在生产环境中，使用*ADMINS* 接收包含错误报告的电子邮件，或使用第三方服务配置错误收集。

参见：

禁止调试模式, 是当地配置管理设置, 收集错误报告

3.4.2 Weblate 日志

Weblate 可以生成后台正在发生什么的详细日志。在默认配置中，它使用 syslog，并使日志出现在 `/var/log/messages` 或 `/var/log/syslog` 中（取决于您的 syslog 守护程序配置）。

Celery 进程（请参见使用 *Celery* 的后台任务）通常也产生自己的日志。示例的系统范围的安装被记录到 `/var/log/celery/` 下的几个日志文件中。

Docker 容器记录它们的输出（按照 Docker 世界的惯例），因此可以使用 `docker-compose logs` 来查看日志。

参见：

配置的示例 包含 `LOGGING` 配置。

3.4.3 不处理后台任务

Celery workers 在后台完成很多事。如果像发送电子邮件或删除部件这样的事情不能正常运作，那么可能有一个相关的问题。

在那种情况下需要检查的事情：

- 检查 Celery 进程是否正在运行，见*使用 Celery 的后台任务*
- 在*管理界面* 中或者使用*celery_queues* 检查 Celery 队列状态
- 查看 Celery 日志寻找错误（请参见*Weblate 日志*）

3.4.4 不接收来自 Weblate 的电子邮件

你可以使用 `sendtestemail` 管理命令（关于在不同环境中如何调用它的指示说明请参见*调用管理命令*）或 *Tools* 选项卡下的*管理界面* 来验证外发电子邮件是否正常工作。

这些直接发送电子邮件，因此这验证您的 SMTP 配置是否正确（请参见*配置电子邮件发件箱*）。然而来自 Weblate 的多数电子邮件在后台发送，并且可能也会有 Celery 相关的问题，请参阅*不处理后台任务* 来调试。

3.4.5 分析应用的崩溃

在应用程序崩溃的情况下，尽可能多地收集有关崩溃的信息是很有用的。这可以通过使用第三方服务来实现，第三方服务可以自动收集这些信息。可以在*收集错误报告* 找到如何设置的信息。

3.4.6 无报告的故障

很多任务写在到 Celery 进行后台处理。故障不显示在用户界面上，但出现在 Celery 的日志中。配置*收集错误报告* 会帮助您更容易地注意到这样的故障。

3.4.7 性能问题

如果 Weblate 在某些情况下表现不佳，请收集显示问题的相关日志，以及任何可能有助于确定代码可能改进的地方的日志。

如果有些请求在没有任何提示的情况下花费了很长时间，你可能想要安装 dogslow，附加参数*收集错误报告* 并在错误收集工具中获取精确和详细的回溯信息。

如果性能缓慢与数据库有关，您还可以在启用*DEBUG* 后使用以下配置启用所有数据库查询的日志记录：

```
LOGGING["loggers"]["django.db.backends"] = {"handlers": ["console"], "level":
↪"DEBUG"}
```

3.5 Weblate 内部

备注： 这一章将给出 Weblate 内部的基本概况。

Weblate 从 Django 得到其多数代码架构，并基于它。

3.5.1 目录结构

Weblate 主仓库目录结构的速览：

docs
本文档的源码，可使用 Sphinx 来构建。

dev-docker
运行开发服务器的 Docker 代码，请参见在 *Docker* 中本地运行 *Weblate*。

weblate
Weblate Django 应用的的源代码作为，请参见*Weblate* 内部。

weblate/static
客户端文件（CSS、Javascript 和图片），请参见*Weblate* 前端。

3.5.2 模块

Weblate 包括几个 Django 应用（一些是可选的，请参见可选的 *Weblate* 模块）：

`accounts`
用户账户、简介和通知。

`addons`
微调 Weblate 行为的附加组件，请参见:ref:‘addons。

`api`
基于 Django REST 框架 的 API。

`auth`
认证和权限。

`billing`
可选的账单 模块。

`checks`
翻译字符串质量检查 模块。

`fonts`
字体渲染检查模块。

`formats`
基于 translate-toolkit 的文件格式抽象层。

`gitexport`
可选的*Git* 导出器 模块。

`lang`
定义语言和复数模型的模块。

`legal`
可选的法律声明 模块。

`machinery`
机器翻译服务的集成。

`memory`
内置的翻译记忆库，请参见翻译记忆库。

`screenshots`

屏幕截图管理与 OCR 模块。

`trans`

处理翻译的主模块。

`utils`

各种帮助功能。

`vcs`

版本控制系统抽象概念。

`wladmin`

Django 管理界面定制化。

3.6 开发附加组件

附加组件 是在 Weblate 中自定义本地化工作流的方法。

class weblate.addons.base.**BaseAddon**(*storage=None*)

Weblate 附加组件的基础类。

classmethod can_install(*component*, *user*)

检查附加组件是否与给定部件兼容。

configure(*settings*)

保存配置。

daily(*component*)

每日触发钩子。

classmethod get_add_form(*user*, *component*, ***kwargs*)

返回用于添加新附加组件的配置表单。

get_settings_form(*user*, ***kwargs*)

返回此附加组件的配置表单。

post_add(*translation*)

添加新的翻译后触发钩子。

post_commit(*component*)

更改提交到仓库后触发钩子。

post_push(*component*)

仓库推送到上游之后触发钩子。

post_update(*component*, *previous_head: str*, *skip_push: bool*)

在仓库从上游更新之后触发钩子。

参数

- **previous_head**(*str*) –仓库更新前的 HEAD，在初始克隆时可以是空白的。
- **skip_push**(*bool*) –附加组件操作是否应该跳过将更改推送到上游。通常可以将这个作为 “commit_and_push“或 `commit_pending` 传递到底层方法。

pre_commit(*translation*, *author*)

更改被提交到仓库前触发钩子。

pre_push(*component*)
在仓库推送上游之前触发钩子。

pre_update(*component*)
在仓库从上游更新之前触发钩子。

save_state()
保存附加组件状态信息。

store_post_load(*translation*, *store*)
文件解析后触发钩子。

它接收文件格式类的事件作为参数。

这对修复该文件格式类参数有用，例如，调整文件如何存储。

unit_pre_create(*unit*)
创建新的单元前触发的钩子。

这里是一个附加组件的示例：

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

from django.utils.translation import gettext_lazy as _

from weblate.addons.base import BaseAddon
from weblate.addons.events import EVENT_PRE_COMMIT


class ExampleAddon(BaseAddon):
    # Filter for compatible components, every key is
    # matched against property of component
    compat = {"file_format": {"po", "po-mono"}}
    # List of events add-on should receive
    events = (EVENT_PRE_COMMIT,)
    # Add-on unique identifier
    name = "weblate.example.example"
    # Verbose name shown in the user interface
    verbose = _("Example add-on")
    # Detailed add-on description
    description = _("This add-on does nothing it is just an example.")

    # Callback to implement custom behavior
    def pre_commit(self, translation, author):
        return
```

3.7 Weblate 前端

前端目前是使用 Bootstrap、jQuery 和少量第三方库构建的。

3.7.1 支持的浏览器

Weblate 支持所有主流浏览器和平台最新的稳定版。

不明确支持使用最新版 WebKit、Blink 或 Gecko 内核的替代浏览器，无论是直接使用还是通过平台的 web view API。但是，Weblate 应该（在大多数情况下）能在这些浏览器中正常显示和运行。

其它浏览器也能工作，但一些特性会受到限制。

3.7.2 依赖项管理

yarn 软件包管理工具用于更新第三方库。配置在 `scripts/yarn` 中，并且有个打包脚本 `scripts/yarn-update` 来更新库、构建它们，并复制到 `weblate/static/vendor` 中的正确位置上，所有的第三方前端代码都位于此目录。Weblate 特定的代码应该直接放在 `weblate/static` 或特性相关的子目录中（例如 `weblate/static/editor`）。

添加了新的第三方库，典型包括：

```
# Add a yarn package
yarn --cwd scripts/yarn add PACKAGE
# Edit the script to copy package to the static folder
edit scripts/yarn-update
# Run the update script
./scripts/yarn-update
# Add files to git
git add .
```

3.7.3 代码风格

Weblate 依赖于 Prettier 来进行 JavaScript 和 CSS 文件的代码格式化。

我们还使用 ESLint 来检查 JavaScript 代码。

3.7.4 本地化

如果在前端代码中需要任何用户可见的文本，那么应该将其本地化。在多数情况下，所有需要的是将文本打包到 `gettext` 函数内部，但也有更复杂的特性来使用：

```
document.write(gettext('this is to be translated'));

var object_count = 1 // or 0, or 2, or 3, ...
s = ngettext('literal for the singular case',
        'literal for the plural case', object_count);

fmts = ngettext('There is %s object. Remaining: %s',
        'There are %s objects. Remaining: %s', 11);
s = interpolate(fmts, [11, 20]);
// s is 'There are 11 objects. Remaining: 20'
```

参见：

Django 文档中的翻译主题

3.7.5 图标

Weblate 目前使用 material design 图标。如果你正在找新符号，请查看 Material Design Icons 或 Material Design Resources。

此外，有 `scripts/optimize-svg` 来减小 SVG 的大小，因为多数图标嵌入在 HTML 中，而使路径有风格。

3.8 在 Weblate 中汇报问题

Weblate 问题跟踪器 托管于 Github。

欢迎报告任何问题，或提出改进建议。我们准备了各种模板，可以轻松地指导你完成问题报告。

如果你在 Weblate 中发现了安全问题，请查阅下方的安全问题 部分。

如果您不确定您的缺陷报告或功能请求，您可以尝试*Weblate 讨论*。

3.8.1 安全问题

为了给予社区实践来响应并升级，强烈敦促您私下汇报所有的安全问题。HackerOne 用于处理安全问题，并且而可以在 HackerOne 直接汇报。一旦你在那里提交它，社区有有限但足够的时间来解决事件。

另外，可以汇报给 security@weblate.org，最后也会到 HackerOne 上。

如果你因为任何原因不想使用 HackerOne，你可以通过电子邮件将报告发送给 michal@weblate.org。你可以选择使用这个 PGP 密钥 *3CB 1DF1 EF12 CF2A C0EE 5A32 9C27 B313 42B7 511D* 对其加密。你还可以从 Keybase 获取 PGP 密钥。

备注： Weblate 在很多事情上依赖于第三方部件。如果你发现一个影响这些部件的漏洞，请直接报告给相应的项目。

这些中的一些是：

- Django
- Django REST 框架
- Python 社交认证

3.9 Weblate 测试套件与持续集成

测试套件存在于多数当前代码，通过为任何新的功能添加测试情况来扩大覆盖范围，并确认其正常工作。

3.9.1 持续集成

当前的测试结果可以在 GitHub Actions 上找到，覆盖范围在 Codecov 上进行报告。

有几项工作来确认不同的方面：

- 单元测试
- 文档构建与外部链接
- 来自所有支持的发布版本的合并测试
- 代码整理
- 设置确认（确保生成的 dist 文件不丢失任何内容，并可以测试）

CI 的配置在 `.github/workflows` 目录中。它重度使用了 `ci` 目录中的帮助脚本。脚本还可以手动执行，但它们需要一些环境变量，多数用来定义 Django 设置要使用的文件和数据库连接。它的示例定义在 `scripts/test-database.sh` 中：

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

# Simple way to configure test database from environment

# shellcheck shell=sh

# Database backend to use postgresql / mysql / mariadb
export CI_DATABASE="${1:-postgresql}"

# Database server configuration
export CI_DB_USER=weblate
export CI_DB_PASSWORD=weblate
export CI_DB_HOST=127.0.0.1

# Django settings module to use
export DJANGO_SETTINGS_MODULE=weblate.settings_test
```

简单的执行看起来可以像这样：

```
. scripts/test-database.sh
./ci/run-migrate
./ci/run-test
./ci/run-docs
```

3.9.2 本地测试

如果本地运行测试套件，要使用：

```
DJANGO_SETTINGS_MODULE=weblate.settings_test ./manage.py test
```

提示： 您会需要使用数据库（PostgreSQL）服务器来检测。Django 默认新建另外的数据库，以 `test_` 前缀来运行测试，因此在您的设置配置使用 `weblate` 的情况下，测试会使用 `test_weblate` 数据库。设置的指示请参见*Weblate 的数据库设置*。

`weblate/settings_test.py` 也用在 CI 环境中（请参见持续集成），并且可以使用环境变量调整：

```
# Copyright © Michal Čihař <michal@weblate.org>
#
# SPDX-License-Identifier: GPL-3.0-or-later

# Simple way to configure test database from environment

# shellcheck shell=sh

# Database backend to use postgresql / mysql / mariadb
export CI_DATABASE="${1:-postgresql}"

# Database server configuration
export CI_DB_USER=weblate
export CI_DB_PASSWORD=weblate
export CI_DB_HOST=127.0.0.1
```

(续下页)

(接上页)

```
# Django settings module to use
export DJANGO_SETTINGS_MODULE=weblate.settings_test
```

运行测试前，应该收集静态文件，因为一些测试在出现时依赖于它们：

```
DJANGO_SETTINGS_MODULE=weblate.settings_test ./manage.py collectstatic
```

您也可以指定要运行的各自测试：

```
DJANGO_SETTINGS_MODULE=weblate.settings_test ./manage.py test weblate.gitexport
```

提示： 测试也可以在开发者 docker 容器内执行，请参见*在 Docker 中本地运行 Weblate*。

参见：

运行并为 Django 写测试的更多信息请参见 Django 中的测试。

3.10 数据架构

Weblate 使用 JSON Schema 来定义外部 JSON 文件的输入。

3.10.1 Weblate 翻译记忆库概要

https://weblate.org/schemas/weblate-memory.schema.json			
类型	数组		
项目	翻译记忆项		
	类型	对象	
	属性		
	• **category**	字符串类别	
		1 是全局的，2 是共享的，10000000+ 是项目特定的，20000000+ 使用户特定的	
		类型	整数
		示例	1
		最小	0
		默认	1
	• **origin**	字符串来源	
		文件名或部件名	
		类型	字符串
		示例	test.tmx
			项目/部件
		默认	
	• **source**	源字符串	
		类型	字符串
		示例	你好
		最小长度	1
		默认	
	• **source_language**	源语言	
		ISO 639-1 / ISO 639-2 / IETF BCP 47	
		类型	字符串
		示例	英语
		模式	^[^]+$

续下页

表 1 – 接上页

		默认	
	• **target**	目标字符串	
		类型	字符串
		示例	Ahoj
		最小长度	1
		默认	
	• **target_language**	目标语言	
		ISO 639-1 / ISO 639-2 / IETF BCP 47	
		类型	字符串
		示例	cs
		模式	^[^]+$
		默认	
	additionalProperties	假	
定义			

参见:

翻译记忆库, *dump_memory*, *import_memory*

3.10.2 Weblate 用户数据导出

https://weblate.org/schemas/weblate-userdata.schema.json			
类型	对象		
属性			
• **basic**	基本		
	类型	对象	
	属性		
	• **username**	用户名	
		类型	字符串
		示例	管理员
		默认	
	• **full_name**	全名	
		类型	字符串
		示例	Weblate 管理员
		默认	
	• **email**	电子邮件	
		类型	字符串
		示例	noreply@example.com
		默认	
		格式	电子邮箱
	• **date_joined**	加入日期	
		类型	字符串
		示例	2019-11-18T18:53:54.862Z
		默认	
		格式	日期-时间
	additionalProperties	假	
• **profile**	个人资料		
	类型	对象	
	属性		
	• **language**	语言	
		类型	字符串
		示例	cs
		模式	^[^]*$

续下页

表 2 – 接上页

	默认		
• **suggested**	建议的字符串数量		
	类型	整数	
	示例	1	
	默认	0	
• **translated**	已翻译字符串的数量		
	类型	整数	
	示例	24	
	默认	0	
• **uploaded**	已上传的截屏数量		
	类型	整数	
	示例	1	
	默认	0	
• **hide_complete**	在操作面板上隐藏已完成的翻译		
	类型	布尔值	
	示例	假	
	默认	真	
• **secondary_in_zen**	在禅模式下显示第二语言翻译		
	类型	布尔值	
	示例	真	
	默认	真	
• **hide_source_s**	如果有第二语言翻译则隐藏原文		
	类型	布尔值	
	示例	假	
	默认	真	
• **editor_link**	编辑器链接		
	类型	字符串	
	示例		
	模式	^.*$	
	默认		
• **translate_mode**	翻译编辑器模式		
	类型	整数	
	示例	0	
	默认	0	
• **zen_mode**	禅编辑器模式		
	类型	整数	
	示例	0	
	默认	0	
• **special_chars**	特殊字符		
	类型	字符串	
	示例		
	模式	^.*$	
	默认		
• **dashboard_view**	操作面板默认视图		
	类型	整数	
	示例	1	
	默认	0	
• **dashboard_compon**	默认部件列表		
	默认	空	
	anyOf	类型	空
		类型	整数
• **languages**	翻译语言		
	类型	数组	
	默认		
	项目	语言代码	
		类型	字符串

续下页

表 2 – 接上页

			示例	cs
			模式	^.*$
			默认	
		• **secondary_langu**	第二语言	
			类型	数组
			默认	
			项目	语言代码
			类型	字符串
			示例	sk
			模式	^.*$
			默认	
		• **watched**	已关注项目	
			类型	数组
			默认	
			项目	项目标识串
			类型	字符串
			示例	weblate
			模式	^.*$
			默认	
		additionalProperties	假	
• **auditlog**				
	审计日志			
	类型	数组		
	默认			
	项目	项目		
		类型	对象	
		属性		
			• **address**	*IP* 地址
				类型 字符串
				示例 127.0.0.1
				模式 ^.*$
				默认
			• **user_agent**	用户代理
				类型 字符串
				示例 PC / Linux / Firefox 70.0
				模式 ^.*$
				默认
			• **timestamp**	时间戳
				类型 字符串
				示例 2019-11-18T18:58:30.845Z
				默认
				格式 日期-时间
			• **activity**	活动
				类型 字符串
				示例 登录
				模式 ^.*$
				默认
		additionalProperties	假	
定义				

参见:

用户个人资料, *dumpuserdata*

3.11 发布 Weblate

3.11.1 发布日程

Weblate 每两个月更新一次大版本 (x.y)。之后通常会有数个小版本 (x.y.z) 来修复大版本中出现的问题。

主要版本的更改指示了升级过程不能跳过这个版本——在升级到更高版本的版本 x.y 之前总是必须升级到版本 x.0。

参见:

升级 Weblate

3.11.2 发布计划

即将发布版本的功能是通过 GitHub 里程碑收集的，可以在 <https://github.com/WeblateOrg/weblate/milestones> 上查看路线图。

3.11.3 发布过程

发布前要检查的事情：

1. 由 **./scripts/list-translated-languages** 来检查新翻译的语言。
2. 由 **./scripts/prepare-release** 来设置最终版本。
3. 确保截图是最新的 **make -j 12 -C docs update-screenshots**。
4. 合并任何可能挂起的翻译 **wlc push; git remote update; git merge origin/weblate**

执行发布：

5. 创建一个发布 **./scripts/create-release --tag**（要求请参见下面）。

手动发布步骤：

6. 更新 Docker 镜像。
7. 关闭 GitHub 里程碑。
8. 一旦检测到 Docker 镜像，则添加标签并推送。
9. 将 Helm 图表更新到新版本。
10. 在 `.github/workflows/migrations.yml` 中包含新版本，以便在迁移测试中涵盖它。
11. 在网站下载链接中增加版本。
12. 通过 **./scripts/set-version** 在仓库中增加版本。
13. 检查 readthedocs.org 是否使用 **./scripts/rtd-projects**。构建了文档的所有翻译。

要使用 **./scripts/create-release** 脚本创建标签，需要：

- 带有私钥的 GnuPG，用于为发行版签名
- 推送访问 Weblate git 仓库（它推送标签）
- 配置的 **hub** 工具和访问，在 Weblate repo 上创建发布版本
- SSH 访问 Weblate 下载服务器（网站下载内容被复制到的地方）

3.12 安全和隐私

小技巧： 在 Weblate，安全维持着一个重视用户隐私的环境。

Weblate 的开发符合 Linux 基金会发起的核心基础设施计划最佳实践。

参见：

安全问题

3.12.1 安全更新

只保证最新版能接收安全更新。

3.12.2 跟踪漏洞的依赖关系

我们使用 Dependabot 来监控我们依赖关系中的安全问题。这涵盖了 Python 和 JavaScript 库，最新的稳定版已经更新了其依赖关系，以避免漏洞。

提示： 第三方库中可能存在不影响 Weblate 的漏洞，所以这些漏洞不会通过发布 Weblate 的错误修复版本来解决。

3.12.3 Docker 容器安全

使用 Anchore 和 Trivy 安全扫描器对 Docker 容器进行定期扫描。

这使我们能够及早发现漏洞并迅速发布改进措施。

你可以在 GitHub 上获得这些扫描的结果–它们以 SARIF 格式（静态分析结果交换格式）作为工件存储在我们的 CI 上。

参见：

持续集成

3.13 为 Weblate 模块作贡献

除了主仓库之外 Weblate 还包含几个 Python 模块，所有这些都遵循相同的结构本文档涵盖了所有这些。

例如，这涵盖了：

- wlc,Python 客户端库, 请参阅*Weblate 客户端*
- translation-finder 用于发现仓库中的可翻译文件
- language-data，Weblate 的语言定义，请参见语言定义

3.13.1 代码编写准则

在为 Weblate 编写任何代码时都应该考虑到 通过设计保证安全原则。

任何代码都应附带解释行为的文档。不要忘记记录方法、复杂的代码块或用户可见的功能。

任何新代码都应使用 **PEP 484** 类型提示。我们尚未在 CI 中检查此项，因为现有代码尚未包含它们。

3.13.2 运行测试

测试使用:program:‘py.test‘执行, 首先你需要安装测试需求：

```
pip install -r requirements-test.txt
```

然后您可以在仓库签出中执行测试套件:

```
py.test
```

参见:

CI 集成与*Weblate 测试套件与持续集成* 非常相似。

3.13.3 编码标准和代码检查

代码应该遵循 PEP-8 代码编写准则，并且应该使用 **black** 代码格式化程序进行代码格式化。

为了检查代码质量，可以使用 **flake8**，推荐的插件列在 `.pre-commit-config.yaml` 中，而其配置放置在 `setup.cfg` 中。

将所有这些强制的最简单的方法是安装 pre-commit。仓库为此包含了配置，来确定提交的文件是正常的。安装后（他已经包括在 `requirements-lint.txt` 中了）通过在 Weblate 的付款台运行 `pre-commit install` 来将它打开。通过这种方法，所有更改都将被自动检查。

还能够手动触发检查，来检查所有文件的运行：

```
pre-commit run --all
```

参见:

Weblate 源代码

3.14 关于 Weblate

3.14.1 项目目标

基于 Web、与版本控制紧密集成 的持续本地化工具，支持多种文件格式，让译者可以轻松地作贡献。

3.14.2 项目名称

“Weblate”由单词“web”（网络）和“translate”（翻译）融合而来。

3.14.3 项目网站

着陆页是 https://weblate.org，云托管服务在 https://hosted.weblate.org。文档在 https://docs.weblate.org。

3.14.4 项目标志

项目标志和其它图片可以在 https://github.com/WeblateOrg/graphics 中找到。

3.14.5 领导

本项目由 Michal Čihař 维护，联系方式是 michal@weblate.org。

3.14.6 作者

Weblate 由 Michal Čihař 创立。自 2012 年成立以来，已有数千人做出了贡献。

3.15 许可协议

更详细的许可信息可以在 Weblate 源码中找到并遵守 REUSE 3.0 规范.

CHAPTER 4

更新历史记录

4.1 Weblate 4.16.2

发布于 2023 年 3 月 8 日。

- 修复了翻译记忆库中搜索的问题。
- 修复了使用更多服务进行自动化翻译的问题。
- 改进了重叠的术语表术语匹配的解析。
- 修复了某些格式下非英语源语言的复数形式解析。
- 新增支持 go-i18n v2 JSON 文件。

所有变化详情。

4.2 Weblate 4.16.1

发布于 2023 年 3 月 1 日。

- 修复了测试套件错误。

所有变化详情。

4.3 Weblate 4.16

发布于 2023 年 3 月 1 日。

- 格式字符串检查现在同样检查重复和格式。
- 改进了搜索某些特殊格式字符串的性能。
- Celery beat 现在在数据库中保存任务计划。
- 新增支持 IBM Watson Language Translator。
- 去掉了对在 4.14 版本中废弃的 VCS 集成设置的支持。
- 新增支持 Bitbucker 服务器拉取请求。

- 改进了 gettext PO 文件中的冲突处理。
- 新增支持通过 API 添加字符串时定义字符串的状态。
- 新增支持配置允许的 CORS 源。
- 自动建议新增复数形式支持。

所有变化详情。

4.4 Weblate 4.15.2

发布于 2023 年 1 月 25 日。

- 在默认配置中启用了 gotext JSON 和 i18next v4 版格式。
- 修复了上传破损文件时的崩溃。
- 在 Git 仓库状态中显示 stale 目录。

所有变化详情。

4.5 Weblate 4.15.1

发布于 2023 年 1 月 19 日。

- 修复了来自自动翻译的建议。
- 修复了某些极端情况下附加组件页面崩溃的问题。
- 修复了某些情况下取消翻译新翻译的模板的问题。
- 文档许可使用 REUSE 3.0.
- 修复了团队管理上的用户分页。
- 改进了项目创建和保存的性能。
- 新增支持 gotext JSON 文件。
- 新增支持 i18next v4 文件。
- API 中的分页现在可定制。

所有变化详情。

4.6 Weblate 4.15

发布于 2022 年 12 月 16 日。

- 新增支持浏览单条字符串的更改。
- 修复了来自其他部件自动翻译复数处理的问题。
- 添加了将字符串提交为建议的键盘快捷键 Alt+Enter。
- 新增支持 Fluent 格式的可放置对象。
- 改进了翻译记忆库的性能。
- 为知名的代码托管服务自动生成 repoweb 浏览链接。
- 改进了多个视图的性能。
- 改进了含复数的字符串的列表。

- 新增支持将自定义标记添加到 HTML 头。
- 修复了附加组件中 MO 文件生成的问题，现在只包含已翻译文件。
- 修复了正则表达式标记渲染的问题。
- 改进了复数的占位符检查行为。
- 新增支持适合 Google Play 商店的翻译文件命名。
- 新增支持 API 中的标签。
- 新增支持为译文提交和通知选择不同的电子邮箱。
- 该 Docker 镜像不再默认启用调试模式。
- 根据术语表部件优先级对术语表术语进行排序。
- 添加了可以添加或删除团队成员的管理员。
- 增加了删除用户前的弹窗确认。
- 添加了自定义 XML 输出的附加组件。

所有变化详情。

4.7 Weblate 4.14.2

发布于 2022 年 11 月 5 日。

- 新增支持从翻译记忆库中删除条目。
- 改善了对重复语言警告的分析。
- 提高了连续重复的单词检查的准确性。
- 改进了发送许多通知的缩放。
- 改进了字幕翻译的字符串状态处理。
- 已弃用通过 _TOKEN/_USERNAME 配置而不是 _CREDENTIALS 列表对版本控制系统（VCS）服务 API 密钥进行不安全配置。
- 修复了某些上传的 CSV 文件的处理。
- 改进了差异显示中的空白更改处理。
- 管理页面添加了自动建议管理链接。
- 在历史记录中跟踪评论删除/解决。
- 修复了还原带链接部件的项目备份的问题。
- 修复了注册失败时的 captcha 验证码输入。
- 改进了 DeepL 中的语言支持。
- 改进了 webhook 与经身份验证的仓库的兼容性。
- 增加了对 Python 3.11 的支持。

所有变化详情。

4.8 Weblate 4.14.1

发布于 2022 年 9 月 15 日。

- 修复了在某些情况下生成项目备份的问题。
- 改进了文件上传时的错误报告。
- 在身份验证期间从 GitHub 获取所有用户验证的电子邮件。
- 避免在上下文或键上匹配术语表术语。
- 添加了字符串删除通知。
- 改进了术语表中不可翻译术语的管理。
- 在团队管理页面上列出团队成员的数量。
- 添加群组管理界面。
- 启用审校时始终显示审校统计数据。
- 单元 API 中新增搜索支持。
- 修复了审阅工作流程中只读字符串进度条显示的问题。
- 改进了缅甸文标点符号检查。
- 修复了指标数据的垃圾收集。

所有变化详情。

4.9 Weblate 4.14

发布于 2022 年 8 月 22 日。

- 跟踪历史记录中的附加组件更改。
- 修复了从 Windows RC、HTML 和文本文件解析翻译的问题。
- 扩展了语言代码样式配置选项。
- 添加了对最近 CLDR 发行版本中更新的复数的支持。
- 降低了更新包含大量翻译的组件时的内存用量。
- 新增支持 SAP 翻译中心翻译域。
- 允许源字符串位置中的绝对链接。
- 改进了通过一些反向代理进行的操作。
- 扩展 API 以涵盖翻译记忆库。
- 改进了文档翻译工作流程。
- 提高了 HTML 和文本文件翻译的可靠性。
- 新增支持项目级别备份。
- 改进了翻译记忆库查找的性能和内存使用情况。

所有变化详情。

4.10 Weblate 4.13.1

发布于 2022 年 7 月 1 日。

- 修复了历史记录中的跟踪建议。
- 修复了从 Cloudflare 解析反向代理信息的问题。
- 让解析错误锁定部件，使其无法翻译。
- 修复了在发现加载项中配置中间文件的问题。
- 修复了带有占位符的 DeepL 翻译行为。
- 通过 API 修复了未翻译的字符串。
- 添加了对通过 API 从群组中删除用户的支持。
- 修复了用户邀请电子邮件的审核日志。
- 修复了 Java 格式化字符串的标记名称的问题。

所有变化详情。

4.11 Weblate 4.13

发布于 2022 年 6 月 15 日。

- 改变了更新语言名称的行为。
- 为项目列表添加了分页。
- 用于创建新单元的 API 现在返回有关新创建单元的信息。
- 部件发现现在支持配置中间语言。
- 为 CSV 格式增加了修复编码变体。
- 改变了对某些格式的上下文和位置的处理以更好地适应底层实现。
- 新增支持 ResourceDictionary 格式。
- 为色盲用户改进了进度条颜色。
- 修复了字符串删除时变体处理的问题。
- 兼容 Django 4.1。
- 新增支持以 XLIFF 格式储存转义 XML 元素。
- 改进了占位符检查错误的格式化。
- 将 /.well-known/change-password 重定向到 /accounts/password/.
- 现在可以为每个项目配置不同的翻译服务了。
- 添加了用于解析评论的单独权限并将其授予 审校字符串角色。
- 添加了对在 CSV 文件中存储替代翻译的支持。
- 占位符检查现在也可以不区分大小写。

所有变化详情。

4.12 Weblate 4.12.2

发布于 2022 年 5 月 11 日。

- 修复了重建某些部件的项目翻译记忆库的问题。
- 修复了按未翻译字符串排序部件的问题。
- 修复了添加新语言时可能丢失翻译的问题。
- 确保在迁移过程中生成 Weblate SSH 密钥。

所有变化详情。

4.13 Weblate 4.12.1

发布于 2022 年 4 月 29 日。

- 修复了拉取请求说明标题的问题。
- 改进了 Fluent 格式的语法错误处理。
- 修复了通知电子邮件中的头像显示问题。
- 新增支持网络变现。
- 修复了删除翻译时删除陈旧源字符串的问题。

所有变化详情。

4.14 Weblate 4.12

发布于 2022 年 4 月 20 日。

- 不匹配的句号 新增了对阿姆哈拉语的支持。
- 不匹配的问号 新增了对缅甸语的支持。
- 扩展了假语言环境（*Pseudolocale*）生成 附加组件的选项。
- 添加 `ignore-all-checks` 标记以忽略对字符串的所有质量检查。
- 避免假语言环境（*Pseudolocale*）生成 附加组件触发未通过的检查。
- 新增了对*Gitea 拉取请求* 的支持。
- 在语言代码风格 中添加了 Linux 风格的语言代码。
- 新增支持重建项目翻译记忆库。
- 改进了从文件创建部件的 API。
- 将复制和克隆按钮添加到其他翻译。
- 使合并请求说明可在部件层次上进行配置。
- 改进了 XML 标签的最大长度限制行为。
- 修复了加载带有其他注释的 Fluent 文件的问题。

所有变化详情。

4.15 Weblate 4.11.2

发布于 2022 年 3 月 4 日。

- 修复了二进制发行版中损坏的 MO 文件。

所有变化详情。

4.16 Weblate 4.11.1

发布于 2022 年 3 月 4 日。

- 修复了遗漏的 Git 和 Mercurial 参数清理 - CVE-2022-23915，请参阅 GHSA-3872-f48p-pxqj 了解更多信息。
- 修复了从 CSV 文件加载模糊字符串的问题。
- 新增支持使用 API 创建团队。
- 修复了显示用户提及建议的问题。
- 现在可以自定义项目令牌访问了。

所有变化详情。

4.17 Weblate 4.11

发布于 2022 年 2 月 25 日。

- 修复了存储型 XSS - CVE-2022-24710，请参阅 GHSA-6jp6-9rf9-gc66 了解更多信息。
- 修复了使用 API 安装附加组件的问题。
- 将 需要处理的字符串（*Strings needing action*）改名为 未完成的字符串（*Unfinished strings*）。
- 修复了来自*ICU MessageFormat 语法* 的误报。
- 在其它的出现位置清单上指示锁定状态和贡献者协议。
- 修复了更新带有过时字符串或缺少复数形式的 PO 文件。
- 改进了挤压附加组件与 Gerrit 的兼容性。
- 根据 Accept-Language 标头自动选择用户语言。
- 改进了字符串删除的错误处理。
- Weblate 现在需要 Python 3.7 或更新版本。
- 修复了一些项目令牌验证的写入操作。
- 修复了仓库中的字符串更改时的状态跟踪。
- 跟踪来自仓库的字符串更改。
- 翻译列表上的粘性标头以改善导航。
- 修复了不翻译*Java 属性* 中的字符串的问题。
- 修复了具有非 ASCII 分支名称的 Git 操作。
- 新附加组件用原文预填充译文。
- 添加了 不快进合并 ‘的*:ref:‘component-merge_style*。
- 修复了自动翻译 附加组件在新添加的字符串触发的问题。
- 改进了缅甸文的标点检查。

- 添加了对在项目级别定义自定义团队以授予用户访问权限的支持，请参阅管理每个项目的访问控制。
- 添加了警报的文档链接。
- Docker 容器会在需要时为发送的电子邮件自动件启用 TLS/SSL。
- 支持了搜索已解决的评论。
- 新增了对 borgbackup 1.2 的支持。
- 修复了 自动翻译标签的应用.

所有变化详情。

4.18 Weblate 4.10.1

发布于 2021 年 12 月 22 日。

- 升级到 Django 4.0 引入的文档化更改。
- 修复了 自动翻译标签的显示。
- 修复了在有共享仓库的部件中分支的 API 显示。
- 改善了对失败的推送警报的分析。
- 修复了浏览更改时手动编辑页面的问题。
- 改进了使用了 *Kashida 字母* 的准确性。
- Weblate Docker 容器现在使用 Python 3.10。

所有变化详情。

4.19 Weblate 4.10

发布于 2021 年 12 月 16 日。

- 新增对 DeepL Formality 参数和占位符的支持。
- 批量编辑、搜索和替换现在可以在项目和语言层面上使用。
- 为搜索和替换功能加入了筛选。
- 已修复：“执行自动翻译”权限不再属于 *Languages* 组。
- “执行自动翻译”位于 * 管理 * 和新的 * 自动翻译 * 组中。
- 修复了生成带特殊字符的 XLSX 文件的问题。
- 为 GitHub 认证后端增加了检查用户是否属于特定 GitHub 组织或团队的功能。
- 改进了对传递给 API 的无效参数的反馈。
- 新增对 API 项目范围访问令牌的支持。
- 修复了一些情况下的字符串移除问题。
- 修复了翻译新添加字符串的问题。
- 给自动翻译的字符串添加标签，以简化其筛选。

所有变化详情。

4.20 Weblate 4.9.1

发布于 2021 年 11 月 19 日。

- 修复了更改模板后单语言文件的上传问题。
- 改进了对标记中空格的处理。
- 下载 API 新增对过滤的支持。
- 修复了添加新翻译时的统计数据显示。
- 缓解 GitHub SSH 密钥更改的问题。

所有变化详情。

4.21 Weblate 4.9

发布于 2021 年 11 月 10 日。

- 提供历史记录中事件的更多详情。
- 改进了历史的呈现。
- 改进了翻译页面的性能。
- 新增支持限制翻译文件下载。
- `safe-html` 现在与 `md-text` 一起使用时能理解 Markdown。
- `max-length` 标签现在与 `xml-text` 一起使用时，会忽略 XML 标记。
- 修复了:ref:‘check-max-size‘中渲染文本的尺寸。
- 将应用商店的标题长度降至 30 以协助应对即将到来的谷歌政策变化。
- 新增支持通过*SSH_EXTRA_ARGS* 自定义 SSH 调用。
- 新增了对 ICU MessageFormat 的检查。
- 改进了机器翻译后端中的错误状况处理。
- 突出显示字符串中不常见的空格。
- 新增编辑时停留在已翻译字符串的选项。
- 新增支持通过*BORG_EXTRA_ARGS* 定制 Borg 调用。
- 修复了为单语言翻译生成 MO 文件。
- 新增 API 端点，支持以 ZIP 文件打包下载所有部件翻译。
- 增加了对 Python 3.10 的支持。
- 新增支持从管理界面重新发送电子邮件邀请。

所有变化详情。

4.22 Weblate 4.8.1

发布于 2021 年 9 月 10 日。

- 修复了 Django 管理界面中的用户删除。
- 在文档中更详细地说明附加组件的参数。
- 修复了术语表的 JavaScript 错误。
- 在一致性检查中增加匹配数量的限制。
- 改进机器翻译中的占位符处理。
- 修复了使用 API 创建附加组件的问题。
- 添加:setting:‘PRIVACY_URL‘设置以将隐私策略链接添加到页脚。
- 对项目管理员隐藏成员电子邮箱地址。
- 改进了 gettext PO 在冲突情况下的合并。
- 改进了术语表高亮。
- 通过 XML 检查改进了 “safe-html“标记行为。
- 修复了链接部件的提交说明。

所有变化详情。

4.23 Weblate 4.8

发布于 2021 年 8 月 21 日。

- 新增对 Apple stringsdict 格式的支持。
- 精确搜索运算符现在对 PostgreSQL 区分大小写。
- 修复了某些情况下保存术语表解释的问题。
- 文档改进。
- 性能改进。
- 改进了挤压附加组件与 Gerrit 的兼容性。
- 修复了向单语术语表部件添加字符串的问题。
- 改进了处理变体的性能。
- 修复了 squash 附加组件有时会跳过解析上游更改。
- 保留下载文件扩展名。
- 增加了对 Fluent 格式的支持。
- 支持使用制表符缩进 JSON 格式。

所有变化详情。

4.24 Weblate 4.7.2

发布于 2021 年 7 月 15 日。

- 支持在项目上配置更多的语言别名。
- 修复了 API 中搜索字符串验证问题。
- 修复了域名变更后的 Git 导出器 URL 问题。
- 修复了 Windows RC 文件的清理附加组件。
- 修复了更新 XLIFF 时可能的崩溃。

所有变化详情。

4.25 Weblate 4.7.1

发布于 2017 年 7 月 30 日。

- 优化了向术语表中添加术语的弹窗。
- 新增对 LibreTranslate 机器翻译服务的支持。
- 增加了创建新项目的速率限制。
- 改进了文件更新的性能。

所有变化详情。

4.26 Weblate 4.7

发布于 2021 年 6 月 17 日。

- 改进了配置健康性的检查。
- 添加了对 gettext PO 中所用的 `object-pascal-format` 的支持，见*Object Pascal 格式*。
- 已将 *Nearby keys* 重命名为:guilabel:‘Similar keys 以更好地描述目的。
- 新增了对*mi18n lang 文件* 的支持。
- 改进了 SAML 验证集成。
- 修复了*Gerrit* 集成问题，可更好地处理极端情况。
- Weblate 现在需要 Django 3.2。
- 修复了电子邮件身份验证被禁用时邀请用户的问题。
- 改进了语言定义。
- 增加了阻止用户参与项目的功能。
- 修复了术语表语言的自动创建。
- 扩展了关于附加组件的文档。
- 改善了带链接仓库的部件的性能。
- 增加了对免费 DeepL API 的支持。
- 用户管理不再需要 Django 管理界面。

所有变化详情。

4.27 Weblate 4.6.2

发布于 2021 年 5 月 8 日。

- 修复了在项目之间移动共享部件后的崩溃。
- 修复了将新字符串添加到空属性文件的问题。
- 修复了 RTL 语言中的副本图标对齐。
- 扩展了信息选项卡上的字符串统计信息。
- 修复了 Git 中忽略的翻译文件的处理。
- 改进了度量性能。
- 修复了保存术语表可能发生的错误。
- 修复了具有不同复数规则的语言的一致性检查行为。

所有变化详情。

4.28 Weblate 4.6.1

发布于 2021 年 5 月 2 日。

- 删除了过时的垃圾电子邮件防护代码。
- 提高了原文复数检查的准确性。
- 更新 Docker 中的用户界面语言列表。
- 改进了创建拉取请求的错误信息。
- 修复了在 Pagure 上创建拉取请求的问题。
- 修复了触发自动安装的附加组件的问题。
- 修复了升级时可能出现的缓存问题。
- 修复了使用上传将新单元添加到单语翻译中的问题。

所有变化详情。

4.29 Weblate 4.6

发布于 2020 年 4 月 19 日。

- auto_translate 管理命令现在有了用于指定翻译模式的参数。
- 新增了对文本文件 的支持。
- 增加了所有对象的趋势和指标。
- 支持了直接从第二语言复制文本。
- 新增浏览变化时日期过滤。
- 改进了活动图表。
- 现在可以配置联系人表单电子邮件的发件人。
- 改进了部件新建 API 中的参数验证。
- 速率限制不再适用于超级用户。
- 改进了自动翻译附加组件的性能和可靠性。

- 现在可以在 Docker 容器中自定义速率限制。
- 创建部件的 API 现在自动使用*Weblate 内部网址*。
- 简化了列出字符串时的状态提示。
- 密码散列现在默认使用 Argon2。
- 简化了显示翻译状态的进度条。
- 重命名了添加缺少的语言 以澄清目的。
- 修复可了保存字符串状态到 XLIFF 的问题。
- 添加了在全部语言范围内的搜索。
- 对横向扩展 Docker 部署的初步支持。

所有变化详情。

4.30 Weblate 4.5.3

发布于 2020 年 4 月 1 日。

- 修复了指标收集的问题。
- 修复了添加字符串时可能的崩溃。
- 改进的搜索查询示例。
- 修复了替换上传时可能会丢失新添加的字符串的问题。

4.31 Weblate 4.5.2

发布于 2021 年 3 月 26 日。

- 可配置的自动翻译时间表。
- 添加了 Lua 格式检查。
- 忽略:ref:‘check-duplicate‘检查中的格式字符串。
- 允许从翻译页面上传截图。
- 向仓库维护添加了强制文件同步。
- 修复了代码较长的语言的自动建议。
- 改进了添加新字符串时的性能。
- 修复了几处质量检查中的错误。
- 几处性能改进。
- 添加了与发现 *Weblate* 的集成。
- 修复了对只读字符串的检查行为。

所有变化详情。

4.32 Weblate 4.5.1

发布于 2021 年 3 月 5 日。

- 修复了在某些极端情况下编辑术语表标记的问题。
- 扩展指标使用以提高多个页面的性能。
- 以 TMX 文件存储正确的源语言。
- 使用 API 更好地处理单语言 PO 的上传。
- 改进了术语表部件的警报行为。
- 改进了 Markdown 链接检查。
- 在面包屑导航中指示术语表和源语言。
- 对大型项目中的部件清单进行分页。
- 改进翻译、部件或项目移除的性能。
- 改进了批量编辑的性能。
- 修复了在 ODF 文件保存“需要编辑”和“已核准”状态的问题。
- 改进了自定义翻译文件下载的界面

所有变化详情。

4.33 Weblate 4.5

发布于 2021 年 2 月 19 日。

- 添加了对 gettext PO 中所用的 `lua-format` 的支持。
- 新增支持在项目间共享一个部件。
- 修复了带有多个格式化标记的多个未命名变量检查的行为。
- 去掉了项目的电子邮件列表字段，为译者提供通用说明。
- 添加了用于生成假语言环境的附加组件。
- 添加了对 TermBase eXchange 文件的支持。
- 增加了对使用标记手动定义字符串变体的支持。
- 改进了一致性检查的性能。
- 改进了长字符串翻译记忆库的性能。
- 支持了在解释中搜索。
- 现在也可以以双语格式添加和删除字符串。
- 扩展亚马逊机器翻译支持的语言。
- 自动启用 Java 属性的 Java MessageFormat 检查。
- 增加了一种新的将新字符串添加到翻译的上传方法。
- 添加了浏览翻译的简单界面。
- 术语表现在存储为常规部件。
- 删除了术语表的特定 API，因为现在使用部件 API。
- 添加了用于切换某些标记的简化界面。
- 增加了对术语表中不可翻译或禁止的术语的支持。

- 支持在术语表中定义专业术语。
- 移动文本方向切换，为可视键盘获取更多空间。
- 添加了自动关注用户贡献的项目的选项。
- 添加了检查翻译是否与术语表匹配的功能。
- 添加了对自定义导航文本颜色的支持。

所有变化详情。

4.34 Weblate 4.4.2

发布于 2021 年 1 月 14 日。

- 修复了一个发布的 MO 文件的崩溃问题。

4.35 Weblate 4.4.1

发布于 2021 年 1 月 13 日。

- 修复了还原复数更改。
- 修复了展示项目设置帮助。
- 改进了用户管理。
- 改进了单语言 PO 文件中上下文的处理。
- 修复了 HTML、ODF、IDML 和 Windows RC 格式的清理附加组件的行为。
- 修复了从 CSV 文件解析位置的错误。
- 下载文件时使用内容压缩。
- 改进了从 ZIP 文件导入的用户体验。
- 改进了对上传文件格式的检测。
- 避免在 Pagure 上重复拉取请求。
- 改进了显示 ghost 翻译时的性能。
- 重新实现了翻译编辑器以使用浏览器原生文本区域。
- 修复了清理附加组件破坏添加新字符串功能的问题。
- 添加了附加组件 API。

所有变化详情。

4.36 Weblate 4.4

发布于 2020 年 12 月 15 日。

- 改进了新建部件时的验证。
- Weblate 现在需要 Django 3.1。
- 支持在管理界面中自定义外观。
- 修复了批量编辑时只读状态的处理。
- 改进了 CodeMirror 集成。

- 添加了从翻译文件中删除空白字符串的附加组件。
- CodeMirror 编辑器现在用于翻译。
- 为 XML、HTML、Markdown 和 reStructuredText 在翻译编辑器中语法高亮。
- 在翻译编辑器中突出显示可放置位置。
- 改进了对非标准语言代码的支持。
- 添加了使用歧义语言代码时的警报。
- 添加新的翻译时，用户会看到过滤后的语言列表。
- 扩展了更改历史的搜索能力。
- 改进了账单详情页面和自由软件项目托管流程。
- 扩展了翻译统计 API。
- 改进了翻译时“其他翻译”选项卡。
- 添加了任务 API。
- 改进了文件上传的性能。
- 改进了用户定义的特殊字符的显示。
- 改进了自动翻译的性能。
- 几处用户界面的小改进。
- 改进了 ZIP 文件下载的命名。
- 添加了获取未关注项目通知的选项。

  所有变化详情。

4.37 Weblate 4.3.2

发布于 2020 年 11 月 4 日。

- 修复了特定部件文件掩码的崩溃。
- 提高了连续重复的单词检查的准确性。
- 新增了对 Pagure 拉取请求的支持。
- 改进了注册失败的错误消息。
- 撤销了以 Markdown 格式渲染开发者注释。
- 简化了默认分支非“master”的 Git 仓库的安装设置。
- 新建的内部仓库现在使用主干作为默认分支。
- 降低了翻译重构文本时未更改译文的误报率。
- 修复了一些情况下的 Codemirror 显示问题。
- 将模板组重命名为“原文”，以澄清其含义。
- 修复了路径较长代码仓库的 GitLab 拉取请求。

所有变化详情。

4.38 Weblate 4.3.1

发布于 2020 年 10 月 21 日。

- 改进了自动翻译性能。
- 修复了身份已验证用户的会话过期问题。
- 新增了对隐藏版本信息的支持。
- 改进了钩子与 Bitbucket 服务器的兼容性。
- 改进了翻译记忆库更新的性能。
- 减少了内存的使用。
- 改进了矩阵视图的性能。
- 增加了将一名用户从一个项目移除前的确认功能。

  所有变化详情。

4.39 Weblate 4.3

发布于 2020 年 10 月 15 日。

- 包括了 API 中的用户统计数据。
- 修复了分页的页面上订购的部件。
- 定义了术语表的源语言。
- 重写了对 GitHub 和 GitLab 拉取请求的支持。
- 修复了移除建议后的统计数据计数。
- 扩展了公开的用户个人资料。
- 修复了强制检查的配置。
- 改进了内建备份的文档。
- 将源语言属性从项目移动到部件。
- 添加了 Vue I18n 格式化检查。
- 通用占位符的检查现在支持了正则表达式。
- 改进了矩阵模式的外观。
- 机器翻译现在被称为自动建议。
- 增加了与多个 GitLab 或 GitHub 实例交互的支持。
- 扩展了 API 以覆盖项目更新、单元更新与删除，以及术语表。
- 单元 API 现在可以正确处理复数字符串。
- 部件的新建现在能够处理上传的 ZIP 文件或文档。
- 巩固了 API 相应状态代码。
- 在贡献者协议中支持 Markdown。
- 改进了源字符串追踪。
- 改进了 JSON、YAML 和 CSV 格式兼容性。
- 增加了对删除字符串的支持。
- 改进了文件下载的性能。

- 改进了仓库管理视图。
- 为 Android 自动启动 java 格式。
- 增加了对本地化截图的支持。
- 新增了对 Python 3.9 的支持。
- 修复了某些条件下翻译 HTML 文件。

所有变化详情。

4.40 Weblate 4.2.2

发布于 2020 年 9 月 2 日。

- 修复了 JSON 格式源字符串的匹配。
- 修复了一些验证配置的登录重定向。
- 修复了使用组同步的 LDAP 身份验证。
- 修复了与报告自动翻译进度相关的崩溃。
- 修复了启用尾注时的 Git 挤压提交问题。
- 修复了使用 API 创建本地版本控制系统（VCS）部件。

4.41 Weblate 4.2.1

发布于 2020 年 8 月 21 日。

- 修复了在安装资源中一些区域设置存储复数。
- 修复了一些 XLIFF 文件清理附加组件的崩溃。
- 允许在 Docker 镜像中设置本地化 CDN。

4.42 Weblate 4.2

发布于 2020 年 8 月 18 日。

- 改进了用户页面并添加了用户列表。
- 去掉了从 3.x 版本迁移的支持，从 4.0 或 4.1 迁移。
- 添加了几种单语言格式的导出。
- 改进了活动图表。
- 可以配置显示的附近字符串数量。
- 增加了对锁定遇到仓库错误的部件的支持。
- 简化了主导航栏（用图标替换按钮）。
- 改进了 Google Translate 集成中的语言代码处理。
- Git 挤压附加组件可以生成 `Co-authored-by:` 尾注。
- 改进了查询搜索解析。
- 改进了格式字符串检查的用户反馈。
- 改进了批量更改状态的性能。

- 添加了项目或部件重命名后重定向的兼容性。
- 新增了字符串核准、部件锁定和许可证变更的通知。
- 为 ModernMT 添加了支持。
- 允许避免在文件上传时覆盖已核准的译文。
- 去掉了一些对兼容 URL 重定向的支持。
- 新增了对 ECMAScript 模板字面量的检查。
- 添加了关注部件的选项。
- 去掉了来自 JSON 单元密钥的前导的点。
- 删除了翻译记忆库单独的 Celery 队列。
- 允许使用同一种语言同时翻译所有部件。
- 允许配置 `Content-Security-Policy` HTTP 标头。
- 在项目层为语言别名添加支持。
- 帮助 HTML 或 JavaScript 本地化的新附加组件，请参见*JavaScript 本地化 CDN*。
- Weblate 域现在在设置中配置，请参见*SITE_DOMAIN*。
- 增加了对按照部件和项目进行搜索的支持。

4.43 Weblate 4.1.1

发布于 2020 年 6 月 19 日。

- 修复了 Docker 中更改自动修复或附加组件配置。
- 修复了在“关于”页面中可能的崩溃。
- 改进了字节编译的区域设置文件的安装。
- 修复了向术语表添加单词。
- 修复了机器翻译的键盘快捷键。
- 删除了一些设置中导致丢失日志事件的调试输出。
- 修复了在项目列表中所定指示。
- 修复了一些设置中列出 GPG 密钥。
- 为需要使用的 DeepL API 版本添加了选项。
- 为作为 SAML 服务提供商添加了支持，请参见*SAML 身份验证*。

4.44 Weblate 4.1

发布于 2020 年 6 月 15 日。

- 支持了使用包含国家地区代码创建新翻译。
- 支持了搜索带截图的源字符串。
- 扩展了统计数据洞察中可用的信息。
- 改进了在“翻译”页面上的搜索编辑。
- 改进了并发仓库更新的处理。
- 在项目新建表单中包括了源语言。

- 包括了信用的更改计数。
- 修复了一些情况下的 UI 语言选择。
- 允许注册关闭时的白名单注册方法。
- 改进了术语表中相关术语的查找。
- 改进了翻译记忆库匹配。
- 将相同的机器翻译结果分组。
- 为编辑翻译页面的屏幕截图添加了直接链接。
- 改进了删除确认对话。
- 在 ZIP 下载中包括了模板。
- 为公告增加了对 Markdown 的支持和相关通知配置项。
- 扩展了检查列表的细节。
- 新增了对以下文件格式的支持：*Laravel PHP 字符串*, *HTML 文件*, *OpenDocument 格式*, *IDML 格式*, *Windows RC 文件*, *INI 翻译*, *Inno Setup INI 翻译*, *GWT 属性*, *go-i18n JSON 文件*, *ARB 文件*。
- 统一使用已忽略作为已忽略检查的状态。
- 添加了对配置要启用的默认附加组件的支持。
- 修复了编辑器忽略检查的键盘快捷键。
- 改进了带有占位符的字符串的机器翻译。
- 显示了用户语言的幽灵翻译，使之易于启动。
- 改进了语言代码解析。
- 显示了列表中的第一个用户语言的翻译。
- 重命名来塑造为更一般的名称变体。
- 添加了新的质量检查：多个未命名的变量, 长期未翻译, 连续重复的单词。
- 为擦除翻译记忆库重新引入了支持。
- 修复了忽略原文检查的选项。
- 为配置不同分支来解析更改添加了支持。
- API 现在在 HTTP 标头重报告速率限制状态。
- 对 Google Translate V3 API（高级版）添加了支持。
- 添加了对部件层访问限制的能力。
- 为翻译标记中的空格和其它特殊字符添加了支持，请参见使用标记定制行为。
- 总是显示受到的文本检查，如果启动的话。
- API 现在支持对更改的筛选。
- 为项目之间分享术语表添加了支持。

4.45 Weblate 4.0.4

发布于 2020 年 5 月 7 日。

- 修复了测试套件在一些 Python 3.8 环境下的执行。
- 文档中笔误的修复。
- 修复了一些情况下使用 API 新建部件的问题。
- 修复了破坏移动端导航栏的 JavaScript 错误。
- 修复了显示一些检查时的崩溃。
- 修复了屏幕截图列表。
- 修复了每月摘要通知。
- 修复了使用翻译中不存在的单元的中间翻译行为。

4.46 Weblate 4.0.3

发布于 2020 年 5 月 2 日。

- 修复了报告中可能的崩溃。
- 在评论中提及用户时用户名现在不区分大小写。
- 修复了非超级用户的 PostgreSQL 迁移。
- 修复了新建部件时更改仓库 URL。
- 修复了上游仓库丢失时的崩溃。

4.47 Weblate 4.0.2

发布于 2020 年 4 月 27 日。

- 改进了翻译统计数据的性能。
- 改进了更改标签的性能。
- 改进了批量编辑的性能。
- 改进了翻译记忆库的性能。
- 修复了部件删除时可能的崩溃。
- 修复了某些极端情况下翻译变更的显示问题。
- 改进了 celery 队列过长的警告。
- 改进了一致性检查中的误报。
- 修复了更改链接部件仓库时的死锁。
- 包括了更改列表和 CSV 与报告中的编辑距离。
- 避免了对加拿大法语进行符号间隔检查时的误报。
- 修复了用占位符导出 XLIFF。
- 修复了零宽度检查的误报。
- 改进了配置错误的报告。
- 修复了双语原文上传的问题。

- 为 DeepL 机器翻译自动检测支持的语言。
- 修复了某些极端情况下的进度条显示问题。
- 修复了非翻译字符串出发的一些检查。

4.48 Weblate 4.0.1

发布于 2020 年 4 月 16 日。

- 修复了来自 Pypi 的软件包安装。

4.49 Weblate 4.0

发布于 2020 年 4 月 16 日。

- Weblate 现在需要 Python 3.6 或更新版本。
- 添加了部件警报的管理概览。
- 添加了仓库浏览器链接失效的部件警报。
- 改进了登录和注册页面。
- 项目访问控制与工作流程配置集成在项目设置中。
- 为 i18next 插值和嵌套添加了检查和高亮标记。
- 为百分号占位符添加检查和高亮标记。
- 显示建议未通过的检查。
- 在历史中记录源字符串更改。
- 将 Microsoft Translator 升级为版本 3 的 API。
- 重新实现了翻译记忆库后端。
- 为在搜索 中查找几个 `is:` 添加了支持。
- 允许未更改的译文 避免内部黑名单。
- 改进了从单语言 po 文件中提取注释。
- 将白板消息（whiteboard message）改名为公告（announcement）。
- 修复了注册电子邮件偶尔出现的问题。
- 改进了 LINGUAS 更新附加组件来处理更多的语法变量。
- 修复了编辑单语言 XLIFF 源文件。
- 为搜索 中的准确匹配添加了支持。
- 扩展了 API 覆盖屏幕截图、用户、群组、部件列表，并扩展了新建项目。
- 新增了对双语翻译原文上传的支持。
- 为开发者的中间语言添加支持。
- 为源字符串复查添加支持。
- 扩展了平台范围的翻译记忆库的下载选项。

4.50 Weblate 3.x 系列

4.50.1 Weblate 3.11.3

发布于 2020 年 3 月 11 日。

- 修复了以某种优先性来搜索字段。
- 修复了最近添加的字符串的预设查询。
- 修复了搜索返回重复的匹配。
- 修复 Gmail 中提供的通知。
- 修复了从历史中还原更改。
- 在摘要通知中增加了事件的链接。
- 修复了账户删除确认的电子邮件。
- 添加了对 Docker 容器中 Slack 身份认证的支持。
- 避免发送未订阅语言的通知。
- 在性能概况中包括了 Celery 队列。
- 修复了附加组件的文档链接。
- 减少了未更改的译文检查的误报。
- 提升了 bleach 依赖项以解决 CVE-2020-6802。
- 修复了在历史中列出项目层的更改。
- 修复了某些极端情况下的统计数据失效问题。
- 修复了搜索某条字符串状态。
- 改进了格式字符串检查行为丢失百分比的问题。
- 修复了使用第三方提供商的身份验证。

4.50.2 Weblate 3.11.2

发布于 2020 年 2 月 22 日。

- 修复了建议的呈现问题。
- 修复了一些字符串被错误地报告为没有单词的问题。

4.50.3 Weblate 3.11.1

发布于 2020 年 2 月 20 日。

- 记录 Celery 设置更改。
- 改进了新建部件时文件名的验证。
- 修复了一些依赖项的最低版本。
- 修复了以某个 Django 版本添加群组。
- 修复了手动推送到上游仓库。
- 改进了术语表匹配。

4.50.4 Weblate 3.11

发布于 2020 年 2 月 17 日。

- 允许通过 API 新建部件的过程中使用版本控制系统（VCS）推送 URL。
- 宽度检查现在以渲染来显示图片。
- 修复了通知电子邮件中的链接。
- 改进了纯文本电子邮件的外观。
- 显示了忽略的检查并且允许使它们再次激活。
- 在单语言翻译上显示附件的键。
- 添加了对分组字符串整形的支持。
- 在系统检查时推荐升级到新版 Weblate。
- 对重复的语言警报提供了更详细的分析。
- 在项目页面上显示更详细的许可证信息。
- 如果需要的话自动非浅复制本地复件。
- 修复了需要处理的字符串的下载问题。
- 新警报，警告两次使用相同的文件掩码。
- 改进了 XML 代码块提取。
- *SINGLE_PROJECT* 现在可以强制重定向来选择项目。
- 添加了解决评论的选项。
- 添加了批量编辑标记功能。
- 为 labels 添加了支持。
- 添加了批量编辑附加组件。
- 为强制检查 添加了选项。
- 增加了确认链接的默认验证。
- 改进了 Matomo 集成。
- 修复了曾被翻译过 来正确低处理源字符串更改。
- 扩展了通过*AUTO_UPDATE* 的自动更新配置。
- LINGUAS 附加组件现在在 Weblate 中完全同步翻译。

4.50.5 Weblate 3.10.3

发布于 2020 年 1 月 18 日。

- 支持 translate-toolkit 2.5.0。

4.50.6 Weblate 3.10.2

发布于 2020 年 1 月 18 日。

- 为项目添加了锁定指示。
- 修复了导致某些浏览器闪烁的 CSS 错误。
- 修复了以非英语地区设置在系统中搜索。
- 改进了对 Github 和 Bitbucket 钩子的仓库。
- 修复了一些 Python 2.7 安装中的数据的迁移。
- 允许配置 Git 浅克隆。
- 改进了后台通知处理。
- 修复了在浏览器中导航返回时中断的表单提交。
- 配置 YAML 格式化的新附加组件。
- 修复了单复数形式语言中没有进行相同的复数检查。
- 修复了某些字段上的正则表达式搜索。

4.50.7 Weblate 3.10.1

发布于 2020 年 1 月 9 日。

- 扩展了新建翻译 API。
- 修复了数据迁移的几个极端情况。
- 与 Django 3.0 兼容。
- 改进了数据清理性能。
- 为定制的 security.txt 添加了支持。
- 改进了更新日志的面包屑菜单。
- 改进了操作面板上的翻译列表。
- 改进了 Webhooks 的 HTTP 响应。
- 新增了在 Docker 容器中对 GitLab 合并请求的支持。

4.50.8 Weblate 3.10

发布于 2019 年 12 月 20 日。

- 改进了应用用户界面。
- 添加了双空格检查。
- 修复了新建新的语言。
- 避免了向删除的电子邮箱发送审计日志。
- 新增支持只读字符串。
- 在评论中增加了对 Markdown 的支持。
- 允许在项目信息中放置翻译指示文本。
- 为第二语言添加了复制到剪贴板。
- 改进了对 Mercurial 的支持。
- 改进了 Git 仓库获取性能。

- 添加了搜索字符串时间的查找。
- 为所有翻译显示源语言。
- 显示字符串附近的上下文。
- 为仓库操作的通知添加了支持。
- 改进了翻译列表。
- 扩展了搜索能力。
- 为标记为编辑的自动翻译字符串添加了支持。
- 避免对链接部件警报发送重复通知。
- 改进了默认的合并请求消息。
- 更好地指示了在禅模式下的字符串状态。
- 为 Yandex 翻译中的更多语言添加了支持。
- 改进了通知电子邮件的外观。
- 提供了翻译许可的选择。

4.50.9 Weblate 3.9.1

发布于 2019 年 10 月 28 日。

- 从备份中删除了一些不需要的文件。
- 修复了报告的潜在崩溃。
- 修复了数据库交叉迁移故障。
- 对强制推送 Git 仓库添加了支持。
- 降低了注册令牌非法的风险。
- 修复了账户删除点击率的限制。
- 添加了基于优先性的搜索。
- 修复了向 JSON 文件添加字符串可能的崩溃。
- 安全 HTML 检查于修复现在接受源字符串标记。
- 避免向邀请的和删除的用户发送通知。
- 修复了在 Docker 容器中 Celery 的到 redis 的 SSL 连接。

4.50.10 Weblate 3.9

发布于 2019 年 10 月 15 日。

- 在下载的文件中包括了 Weblate 元数据。
- 改进了未通过检查的 UI。
- 在格式检查中指示缺少的字符串。
- 将法语标点间隔检查分开。
- 为修复一些质量检查错误添加了支持。
- 添加了分开的权限来新建新的项目。
- 扩展了字符计数的统计数据。
- 改进了对 Java 风格语言代码的支持。

- 为新的占位符提供的通用检查。
- 为 WebExtension JSON 占位符添加了支持。
- 为纯 XML 格式添加了支持。
- 扩展了 API 的项目、部件和翻译的删除与新建。
- 为 Gitea 和 Gitee webhooks 添加了支持。
- 添加了新的定制的基于正则表达式的检查。
- 允许配置对共享翻译记忆库的贡献。
- 添加了更多翻译文件的 ZIP 下载。
- 使 XLIFF 标准兼容最大宽度和字体的解析。
- 为传输 web 应用的安全 HTML 标记添加了新的检查并进行了修复。
- 对不支持的配置添加了部件警报。
- 添加了自动翻译附加组件来引导翻译。
- 扩展了自动翻译来添加建议。
- 在概览上显示附加组件参数。
- 现在通过当代的 Sengtry SDK 而不是 Raven 支持 Sentry。
- 更改了示例设置，更好地适配生产环境。
- 添加了使用 BorgBackup 的自动备份。
- 分离了 RESX 清理附加组件，来避免不想要的文件更新。
- 添加了高级搜索功能。
- 允许用户下载他们自己的报告。
- 添加了本地化向导来配置部件。
- 增加了对 GitLab Merge Request 的支持。
- 改进了仓库状态的显示。
- 在后台执行自动翻译。

4.50.11 Weblate 3.8

发布于 2019 年 8 月 15 日。

- 为简化创建相似的部件提供了支持。
- 为从基于 XML 的文件格式分析翻译标记添加了支持。
- 将意外记入 Celery 日志。
- 提高仓库范围附加组件的性能。
- 改进了通知电子邮件的外观。
- 修复了密码重置行为。
- 改进了多数翻译页面的性能。
- 修复了 Weblate 未知的语言列表。
- 支持将附加组件克隆到发现的部件。
- 为用上传来替换文件内容添加了支持。
- 为翻译非基于版本控制系统（VCS）的内容添加了支持。
- 添加了 OpenGraph 小挂件图片，以便在社交网络使用。

- 为动画屏幕截图添加了支持。
- 改进了单语言 XLIFF 文件的处理。
- 避免为单个事件发送多个通知。
- 为筛选更改添加了支持。
- 扩展了报告的预定义时间范围。
- 为 Azure Repos 添加了 webhook 支持。
- 新的可选的待处理建议或未翻译字符串通知。
- 添加了通知电子邮件的一键退订。
- 修复了已翻译检查的误报。
- 管理员的新管理界面。
- 字符串优先性现在可以使用标记来指定。
- 添加了语言管理视图。
- 添加了 Qt 库和 Ruby 格式字符串的检查。
- 添加了配置来更好地符合单一项目安装。
- 在单语言翻译中通过源字符串更改中的新字符串。
- 添加带有搜索功能的另外的翻译记忆库视图。

4.50.12 Weblate 3.7.1

发布于 2019 年 6 月 28 日。

- 文档更新。
- 修复了一些要求限制。
- 更新了语言数据库。
- 本地化更新。
- 各种用户界面调整。
- 改进了对不支持但发现的翻译文件的处理。
- 更详细地报告丢失文件格式要求。

4.50.13 Weblate 3.7

发布于 2019 年 6 月 21 日。

- 添加了用于通知的另外的 Celery 队列。
- 对于 API 浏览使用了与应用一致的外观。
- 在报告中包含已核准相关的统计数据。
- 当更新翻译部件时的报告过程。
- 允许终止运行后台部件的更新。
- 为文件名操作扩展了模板语言。
- 对于编辑器链接和仓库浏览器 URL 使用模板。
- 当编辑翻译时指示最大长度和当前字符计数。
- 改进了未更改翻译检查中的缩写处理。

- 为新贡献者设计的全新着陆页。
- 支持配置 msgmerge 附加组件。
- 当发送通知时延迟打开 SMTP 连接。
- 改进了错误日志。
- 允许 MO 生成附加组件中的自定义位置。
- 添加了清理陈旧建议或评论的附加组件。
- 添加了在禅编辑器中启用水平模式的选项。
- 提高了许多被链接部件的导入性能。
- 修复了一些情况下的示例安装。
- 改进了变更中警报的呈现方式。
- 添加了新水平统计数据小挂件。
- 改进了对复数的格式字符串检查。
- 添加了字体管理工具。
- 提供文本尺寸的新检查。
- 为字幕格式添加了支持。
- 包括了语言的全部完成的统计数据。
- 添加了在项目或全体范围内的报告。
- 改进了显示翻译状态时的用户界面。
- 新的 Weblate 标志和配色方案。
- 位图徽章的新外观。

4.50.14 Weblate 3.6.1

发布于 2019 年 4 月 26 日。

- 改进了单语言 XLIFF 文件的处理。
- 修复了某些极端情况下的摘要通知。
- 修复了附加组件脚本错误警报的问题。
- 修复了为单语言 PO 文件生成 MO 文件。
- 修复了未安装检查的显示。
- 指示项目列表中管理的项目。
- 允许更新而从丢失的版本控制系统（VCS）仓库恢复。

4.50.15 Weblate 3.6

发布于 2019 年 4 月 20 日。

- 为下载用户数据添加了支持。
- 附加组件在安装时会被自动触发。
- 改进了解决合并冲突的指示。
- 清理附加组件现在与应用商店元数据翻译兼容。
- 当添加新语言时可配置的语言代码语法。

- 警告 Python 2 的使用计划于 2020 年 4 月终止支持。
- 为视觉键盘从源字符串中提取特定字符。
- 扩展了贡献者统计数据，来反映原文和目标相关计数。
- 管理员和一致性附加组件现在可以添加翻译，即使对用户禁用。
- 修复了停用 `Language-Team` 标头操作说明的错误。
- 通知评论中提及的用户。
- 从部件设置中删除了文件格式自动检测。
- 修复了为单语言 PO 文件生成 MO 文件。
- 添加了摘要通知。
- 为部件通知静音添加了支持。
- 添加了新警报、白板消息或部件的通知。
- 现在可以配置管理项目的通知。
- 改进了三字母语言代码的处理。

4.50.16 Weblate 3.5.1

发布于 2019 年 3 月 10 日。

- 修复了 Celery systemd 单元示例。
- 修复了带有登录的 HTTP 仓库的通知。
- 修复了单语言翻译中编辑源字符串的竞态条件。
- 在日志中包括附加组件执行失败的输出。
- 改进了添加新语言选择的验证。
- 在部件设置中允许编辑文件格式。
- 更新安装指示为首选 Python 3。
- 装入翻译时的性能与一致性改进。
- 使微软术语服务与当前的 Zeep 发布版本兼容。
- 本地化更新。

4.50.17 Weblate 3.5

发布于 2019 年 3 月 3 日。

- 改进了内建翻译记忆库的性能。
- 添加了管理全体翻译记忆库的界面。
- 改进了对不良部件状态的警报。
- 添加了管理白板消息的用户界面。
- 现在可以配置附加组件提交说明。
- 减少更新上游仓库时的提交数量。
- 修复了在项目之间移动部件时可能的元数据丢失。
- 改进了禅模式的导航。
- 添加了几个新的质量检查（相关的标记和 URL ）。

- 新增了对应用商店元数据文件的支持。
- 为切换 GitHub 或 Gerrit 集成添加了支持。
- 为卡什达字母添加了检查。
- 添加了根据作者挤压提交的选项。
- 改进了对 XLSX 文件格式的支持。
- 与 Tesseract 4.0 兼容。
- 账单附加组件现在在未支付账单 45 天后删除项目。

4.50.18 Weblate 3.4

发布于 2019 年 1 月 22 日。

- 为 XLIFF 占位符添加了支持。
- Celery 现在可以使用多个项目队列。
- 为项目与部件的重命名和移动添加了支持。
- 在报告中包括字符计数。
- 添加了带有翻译文件自动检测的翻译部件添加向导。
- 可定制的 Git 合并提交说明。
- 在导航栏中添加了部件警报的视觉指示。
- 改进了装入翻译文件的性能。
- 在推送前挤压提交的新附加组件。
- 改进了翻译更改的显示。
- 将默认的合并方式改为变基，并使其可配置。
- 更好地处理语言代码中的私用子标签。
- 改进了全文索引更新的性能。
- 扩展了文件上传 API 来支持更多的参数。

4.50.19 Weblate 3.3

发布于 2018 年 11 月 30 日。

- 为部件和项目删除添加了支持。
- 改进了一些单语言翻译的性能。
- 添加了翻译部件警报，以突出显示翻译中的问题。
- XLIFF 字符串 resname 可用时将其暴露为上下文。
- 为 XLIFF 状态添加支持。
- 为 DATA_DIR 中的非可写入文件添加检查。
- 改进了更改的 CSV 导出。

4.50.20 Weblate 3.2.2

发布于 2018 年 10 月 20 日。

- 删除了不再需要的 Babel 依赖项。
- 更新了语言定义。
- 改进了附加组件、LDAP 和 Celery 的文档。
- 修复了启动新的 dos-eol 和 auto-java-messageformat 标记。
- 修复了从 PyPI 软件包运行 setup.py 测试。
- 改进了复数处理。
- 修复了某些极端情况下的翻译上传 API 故障。
- 修复了手动更改情况下更新 Git 配置。

4.50.21 Weblate 3.2.1

发布于 2018 年 10 月 18 日。

- Python 2.7 的 backports.csv 的文档依赖项。
- 修复根下运行测试。
- 改进了 gitexport 模块中的错误处理。
- 修复了报告新添加语言的过程。
- 正确地向 Sentry 报告 Celery worker 错误。
- 修复了用 Qt linguist 新建新的翻译。
- 修复了偶发的全文索引更新失败。
- 改进了新建新部件时的验证。
- 为清理旧的建议添加了支持。

4.50.22 Weblate 3.2

发布于 2018 年 10 月 06 日。

- 为自动化的附加组件安装添加 install_addon 管理命令。
- 允许更细粒度的速率限制设置。
- 为 Excel 文件的导出和导入添加了支持。
- 改进了多部件发现附加组件情况下的部件清理。
- 重写了微软术语机器翻译后端。
- Weblate 现在使用 Celery 来卸载一些处理。
- 改进了搜索功能，并添加了正则表达式搜索。
- 为有道智云机器翻译添加了支持。
- 为百度 API 机器翻译添加了支持。
- 集成了使用 Celery 的维护和清理任务。
- 改进了装入翻译的性能几乎 25%。
- 删除了对上传时合并标头的支持。
- 删除了对自定义提交说明的支持。

- 可配置的编辑模式（禅模式/完整模式）。
- 为向 Sentry 报告错误添加了支持。
- 为每天自动更新仓库添加了支持。
- 为用户新建项目和部件添加了支持。
- 内建的翻译记忆库现在自动存储完成的翻译。
- 用户和项目现在可以导入他们现有的翻译记忆库。
- 更好地管理屏幕截图的相关字符串。
- 为检查 Java MessageFormat 添加了支持。

已解决问题的详细列表请参见 GitHub 上的 3.2 里程碑。

4.50.23 Weblate 3.1.1

发布于 2018 年 7 月 27 日。

- 修复了一些设置中失败的测试套件。

4.50.24 Weblate 3.1

发布于 2018 年 7 月 27 日。

- 不再支持从早于 3.0.1 的更旧版本的升级。
- 允许覆盖设置中的默认提交说明。
- 改进 webhooks 与自主环境的兼容性。
- 添加了对 Amazon Translate 的支持。
- 与 Django 2.1 兼容。
- Django system 系统检查现在用于诊断安装问题。
- 删除了对即将关闭的 libravatar 服务的支持。
- 将未改变的翻译标记为需要编辑的新附加组件。
- 为翻译时跳到特定位置添加支持。
- 现在可以定制下载翻译。
- 改进了翻译记忆库中匹配字符串相似度的计算。
- 支持了使用 GnuPG 为 Git 提交签名。

4.50.25 Weblate 3.0.1

发布于 2018 年 6 月 10 日。

- 修复了自 2.20 依赖可能的合并问题。
- 本地化更新。
- 删除了过时的钩子的示例。
- 改进了缓存文档。
- 改进了管理文档的显示。
- 改进了长语言名称的处理。

4.50.26 Weblate 3.0

发布于 2018 年 6 月 1 日。

- 重写了访问控制。
- 导致移动或重命名模块的一些代码清理。
- 自动发现部件的新附加组件。
- import_project 管理命令现在有一些略微不同的参数。
- 为 Windows RC 文件添加了基本支持。
- 新的附加组件，将贡献者姓名存储在 PO 文件标头中。
- 删除了每个部件的钩子脚本，请改用附加组件。
- 为收集贡献者协议添加支持。
- 现在在历史中跟踪访问控制更改。
- 新的附加组件，用于确保项目中的所有部件都有相同的翻译。
- 在提交说明模板中支持更多变量。
- 为提供另外的文本上下文添加支持。

4.51 Weblate 2.x 系列

4.51.1 Weblate 2.20

发布于 2018 年 4 月 4 日。

- 改进了 subversion 仓库克隆的速度。
- 更改了仓库锁定来使用第三方库。
- 支持了只下载需要处理的字符串。
- 支持了同时搜索多种语言。
- 新附加组件来配置 gettext 输出换行。
- 新附加组件来配置 JSON 格式化。
- 为使用 RFC 6750 兼容的 Bearer 认证添加了 API 中认证的支持。
- 未使用机器翻译服务自动翻译添加了支持。
- 为白板消息中的 HTML 标记添加了支持。
- 为大量更改字符串的状态添加了支持。
- 现在至少需要 2.3.0 版本的 Translate-toolkit，较老的版本不再支持。
- 添加了内建翻译记忆库。
- 在操作面板和每个部件列表概览页面中增加了部件列表概览。
- 为 DeepL 机器翻译服务添加了支持。
- 机器翻译结果现在缓存在 Weblate 内。
- 增加了对已提交更改重新排序的支持。

4.51.2 Weblate 2.19.1

发布于 2018 年 2 月 20 日。

- 修复了从 2.18 升级的合并问题。
- 改进了文件上传 API 验证。

4.51.3 Weblate 2.19

发布于 2018 年 2 月 15 日。

- 修复了跨一些文件格式的导入。
- 在审查日志中显示用户友好的浏览器信息。
- 为文件添加了 TMX 导出程序。
- 装入翻译文件的各种性能改进。
- 添加了选项在 Weblate 中禁止访问管理，有利于 Django。
- 改进了大字符串的术语表查询速度。
- 与 django_auth_ldap 1.3.0 兼容。
- 配置错误现在被存储并被持久地报告。
- 在空格自动修复中遵从忽略标记。
- 改进了一些 Subversion 设置的兼容性。
- 改进了内建机器翻译服务。
- 对 SAP 翻译中心服务添加了支持。
- 新增了对微软术语服务的支持。
- 删除了对通知电子邮件中的广告的支持。
- 改进了语言层次的翻译过程报告。
- 改进了对不同复数公式的支持。
- 添加了对不使用 stdlayout 的 Subversion 仓库的支持。
- 添加了自定义翻译工作流程的附加组件。

4.51.4 Weblate 2.18

发布于 2017 年 12 月 15 日。

- 扩展了贡献者的统计数据。
- 改进了特殊字符可视键盘的配置。
- 为 DTD 文件格式添加了支持。
- 更改了键盘快捷键，以减少与浏览器/系统快捷键的冲突。
- 改进了对 XLIFF 文件中的已核准标记的支持。
- 添加了对 gettext PO 文件中不换行的长字符串的支持。
- 添加了按钮为当前翻译复制永久链接。
- 去掉了对 Django 1.10 的支持，而添加了对 Django 2.0 的支持。
- 删除了当翻译时对翻译的锁定。
- 对于为单语言翻译添加新的字符串添加了支持。

- 新增了对具有专门审校员的翻译流程的支持。

4.51.5 Weblate 2.17.1

发布于 2017 年 10 月 13 日。

- 修复了在一些特定情况下运行测试套件的问题。
- 区域设置更新。

4.51.6 Weblate 2.17

发布于 2017 年 10 月 13 日。

- 现在 Weblate 默认进行 Git 浅克隆。
- 改进了更新大的翻译文件时的性能。
- 为阻挡注册的某个电子邮箱地址添加了支持。
- 用户现在可以删除自己的评论。
- 为搜索和替换特性添加了预览步骤。
- 搜索和上传表单时客户端一侧设置的持久性。
- 扩展了搜索能力。
- 每个项目的更细粒度的 ACL 配置。
- BASE_DIR 的默认值已经更改。
- 添加了两步账户删除，来防止意外删除。
- 现在可以编辑项目访问控制设置。
- 添加了可选的垃圾保护功能，建议使用 Akismet。

4.51.7 Weblate 2.16

发布于 2017 年 8 月 11 日。

- 各种性能改进。
- 为嵌套 JSON 格式添加了支持。
- 为 WebExtension JSON 格式添加了支持。
- 修复了 git 导出认证。
- 改进了某些情况下的 CSV 导入。
- 改进了其它翻译小挂件的外观。
- 对于表单中的文本长度现在强制进行最大长度检测。
- 每个部件进行 commit_pending 时间配置。
- 各种用户界面清理。
- 修复了部件/项目/网站范围的翻译搜索。

4.51.8 Weblate 2.15

发布于 2017 年 6 月 30 日。

- 在其它翻译中显示了更多相关的翻译。
- 添加了选项来查看当前字符串到其它语言的翻译。
- 对立陶宛语默认使用 4 种复数形式。
- 修复了不同格式单语言文件的上传。
- 改进了认证失败的错误消息。
- 当从术语表删除单词时保持页面状态。
- 添加了直接链接来编辑第二语言翻译。
- 添加了 Perl 格式质量检查。
- 为了拒绝再次使用的密码添加了支持。
- 为编辑 RTL 语言扩展了工具条。

4.51.9 Weblate 2.14.1

发布于 2017 年 5 月 24 日。

- 修复了将搜索结果分页时可能的错误。
- 修复了某些极端情况下从较老的版本合并的问题。
- 修复了关注和取消关注项目时可能出现的 CSRF。
- 密码重置不再认证用户。
- 修复了忘记密码是可能的略过 CAPTCHA。

4.51.10 Weblate 2.14

发布于 2017 年 5 月 17 日。

- 使用 AJAX 添加术语表条目。
- 退出登录现在使用 POST 来避免 CSRF。
- API 密钥令牌重置现在使用 POST 来避免 CSRF。
- Weblate 默认设置 Content-Security-Policy。
- 验证本地编辑器 URL 来避免 self-XSS。
- 现在默认相对于一般缺陷来验证密码。
- 向用户通知他们账户的重要活动，如密码变更。
- CSV 导出现在会避免可能的公式。
- 安全上的各种小改进。
- 现在限制了认证尝试的次数。
- 建议的内容存储在历史中。
- 在审计日志中存储重要的账户活动。
- 当删除账户或添加新的团体时，询问密码确认。
- 显示提出建议的时间。
- 新的质量检查：分号结尾。

- 确保搜索链接可以被分享。
- 在 API 中包括了源字符串信息和屏幕截图。
- 允许通过 API 上传覆盖翻译。

4.51.11 Weblate 2.13.1

发布于 2017 年 4 月 12 日。

- 修复了个人资料中管理的项目的列表。
- 修复了在一些许可丢失的情况下的合并问题。
- 修复了翻译下载中当前文件格式的列表。
- 当用户没有特权而尝试访问项目时返回 HTTP 404。

4.51.12 Weblate 2.13

发布于 2017 年 4 月 12 日。

- 修复了翻译模板的质量检查。
- 添加了在丢失翻译时触发的质量检查。
- 添加了查看用户待处理建议的选项。
- 添加选项来自动建立部件列表。
- 未经身份验证用户可以配置默认操作面板。
- 添加选项来浏览 25 个随机字符串用于复查。
- 历史现在指示字符串的更改。
- 添加新的翻译时更好的错误报告。
- 在项目中添加了每种语言的搜索。
- Group（群组）ACLs 现在可以限制为某个权限。
- 现在使用 Group ACL 来实现每项目的 ACL。
- 添加了更精细颗粒的特权控制。
- 各种小的 UI 改进。

4.51.13 Weblate 2.12

发布于 2017 年 3 月 3 日。

- 改进了群组的管理界面。
- 为 Yandex Translate API 添加了支持。
- 改进了网站范围的搜索速度。
- 添加了项目和部件范围的搜索。
- 添加了项目和部件范围的搜索与替换。
- 改进了不一致翻译的渲染。
- 支持了在本地编辑器中打开源文件。
- 为配置带有特殊字符的虚拟键盘添加了支持。
- 改进了带有 OCR 支持的屏幕截图管理来匹配源字符串。

- 默认提交说明现在包括翻译信息和 URL。
- 为 Joomla 翻译格式添加了支持。
- 改进了重要的跨文件格式的可靠性。

4.51.14 Weblate 2.11

发布于 2017 年 1 月 31 日。

- 在语言页面包括了具体语言信息。
- Mercurial 后端改进。
- 添加了选项来制定翻译部件优先性。
- 更一致地使用 Group ACLs，即使具有较少使用的许可。
- 为钩子脚本添加了 WL_BRANCH 变量。
- 改进了开发者文档。
- 在 Git 导出器附加组件中与各种 Git 版本有更好地兼容。
- 包括每个项目和部件的统计数据。
- 添加语言代码映射，来更好地支持 Microsoft Translate API。
- 将全文清理移动为后台工作，使去除翻译更快速。
- 修复了具有单复数形式的语言的复数原文显示问题。
- 改进了 import_project 中的错误处理。
- 各种性能改进。

4.51.15 Weblate 2.10.1

发布于 2017 年 1 月 20 日。

- 不泄露密码重置表单上存在账户（CVE-2017-5537）。

4.51.16 Weblate 2.10

发布于 2016 年 12 月 15 日。

- 添加质量检查，来检查是否复数被翻译出来。
- 对带有验证的仓库修复了 GitHub 钩子。
- 添加了可选的 Git 导出模块。
- 支持 Microsoft Cognitive Services Translator API。
- 简化了项目和部件用户界面。
- 添加了自动修复来删除控制字符。
- 为项目添加了每种语言的概况。
- 为 CSV 导出添加了支持。
- 为统计数据添加了 CSV 下载。
- 增加了矩阵视图，可用于快速概览所有翻译。
- 为更改和字符串添加了基本 API。
- 为 Apertium APy 服务器添加了支持，用于机器翻译。

4.51.17 Weblate 2.9

发布于 2016 年 11 月 4 日。

- 扩展了 createadmin 管理命令的参数。
- 扩展了 import_json，而能够处理现有部件。
- 新增了对 YAML 文件的支持。
- 项目所有者现在可以配置翻译部件和项目细节。
- 使用”Watched”（已关注）而不是”Subscribed”（已订阅）项目。
- 可以从项目页面直接关注项目。
- 添加多语言状态小挂件。
- 如果不显示原文，则将第二语言高亮。
- 将建议删除记录在历史中。
- 改进个人资料中语言选择的用户体验。
- 修复了部件的白板消息显示问题。
- 保存后保持偏好选项卡被选中。
- 更突出地显示源字符串注释。
- 从 Git 仓库自动安装 Gettext PO 合并驱动。
- 添加搜索和替换特性。
- 支持上传翻译的可视化上下文（截图）。

4.51.18 Weblate 2.8

发布于 2016 年 8 月 31 日。

- 文档改进。
- 翻译。
- 更新了附带的 JavaScript 库。
- 添加了 list_translators 管理命令。
- 不再支持 Django 1.8。
- 修复了与 Django 1.10 的兼容性。
- 添加了 Subversion 支持。
- 将 XML 有效性检查与 XML 不匹配的标签检查分开。
- 修复了 API 来接受 HIDE_REPO_CREDENTIALS 设置。
- 在禅模式中显示原文变更。
- 现在也可以在禅模式下也可以使用 Alt+PageUp/PageDown/Home/End 了。
- 添加了显示更改准确时间的工具提示。
- 添加选项从翻译页面中选择筛选程序和搜索。
- 添加了翻译删除的 UI。
- 改进了插入代码块时的行为。
- 修复了禅模式的自动锁定问题。

4.51.19 Weblate 2.7

发布于 2016 年 7 月 10 日。

- 去除了 Google web translation 机器翻译。
- 改进了添加翻译时的提交说明。
- 修复了希伯来语的 Google Translate API。
- 与 Mercurial 3.8 的兼容性。
- 添加了 import_json 管理命令。
- 纠正了列出翻译的顺序。
- 显示全部建议文本，而只是差别。
- 扩展 API（具体的仓库状态、统计数据、……）。
- 测试套件不再需要网络范文来测试仓库。

4.51.20 Weblate 2.6

发布于 2016 年 4 月 28 日。

- 修复了语言筛选程序的验证部件。
- 改进了对 XLIFF 文件的支持。
- 修复了非英语原文的机器翻译。
- 添加了 REST API。
- Django 1.10 的兼容性。
- 增加了白板消息分类。

4.51.21 Weblate 2.5

发布于 2016 年 3 月 10 日。

- 修复了项目所有者的自动翻译。
- 改进了提交和推送操作的性能。
- 新的管理命令，从命令行添加建议。
- 为文件上传时的合并注释添加了支持。
- 为 C printf 格式的一些 GNU 后缀添加了支持。
- 文档改进。
- 为生成译者信用添加了支持。
- 为生成贡献者统计数据添加了支持。
- 网站范围的搜索可以只在一种语言中搜索。
- 改进了亚美尼亚语的质量检查。
- 为开始没有现有翻译的翻译部件添加支持。
- 支持在 Qt TS 中添加新的翻译。
- 改进了翻译 PHP 文件的支持。
- 质量检查的性能提高。
- 修复了全站搜索未通过的检查的问题。

- 新增了指定源语言的选项。
- 改进了对 XLIFF 文件的支持。
- 扩展了 import_project 选项的列表。
- 改进了白板消息的目标。
- 支持跨项目的自动翻译。
- 优化全文搜索索引。
- 为自动翻译添加了管理命令。
- 添加了代码块高亮。
- 为代码块、检查和机器翻译添加了键盘快捷键。
- 改进了翻译锁定。
- 为 AngularJS 插值添加了质量检查。
- 添加了广泛的基于群组的 ACLs。
- 阐明了需要编辑的字符串的术语（以前被称为 fuzzy，即模糊）。
- 明确了需要处理的字符串和未翻译的字符串相关术语。
- 支持 Python 3。
- 去掉了对 Django 1.7 的支持。
- 去掉了用于新建 gettext PO 文件的 msginit 依赖项。
- 增加了操作面板视图的配置项。
- 改进了解析错误的通知。
- 在 import_project 中增加了导入名称重复的部件的选项。
- 改进了翻译 PHP 文件的支持。
- 添加了字典的 XLIFF 导出。
- 为所有翻译添加了 XLIFF 和 gettext PO 导出。
- 文档改进。
- 新增了对可配置的自动群组分配的支持。
- 改进了新翻译的添加。

4.51.22 Weblate 2.4

发布于 2015 年 9 月 20 日。

- 改进了对 PHP 文件的支持。
- 为匿名用户添加 ACL 的能力。
- 改进了 import_project 命令的配置。
- 添加了历史的 CSV 转储。
- 避免复制/粘贴空白字符的错误。
- 为 Bitbucket webhooks 添加了支持。
- 在翻译上传时对模糊字符串更严格的控制。
- 几个 URLs 已经更改，可能必须要更新您的书签。
- 钩子脚本以版本控制系统（VCS）root 作为当前目录执行。
- 钩子脚本以描述当前部件的环境变量来执行。

- 添加了管理命令来优化全文索引。
- 为滚动条上报告错误添加了支持。
- 项目现在可以有多个所有者。
- 项目所有者可以管理自己。
- 为 gettext PO 中使用的 `javascript-format` 添加了支持。
- 在 XLIFF 中添加新翻译的支持。
- 改进了文件格式自动检测。
- 扩展了键盘快捷键。
- 改进了多种语言的字典匹配。
- 改进了多数页面的布局。
- 支持在翻译时将单词添加入字典。
- 支持了筛选 Weblate 管理的语言。
- 为翻译并导入 CSV 文件添加了支持。
- 重写了静态文件的处理。
- 如果只有第三方服务，则登录/注册直接链接到第三方服务。
- 账户删除时提交待处理的更改。
- 添加管理命令来更改网站名称。
- 添加选项来配置默认提交者。
- 在添加新翻译后添加钩子。
- 添加选项来指定多个文件来添加提交。

4.51.23 Weblate 2.3

发布于 2015 年 5 月 22 日。

- 去掉对 Django 1.6 和 South migrations 的支持。
- 支持使用 Java 属性文件时添加新翻译。
- 允许接受建议而不编辑。
- 改进了对 Google OAuth 2.0 的支持。
- 添加了对 Microsoft .resx 文件的支持。
- 微调了默认的 robots.txt 文件而不允许翻译特别缓慢地进行。
- 简化了接受建议的工作流程。
- 添加了总是接收重要通知的项目所有者。
- 允许禁止编辑单语言模板。
- 更具体的仓库状态视图。
- 更改翻译时编辑模板的直接链接。
- 允许为项目所有者添加更多的权限。
- 允许在禅模式下显示第二语言。
- 支持隐藏源字符串而有利于第二语言。

4.51.24 Weblate 2.2

发布于 2015 年 2 月 19 日。

- 性能改进。
- 对位置和评论字段进行全文搜索。
- 基于 SVG/JavaScript 的新活动图表。
- 支持 Django 1.8。
- 支持删除评论。
- 添加自己的 SVG 徽章。
- 新增了对 Google Analytics（分析）的支持。
- 改进了翻译文件名的处理。
- 为单语言 JSON 翻译添加了支持。
- 在历史中记录部件锁定。
- 支持为单语言翻译编辑源（模板）语言。
- 为 Gerrti 添加了基本支持。

4.51.25 Weblate 2.1

发布于 2014 年 12 月 5 日。

- 为 Mercurial 仓库添加了支持。
- 由 Awesome 替换了 Glyphicon 字体。
- 添加了社交验证服务的图标。
- 更一致的按钮颜色和图标。
- 文档改进。
- 各种缺陷修复。
- 对于小屏幕自动隐藏翻译列表中的列。
- 更改了文件系统路径的配置。
- 改进了 SSH 密钥处理与存储。
- 改进了仓库锁定。
- 每个源字符串可定制的质量检查。
- 允许隐藏操作面板上已完成的翻译。

4.51.26 Weblate 2.0

发布于 2014 年 11 月 6 日。

- 使用 Bootsctrap 的新的仓库 UI。
- 重写了版本控制系统（VCS）后端。
- 文档改进。
- 为全站范围的消息添加了白板。
- 可配置的字符串优先性。
- 为 JSON 文件格式添加了支持。

- 修复了某些情况下生成 mo 文件。
- 为 GitLab 通知添加了支持。
- 添加了支持来禁止翻译建议。
- Django 1.7 支持。
- ACL 项目现在具有用户管理。
- 扩展了搜索可能性。
- 为译者给出复数的更多提示。
- 修复了 Git 仓库锁定。
- 与旧的 Git 版本的兼容性。
- 改进了 ACL 支持。
- 新增了每种语言的引号和其他特殊字符按钮。
- 支持导出统计数据作为 JSONP。

4.52 Weblate 1.x 系列

4.52.1 Weblate 1.9

发布于 2014 年 5 月 6 日。

- Django 1.6 兼容性。
- 不再维护与 Django 1.4 的兼容性。
- 用于锁定/解锁翻译的管理命令。
- 改进了对 Qt TS 文件的支持。
- 用户现在可以删除他们的账户。
- 头像可以被禁止。
- 合并了名字和姓氏属性。
- 头像现在可以取回并缓存在服务器一侧。
- 为 shields.io 徽章添加了支持。

4.52.2 Weblate 1.8

发布于 2013 年 11 月 7 日。

- 关于升级的指示，请查看手册。
- 更好地列出了项目概况。
- 优化了分享选项的外观。
- 对匿名用户特权的更多控制。
- 支持使用第三方服务登录，请查阅手册了解详情。
- 用户可以通过电子邮箱地址而不是用户名登录。
- 文档改进。
- 改进了源字符串复查。
- 跨所有字符串的搜索。

- 更好地跟踪源字符串。
- 注册的 Captcha 保护。

4.52.3 Weblate 1.7

发布于 2013 年 10 月 7 日。

- 关于升级的指示，请查看手册。
- 支持检查 Python brace 格式字符串。
- 每个部件的质量检查定制。
- 具体的每个翻译的统计数据。
- 更改了将建议、检查与注释连接到字符串的方式。
- 用户现在可以添加文本来提交说明。
- 支持新语言请求的订阅。
- 支持添加新的翻译。
- 现在使用 Pillow 而不是 Pango+ Cairo 来生成小挂件和图表。
- 添加状态徽章小挂件。
- 去掉了无效的文本方向检查。
- 词典的变化现在会被记录在历史中。
- 翻译视图的性能改进。

4.52.4 Weblate 1.6

发布于 2013 年 7 月 25 日。

- 注册时更好的错误处理。
- 更改的浏览。
- 修复了机器翻译建议的排序。
- 改进了对 MyMemory 机器翻译的支持。
- 为 Amagam 机器翻译添加了支持。
- 频繁使用的页面上的各种优化。
- 在搜索结果中将搜索到的短语高亮。
- 支持存储消息时的自动修复。
- 跟踪翻译历史和选项来还原。
- 为 Google Translate API 添加了翻译。
- 为管理 SSH 主机密钥添加了支持。
- 各种表单验证改进。
- 各种质量检查改进。
- 导入的性能改进。
- 为建议投票添加了支持。
- 清理管理员界面。

4.52.5 Weblate 1.5

发布于 2013 年 4 月 16 日。

- 关于升级的指示，请查看手册。
- 添加了公共用户页面。
- 更好地命名复数形式。
- 为术语表的 TBX 导出添加了支持。
- 为 Bitbucket 通知添加了支持。
- 现在可以为每个翻译、语言或用户提供活动图表。
- 扩展了 import_project admin 命令的选项。
- 与 Django 1.5 兼容。
- 现在使用 libravatar 显示头像。
- 为 pretty print JSON 导出添加了可能性。
- 各种性能改进。
- 在项目或语言的进度条中也会显示未通过的检查或模糊字符串了。
- 为客户预提交投资与提交另外的文件添加了支持。
- 重写了搜索实现更好的性能与用户体验。
- 机器翻译的新界面。
- 添加了对单语言 po 文件的支持。
- 扩展了缓存的元数据的量，来改进各种搜索的速度。
- 现在也显示单词计数。

4.52.6 Weblate 1.4

发布于 2013 年 1 月 23 日。

- 修复了删除字符串时检查/评论的删除问题。
- 添加了选项来禁止自动传播翻译。
- 添加了选项为合并失败而订阅。
- 正确导入需要自定义 ttkit 加载器的项目。
- 添加了网站地图允许爬虫更容易地访问。
- 在通知电子邮件或 feed 中提供到字符串的直接链接。
- 对管理员界面的各种改进。
- 在管理员界面中为生产设置提供提示。
- 添加了每种语言的小挂件和参与页面。
- 改进了翻译锁定处理。
- 显示小挂件更多款式对应的代码片段。
- 在进度条中显示未通过的检查或模糊字符串。
- 用于将提交说明格式化的更多选项。
- 修复了机器翻译服务的错误处理。
- 改进了自动翻译锁定行为。

- 支持显示不同于之前的源字符串的更改。
- 为字串搜索添加了支持。
- 各种质量检查改进。
- 支持每个项目的 ACL。
- 基本代码由单元测试来覆盖。

4.52.7 Weblate 1.3

发布于 2012 年 11 月 16 日。

- 与 PostgreSQL 数据库后端兼容。
- 将上游 git 仓库中删除的语言也删除。
- 改进了质量检查处理。
- 添加了新的检查（BBCode、XML 标记和换行符）。
- 支持可选的变基而不是合并。
- 重新定位 Weblate 的可能性（例如在 /weblate 路径下运行）。
- 支持在自动检测失败的情况下手动选择文件类型。
- 更好地支持 Android 资源。
- 支持从 web 界面生成 SSH 密钥。
- 更多的可视化数据导出。
- 新的按钮来输入一些特殊字符。
- 支持导出字典。
- 支持锁定整个 Weblate 安装。
- 检查源字符串，并支持源字符串复查。
- 支持翻译和源字符串的用户评论。
- 更好地更改日志跟踪。
- 现在更改可以使用 RSS 监测。
- 改进了对 RTL 语言的支持。

4.52.8 Weblate 1.2

发布于 2012 年 8 月 14 日。

- Weblate 现在为数据库迁移使用 South，如果升级的话请检查升级的指示。
- 修复了连接 git repos 的小问题。
- 用于吸引人们使用 Weblate 参与翻译的全新介绍页面。
- 添加了可用于推广翻译项目的小挂件。
- 添加了选项将仓库重置为原始状态（对于有特权的用户）。
- 现在可以为翻译锁定项目和部件。
- 禁止一些翻译的可能性。
- 添加新翻译的可配置选项。
- 每个项目的 git 提交的配置。

- 简单的防垃圾电子邮件保护。
- 主页面更好的布局。
- 支持每次提交时自动推送更改。
- 支持译者电子邮件通知。
- 首选项中只列出使用的语言。
- 改进了当导入项目时对未知语言的处理。
- 支持译者锁定翻译。
- 可以选择维护 po 文件中的 `Language-Team` 标头。
- 在 about 页面中包括一些统计数据。
- 支持（并且需要）django-registration 0.8。
- 缓存带有未通过检查的字符串的数量。
- 在设置中检查要求。
- 文档改进。

4.52.9 Weblate 1.1

发布于 2012 年 7 月 4 日。

- 改进了几个翻译。
- 新建部件时更好的验证。
- 为在部件之间分享 git 仓库添加了支持。
- 不必每次尝试都提交来拉取远程 repo。
- 为卸载索引添加了支持。

4.52.10 Weblate 1.0

发布于 2012 年 5 月 10 日。

- 改进了添加/存储部件时的验证。
- 实验性支持 Android 组件文件（需要打补丁的 ttkit）。
- 来自钩子的更新在后台运行。
- 改进了安装指示。
- 改进了字典中的导航。

4.53 Weblate 0.x 系列

4.53.1 Weblate 0.9

发布于 2012 年 4 月 18 日。

- 修复了未知语言的导入。
- 改进了附近消息的列表。
- 改进了几项检查。
- 文档更新。

- 增加了另外几种语言的定义。
- 各种代码清理。
- 文档改进。
- 更改了文件布局。
- 将帮助脚本更新为 Django 1.4。
- 改进了翻译时的导航。
- 更好地处理 po 文件的重命名。
- 新建部件时更好的验证。
- 将完全设置集成到 syncdb 中。
- 在所有翻译页面中增加了近期变更列表。
- 检查未翻译的字符串，忽略格式字符串只检查消息。

4.53.2 Weblate 0.8

发布于 2012 年 4 月 3 日。

- 用 Whoosh 代替自己的全文搜索。
- 对检查的各种修复和改进。
- 新的命令更新检查。
- 很多翻译更新。
- 添加了字典，来存储最常用的术语。
- 为仓库状态的概况添加了 /admin/report/。
- 机器翻译服务不会再阻碍页面加载了。
- 管理界面现在也包含用于更新数据的有用操作。
- 记录用户所做更改的日志。
- 推至提交给 Git 的能力，来产生来自单一用户的更少的提交。
- 可以浏览未通过的检查。
- 使用已经翻译的字符串来自动翻译。
- 新的显示使用的版本的页面。
- Django 1.4 的兼容性。
- 从 web 界面将更改推送给远程 repo 的能力。
- 添加了对其他人完成的翻译进行复查。

4.53.3 Weblate 0.7

发布于 2012 年 2 月 16 日。

- 对 GitHub 通知的直接支持。
- 添加了对清洁孤立的检查与翻译的支持。
- 翻译时显示附近的字符串。
- 翻译时显示相似的字符串。
- 改进了字符串的搜索。

4.53.4 Weblate 0.6

发布于 2012 年 2 月 14 日。

- 添加了对翻译的消息的各种检查。
- 可调整的访问控制。
- 改进了对新一行翻译的处理。
- 添加了客户端一侧表单的排序。
- 升级时请查看升级说明。

4.53.5 Weblate 0.5

发布于 2012 年 2 月 12 日。

- **支持使用以下在线服务进行机器翻译：**
  - Apertium
  - 微软翻译
  - MyMemory
- 几个新的翻译。
- 改进了上游更改的合并。
- 更好地处理并发的 git 拉取与翻译。
- 传播那些同样用于模糊更改的工作。
- 传播那些同样用于文件上传的工作。
- 修复了使用 FastCGI（也可能是其它）时的文件下载。

4.53.6 Weblate 0.4

发布于 2012 年 2 月 8 日。

- 将使用向导添加到文档中。
- 修复了不需要 CSRF 保护的 API 钩子。

4.53.7 Weblate 0.3

发布于 2012 年 2 月 8 日。

- 优化复数翻译原文的显示。
- Sphinx 格式的新文档。
- 翻译时显示第二语言。
- 改进了错误页面，来给出现有项目的列表。
- 新的每种语言的统计数据。

4.53.8 Weblate 0.2

发布于 2012 年 2 月 7 日。

- 改进了几种形式的验证。
- 在个人资料升级时警告用户。
- 记住 URL 用于登录。
- 输入复数形式时对文本区域的命名。
- 自动扩大翻译区域。

4.53.9 Weblate 0.1

发布于 2012 年 2 月 6 日。

- 初始发布。

Python 模块索引

W

HTTP Routing Table

/

/api

/api/addons

/api/changes

/api/component-lists

/api/components

/api/groups

/api/languages

/api/memory

/api/metrics

/api/projects

/api/roles

/api/screenshots

/api/tasks

/api/translations

/api/units

/api/users

/exports

/hooks

索引

符号

A

B

C

D

E

F

M

N

P

Q

R

S

T

U

V

W