The Weblate Manual

发布 *4.4.1*

Michal Čihař

2021 年 01 月 13 日

User docs

CHAPTER 1

User docs

1.1 Weblate 基础知识

1.1.1 项目和组件架构

在 Weblate 中，翻译组织成为项目和组件。每个项目可以包含几个组件，并且组件包含各个语言的翻译。组件相应于一个翻译文件（例如*GNU gettext* 或*Android string resources*）。项目帮助您将组件组织为逻辑的组（例如，将一个应用中使用的所有翻译分组）。

默认情况下，每个项目内都有对跨组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 通过组件配置 禁用翻译传播。

参见:

../devel/integration

1.2 Registration and user profile

1.2.1 注册

Everybody can browse projects, view translations or suggest translations by default. Only registered users are allowed to actually save changes, and are credited for every translation made.

You can register by following a few simple steps:

1. Fill out the registration form with your credentials.
2. Activate registration by following the link in the e-mail you receive.
3. Optionally adjust your profile to choose which languages you know.

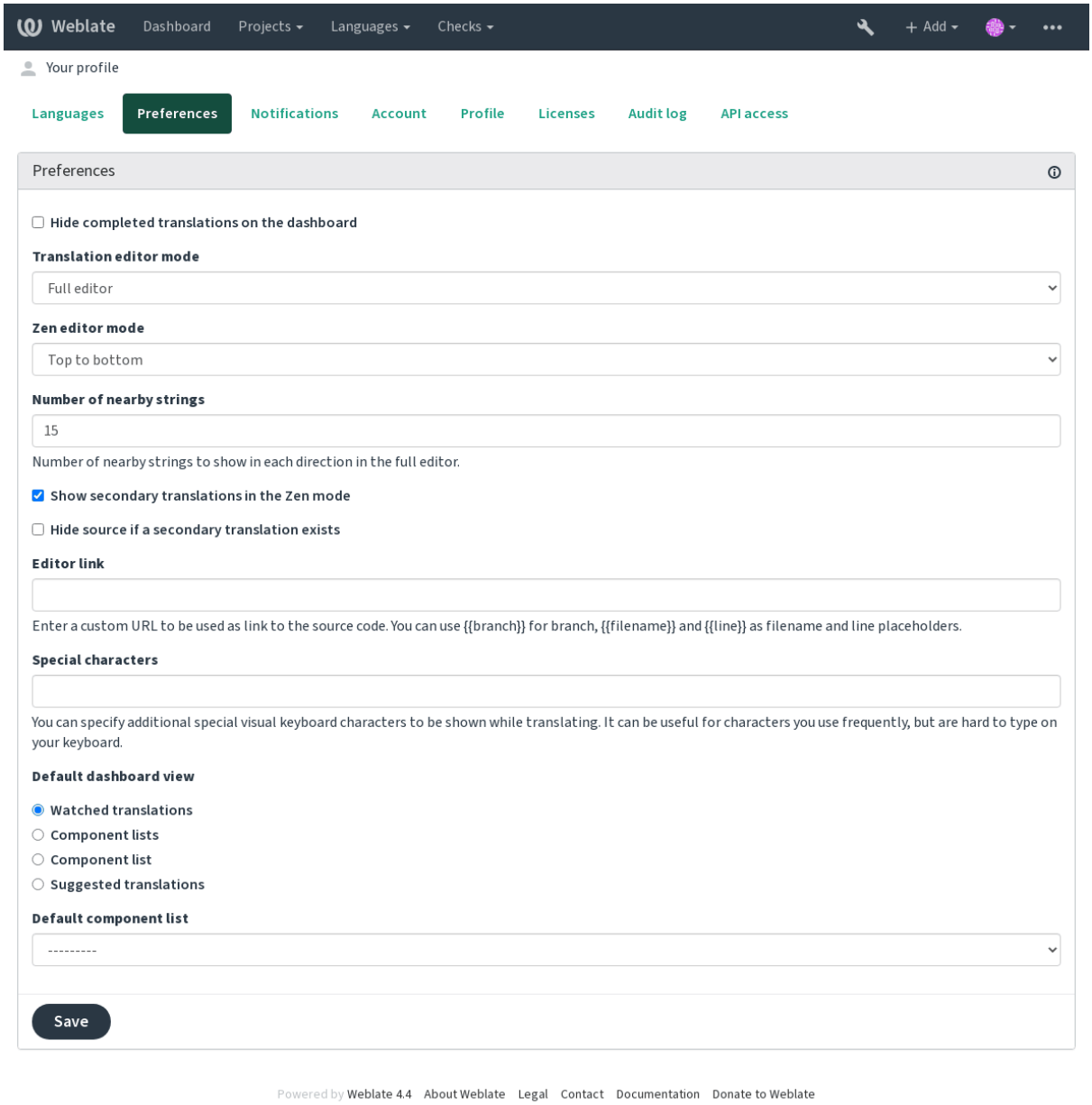
1.2.2 控制面板

When you sign in, you will see an overview of projects and components, as well as their respective translation progression.

2.5 新版功能.

Components of projects you are watching are shown by default, and cross-referenced with your preferred languages.

提示： You can switch to different views using the navigation tabs.



The menu has these options:

- *Projects* > *Browse all projects* in the main menu showing translation status for each project on the Weblate instance.
- Selecting a language in the main menu *Languages* will show translation status of all projects, filtered by one of your primary languages.

- *Watched translations* in the Dashboard will show translation status of only those projects you are watching, filtered by your primary languages.

In addition, the drop-down can also show any number of *component lists*, sets of project components preconfigured by the Weblate administrator, see 组件列表.

You can configure your personal default dashboard view in the *Preferences* section of your user profile settings.

注解： When Weblate is configured for a single project using `SINGLE_PROJECT` in the `settings.py` file (see 配置), the dashboard will not be shown, as the user will be redirected to a single project or component instead.

1.2.3 用户个人资料

The user profile is accessible by clicking your user icon in the top-right of the top menu, then the *Settings* menu.

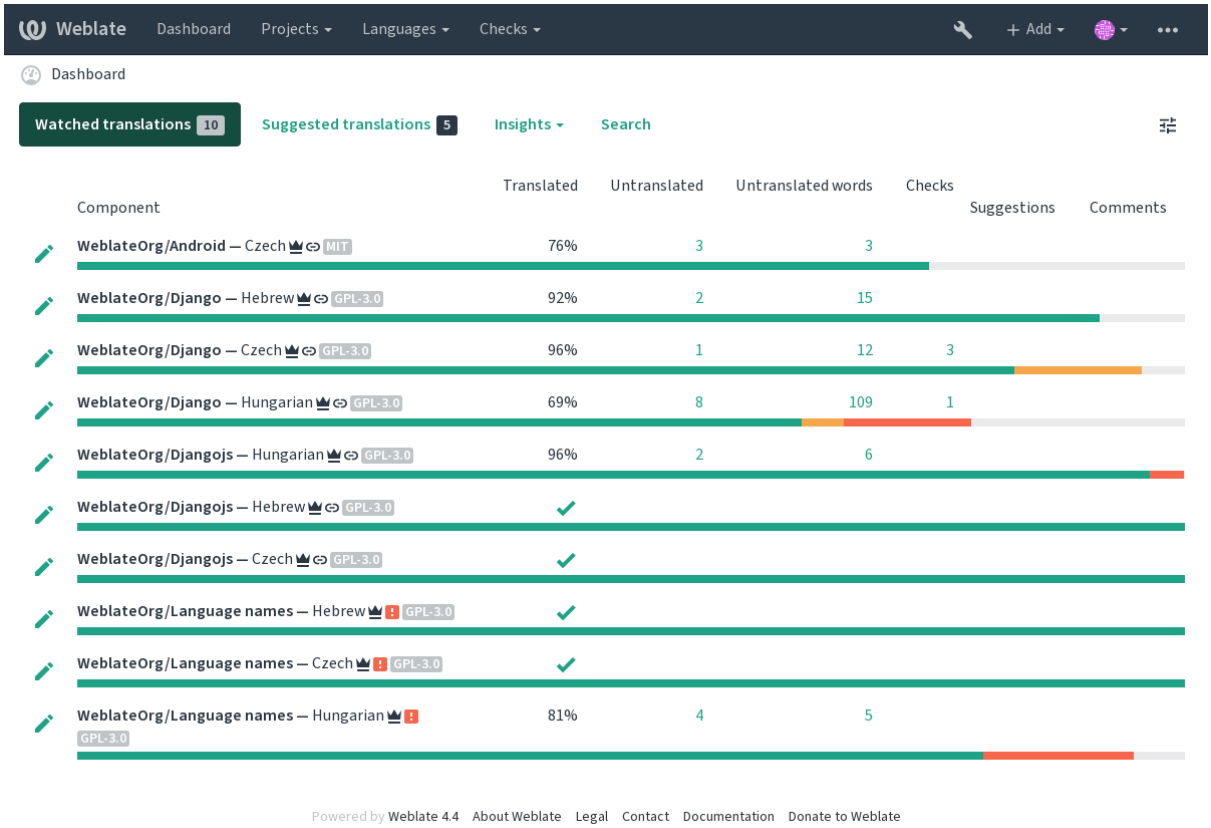
The user profile contains your preferences. Name and e-mail address is used in VCS commits, so keep this info accurate.

注解： All language selections only offer currently translated languages.

提示： Request or add other languages you want to translate by clicking the button to make them available too.

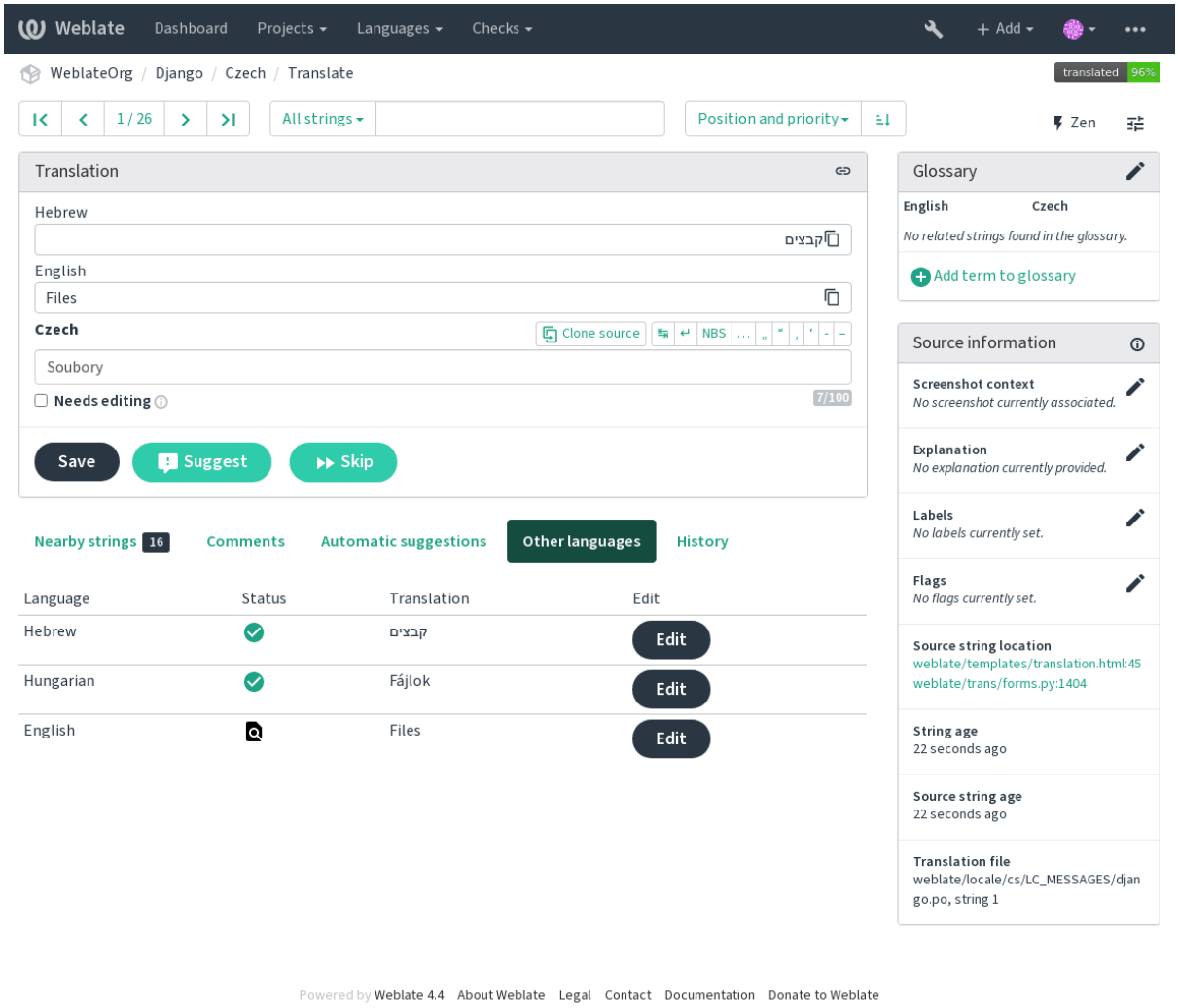
已翻译的语言

Choose which languages you prefer to translate, and they will be offered on the main page of watched projects, so that you have easier access to these all translations in each of those languages.



第二语言

You can define which secondary languages are shown to you as a guide while translating. An example can be seen in the following image, where the Hebrew language is shown as secondarily:



控制面板默认界面

On the *Preferences* tab, you can pick which of the available dashboard views to present by default. If you pick the *Component list*, you have to select which component list will be displayed from the *Default component list* drop-down.

参见:

组件列表

公开资料

本页的所有字段都是可选的，并且可以在任何时间删除，通过填写这些字段，您同意我们在您的用户资料出现的任何地方分享这些数据。

Avatar can be shown for each user (depending on *ENABLE_AVATARS*). These images are obtained using https://gravatar.com/.

编辑者链接

A source code link is shown in the web-browser configured in the 组件配置 by default.

提示: By setting the *Editor link*, you use your local editor to open the VCS source code file of translated strings. You can use 模板标记.

Usually something like `editor://open/?file={{filename}}&line={{line}}` is a good option.

参见:

You can find more info on registering custom URL protocols for the editor in the Nette documentation.

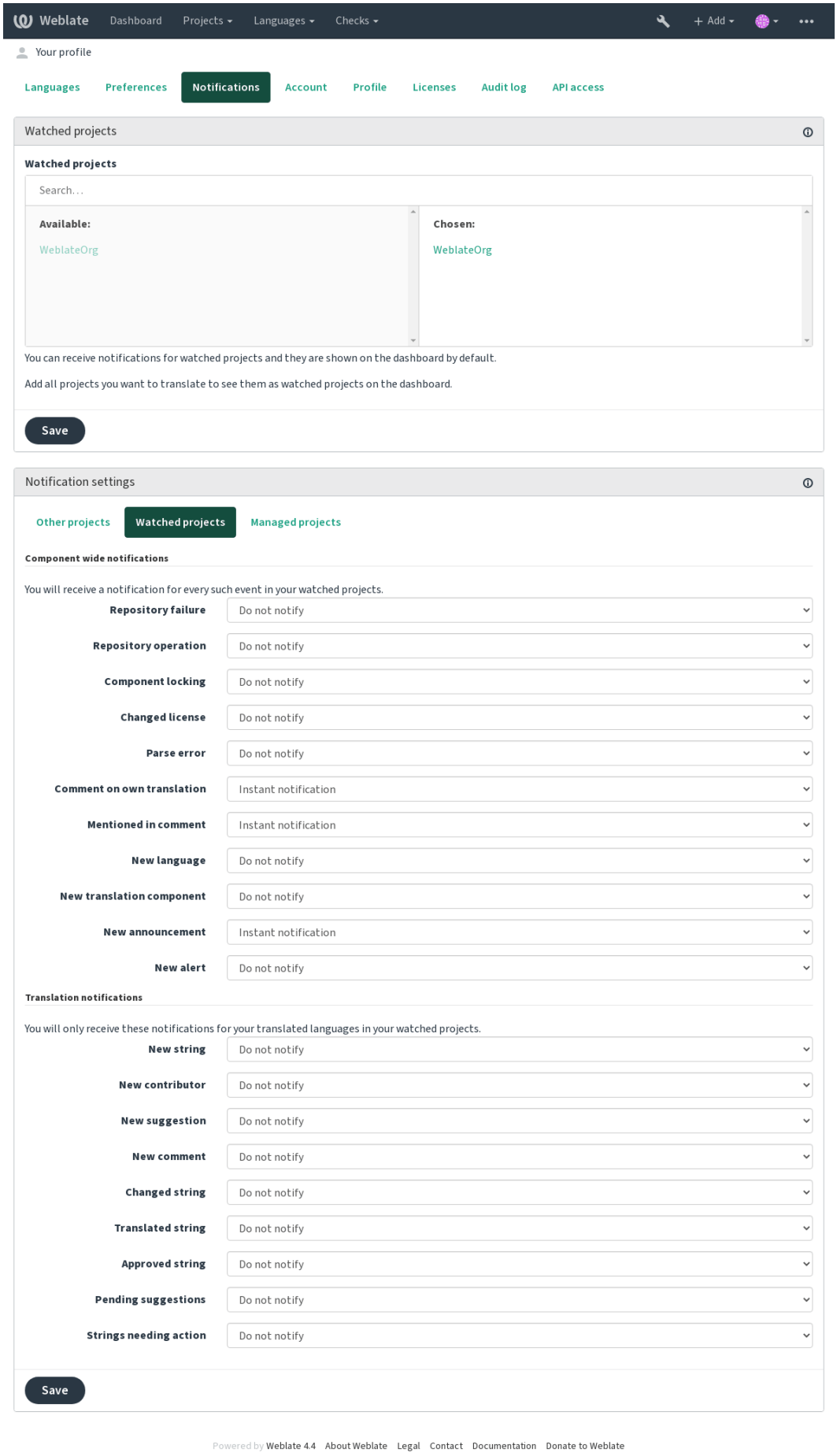
1.2.4 通知

Subscribe to various notifications from the *Notifications* tab. Notifications for selected events on watched or administered projects will be sent to you per e-mail.

Some of the notifications are sent only for events in your languages (for example about new strings to translate), while some trigger at component level (for example merge errors). These two groups of notifications are visually separated in the settings.

You can toggle notifications for watched projects and administered projects and it can be further tweaked (or muted) per project and component. Visit the component overview page and select appropriate choice from the *Watching* menu.

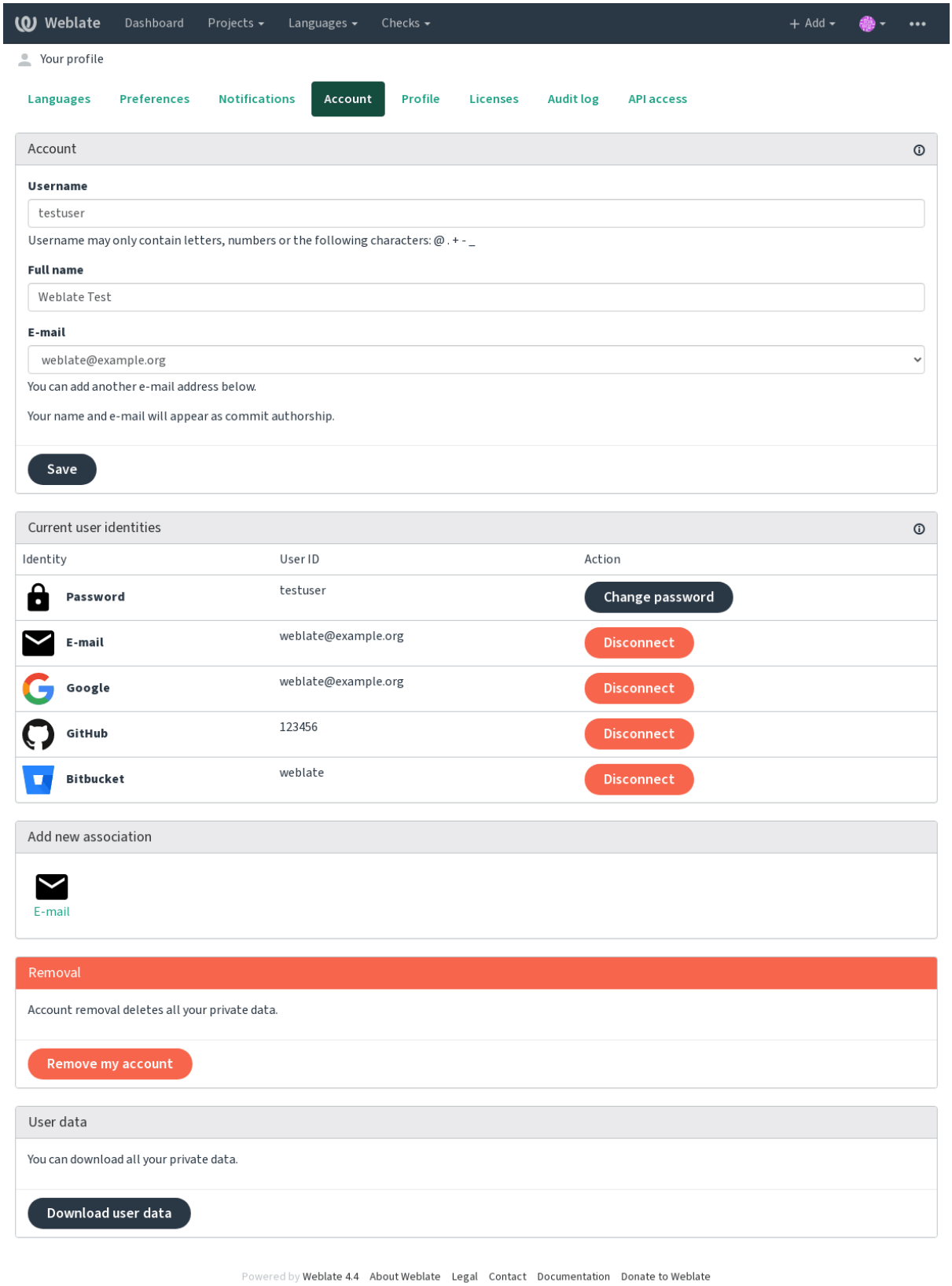
注解: You will not receive notifications for your own actions.


Weblate
Dashboard
Projects
Languages
Checks
Add
Your profile
Languages
Preferences
Notifications
Account
Profile
Licenses
Audit log
API access
Watched projects
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
Strings needing action
Save
Powered by Weblate 4.4
About Weblate
Legal
Contact
Documentation
Donate to Weblate

1.2.5 账户

The *Account* tab lets you set up basic account details, connect various services you can use to sign in into Weblate, completely remove your account, or download your user data (see *Weblate 用户数据导出*).

注解： The list of services depends on your Weblate configuration, but can be made to include popular sites such as GitLab, GitHub, Google, Facebook, or Bitbucket or other OAuth 2.0 providers.


Weblate
Dashboard
Projects
Languages
Checks
Add
Your profile
Languages
Preferences
Notifications
Account
Profile
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.
Your name and e-mail will appear as commit authorship.
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 4.4
About Weblate
Legal
Contact
Documentation
Donate to Weblate

1.2.6 API 访问

You can get or reset your API access token here.

1.2.7 审计日志

Audit log keeps track of the actions performed with your account. It logs IP address and browser for every important action with your account. The critical actions also trigger a notification to a primary e-mail address.

参见:

在反向代理后面运行

1.3 Translating using Weblate

Thank you for interest in translating using Weblate. Projects can either be set up for direct translation, or by way of accepting suggestions made by users without accounts.

Overall, there are two modes of translation:

- The project accepts direct translations
- The project accepts only suggestions, which are automatically validated once a defined number of votes is reached

Please see 翻译工作流 for more information on translation workflow.

Options for translation project visibility:

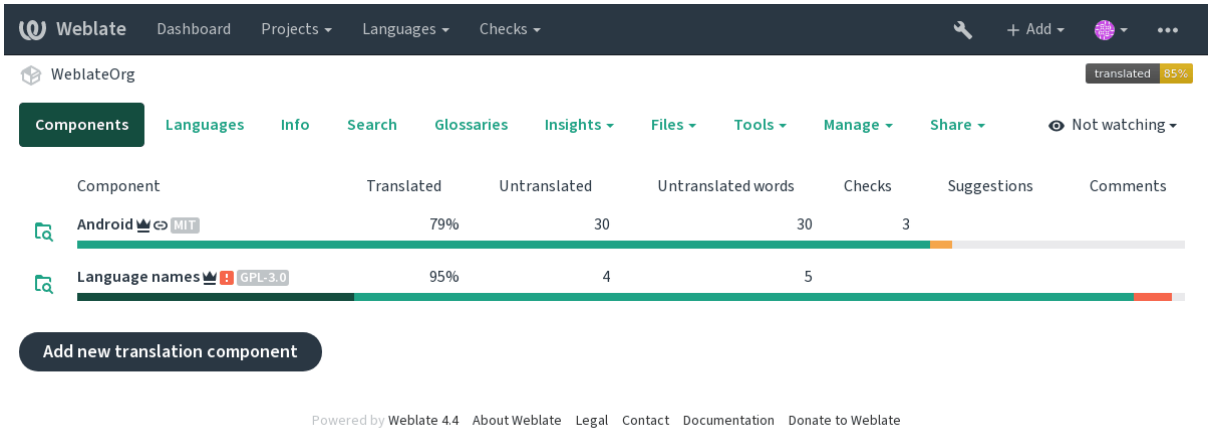
- Publicly visible and anybody can contribute
- Visible only to a certain group of translators

参见:

访问控制, 翻译工作流

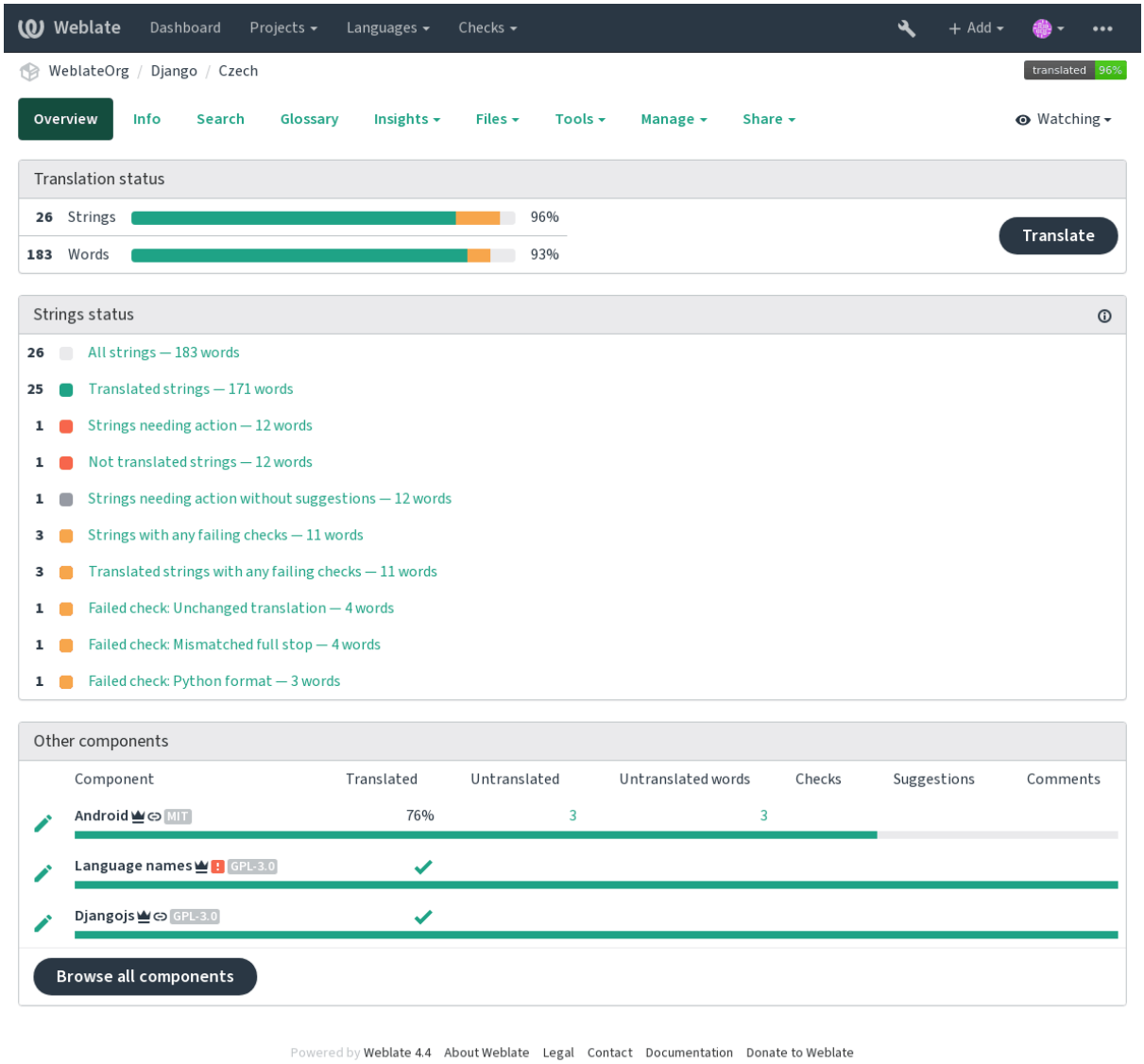
1.3.1 翻译项目

Translation projects hold related components, related to the same software, book, or project.



1.3.2 Translation links

Having navigated to a component, a set of links lead to actual translation. The translation is further divided into individual checks, like *Not translated strings* or *Strings needing action*. If the whole project is translated, without error, *All strings* is still available. Alternatively you can use the search field to find a specific string or term.



1.3.3 建议

注解： Actual permissions might vary depending on your Weblate configuration.

Anonymous users can only (if permitted) forward suggestions. Doing so is still available to signed in users, in cases where uncertainty about the translation arises, which will prompt another translator to review it.

The suggestions are scanned on a daily basis to remove duplicate ones or suggestions that match the current translation.

1.3.4 注释

The comments can be posted in two scopes - source string or translation. Choose the one which matches the topic you want to discuss. The source string comments are good for providing feedback on the original string, for example that it should be rephrased or it is confusing.

You can use Markdown syntax in the comments and mention other users using `@mention`.

参见:

report-source

1.3.5 变体

Variants are used to group variants of the string in different lengths. The frontend can use different strings depending on the screen or window size.

参见:

variants

1.3.6 标签

Labels are used to categorize strings within a project. These can be used to further customize the localization workflow, for example to define categories of strings.

参见:

labels

1.3.7 翻译

On the translation page, the source string and an edit area for translating are shown. Should the translation be plural, multiple source strings and edit areas are shown, each described and labeled in plural form.

All special whitespace characters are underlined in red and indicated with grey symbols. More than one subsequent space is also underlined in red to alert the translator to a potential formatting issue.

Various bits of extra information can be shown on this page, most of which coming from the project source code (like context, comments or where the message is being used). When you choose secondary languages in your preferences, translation to these languages will be shown (see 第二语言) above the source string.

Below the translation, any suggestion made by others will be shown, which you can in turn accept, accept with changes, or delete.

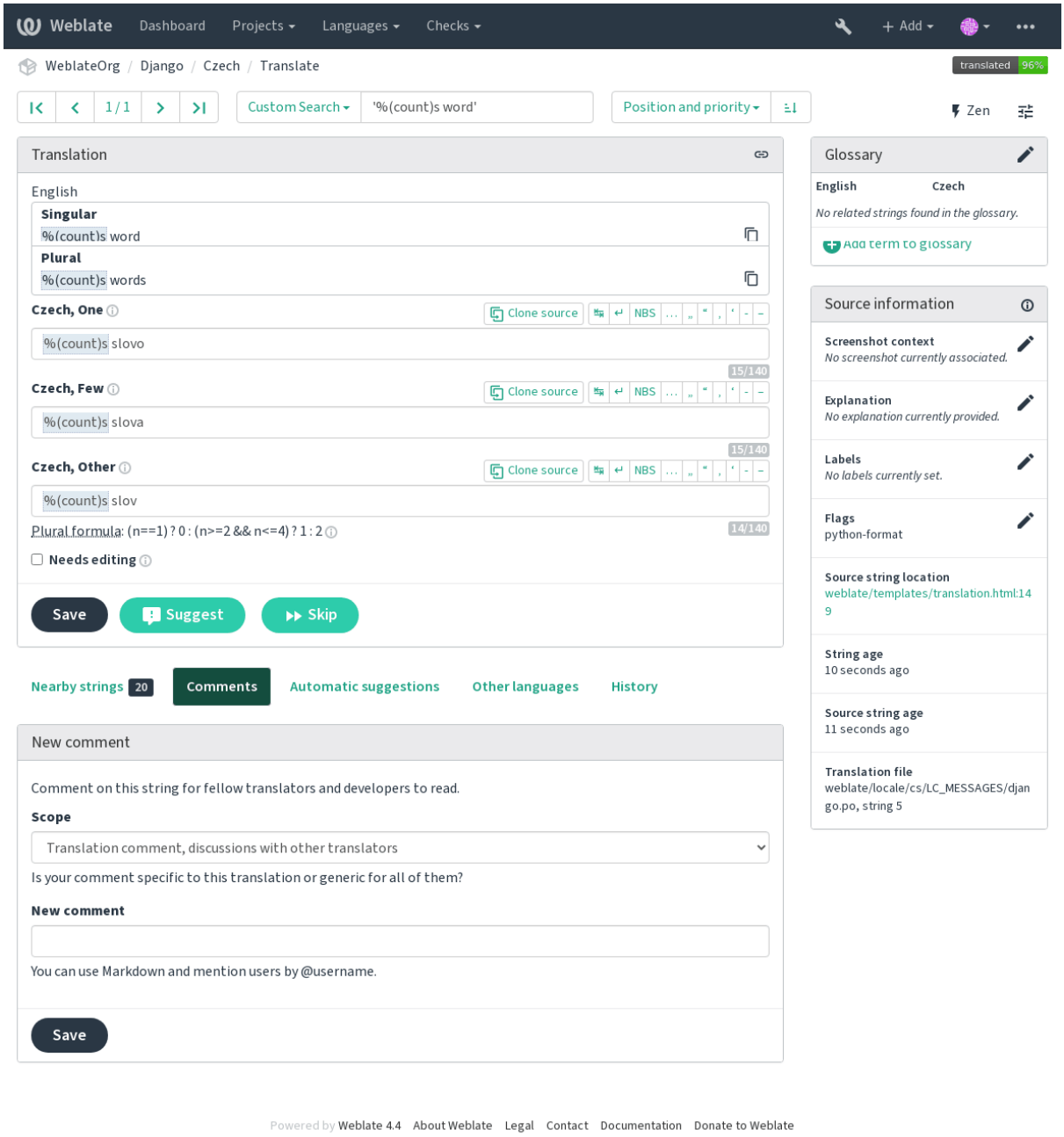
复数形式

Words that change form to account of their numeric designation are called plurals. Each language has its own definition of plurals. English, for example, supports one plural. In the singular definition of for example "car", implicitly one car is referenced, in the plural definition, "cars" two or more cars are referenced, or the concept of cars as a noun. Languages like for example Czech or Arabic have more plurals and also their rules for plurals are different.

Weblate has full support for each of these forms, in each respective language by translating every plural separately. The number of fields and how it is used in the translated application depends on the configured plural formula. Weblate shows the basic information, but you can find a more detailed description in the Language Plural Rules by the Unicode Consortium.

参见:

复数式



Keyboard shortcuts

在 2.18 版更改: The keyboard shortcuts have been revamped in 2.18 to less likely collide with browser or system defaults.

The following keyboard shortcuts can be utilized during translation:

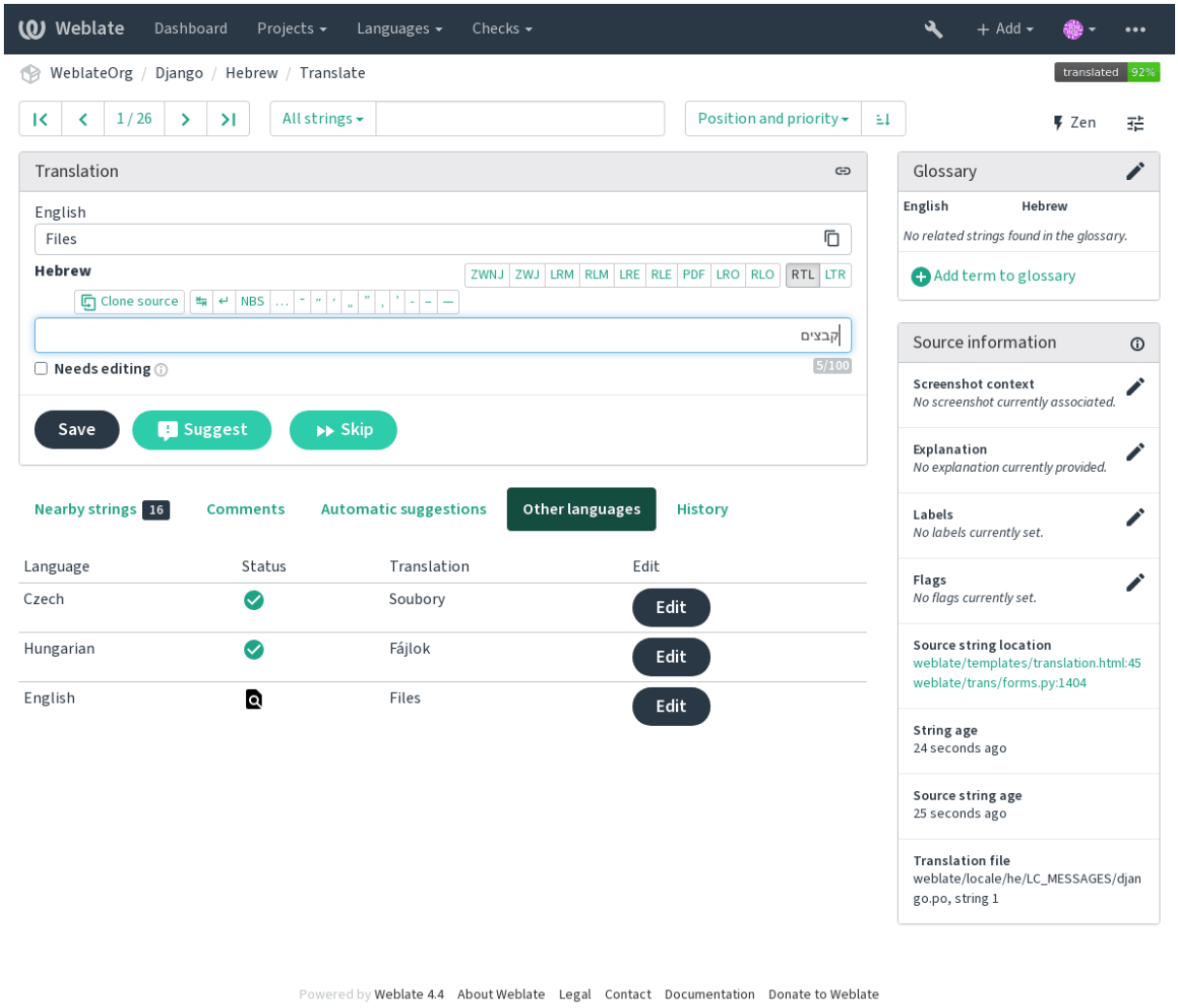
键盘快捷键	描述
`Alt Home`	导航到当前搜索中的第一个翻译。
`Alt Home`	导航到当前搜索中的第一个翻译。
`Alt End`	导航到当前搜索中的最后一个翻译。
`Alt PageUp` or `Ctrl ↑` or `Alt ↑` or `Cmd ↑`	导航到当前搜索中的前一处翻译。
`Alt PageDown` or `Ctrl ↓` or `Alt ↓` or `Cmd ↓`	导航到当前搜索中的下一处翻译。
`Alt Enter` or `Ctrl Enter` or `Cmd Enter`	保存当前翻译。
`Ctrl Shift Enter` or `Cmd Shift Enter`	取消标记翻译为“需要编辑”并提交它。
`Ctrl E` or `Cmd E`	Focus translation editor.
`Ctrl U` or `Cmd U`	Focus comment editor.
`Ctrl M` or `Cmd M`	显示 *Automatic suggestions* 标签，请参见自动建议。
`Ctrl 1` to `Ctrl 9` or `Cmd 1` to `Cmd 9`	Copies placeable of given number from source string.
`Ctrl M 1` to `9` or `Cmd M 1` to `9`	Copy the machine translation of given number to current translation.
`Ctrl I 1` to `9` or `Cmd I 1` to `9`	忽略失败检查列表中的一个项目。
`Ctrl J` or `Cmd J`	显示:guilabel: ‘附近字符串 ‘选项卡。
`Ctrl S` or `Cmd S`	Focuses search field.
`Ctrl O` or `Cmd O`	Copies source string.
`Ctrl Y` or `Cmd Y`	Toggles the *Needs editing* flag.

Visual keyboard

A small visual keyboard is shown just above the translation field. This can be useful for typing characters not usually found or otherwise hard to type.

The shown symbols factor into three categories:

- User configured characters defined in the 用户个人资料
- Per-language characters provided by Weblate (e.g. quotes or RTL specific characters)
- Characters configured using `SPECIAL_CHARS`



Translation context

This contextual description provides related information about the current string.

String attributes Things like message ID, context (`msgctxt`) or location in source code.

截图 Screenshots can be uploaded to Weblate to better inform translators of where and how the string is used, see 字符串的可见语境.

附近字符串 Displays neighbouring messages from the translation file. These are usually also used in a similar context and prove useful in keeping the translation consistent.

其它的出现位置 In case a message appears in multiple places (e.g. multiple components), this tab shows all of them if they are found to be inconsistent (see 不一致的). You can choose which one to use.

翻译记忆库 Look at similar strings translated in past, see Memory Management.

词汇表 Displays terms from the project glossary used in the current message.

最近的变更 List of people whom have changed this message recently using Weblate.

项目 Project information like instructions for translators, or information about its version control system repository.

If the translation format supports it, you can also follow supplied links to respective source code containing each source string.

Translation history

Every change is by default (unless turned off in component settings) saved in the database, and can be reverted. Optionally one can still also revert anything in the underlying version control system.

Translated string length

Weblate can limit length of translation in several ways to ensure the translated string is not too long:

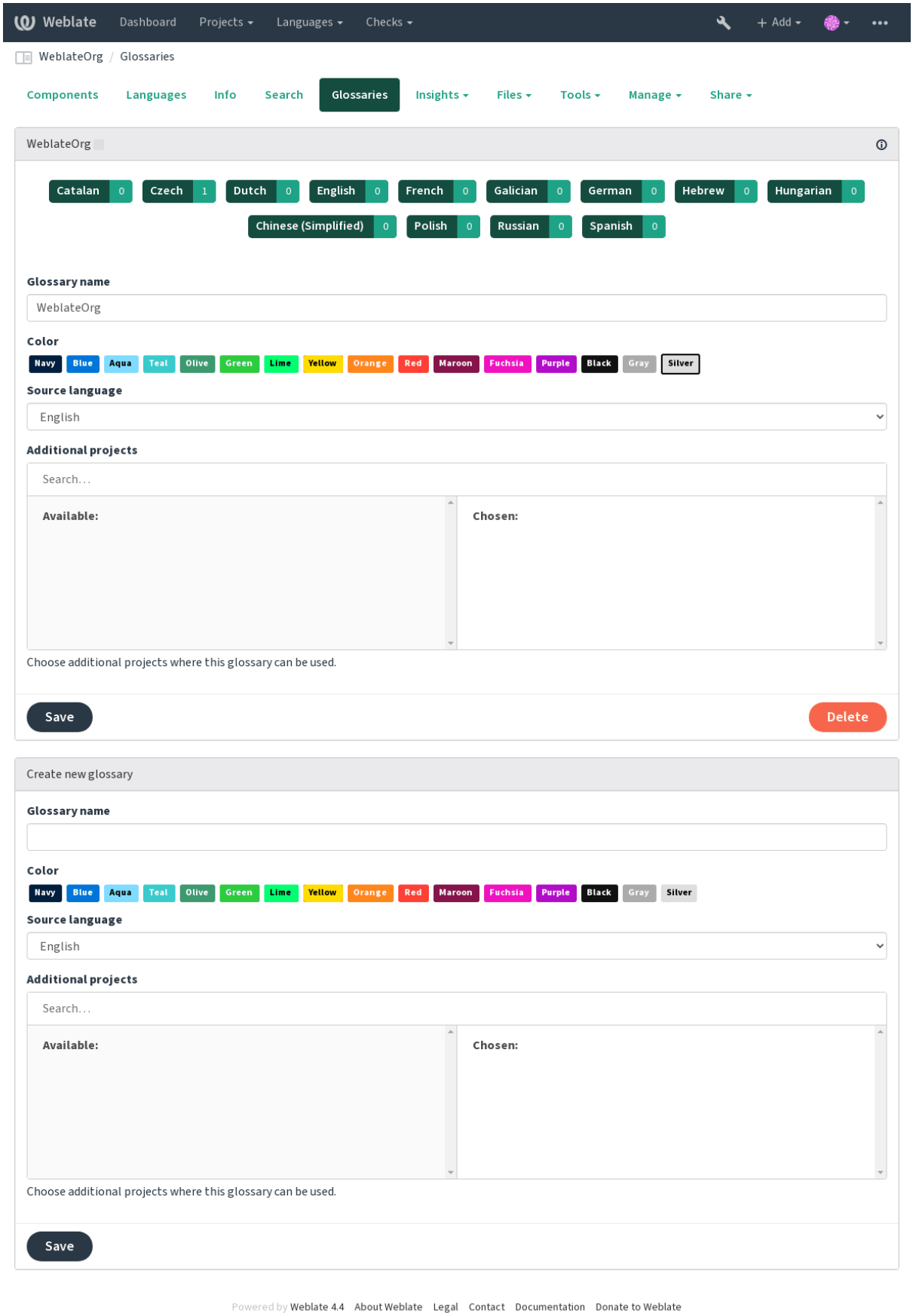
- The default limitation for translation is ten times longer than source string. This can be turned off by `LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH`. In case you are hitting this, it might be also caused by monolingual translation being configured as bilingual, making Weblate see translation key as source string instead of the actual source string. See 双语和单语格式 for more info.
- Maximal length in characters defined by translation file or flag, see 译文最大长度.
- Maximal rendered size in pixels defined by flags, see 最大翻译大小.

1.3.8 词汇表

Each project can have an assigned glossary for any language as a shorthand for storing terminology. Consistency is more easily maintained this way. Terms from the currently translated string can be displayed in the bottom tabs.

Managing glossaries

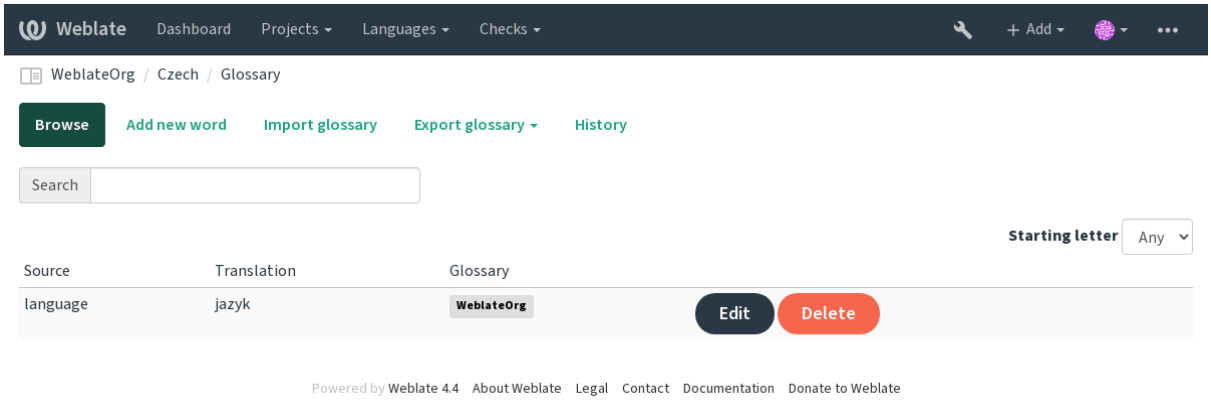
On the *Glossaries* tab of each project page, you can edit existing glossaries.



An empty glossary for a given project is automatically created when project is created. Glossaries are shared among all components of the same project and you can also choose to share them with another projects. You can do this

only for projects you can administer.

On this list, you can choose which glossary to manage (all languages used in the current project are shown). Following one of the language links will lead you to a page which can be used to edit, import or export the selected glossary, or view the edit history:



1.3.9 自动建议

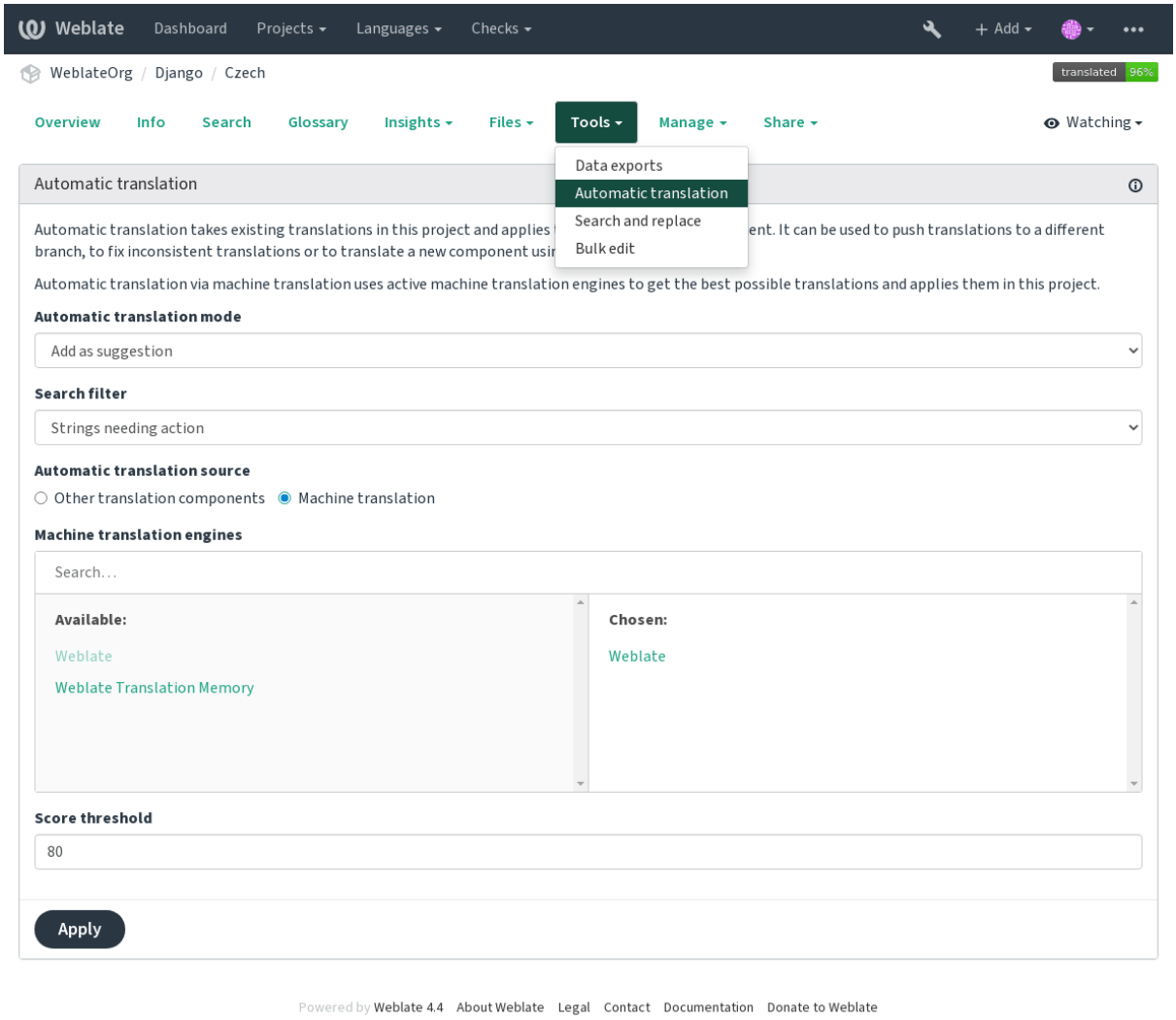
Based on configuration and your translated language, Weblate provides you suggestions from several machine translation tools and 翻译记忆库. All machine translations are available in a single tab of each translation page.

参见:

You can find the list of supported tools in 机器翻译.

1.3.10 自动化翻译

You can use automatic translation to bootstrap translation based on external sources. This tool is called *Automatic translation* accessible in the *Tools* menu, once you have selected a component and a language:



Two modes of operation are possible:

- Using other Weblate components as a source for translations.
- Using selected machine translation services with translations above a certain quality threshold.

You can also choose which strings are to be auto-translated.

> **警告：** Be mindful that this will overwrite existing translations if employed with wide filters such as *All strings*.

Useful in several situations like consolidating translation between different components (for example website and application) or when bootstrapping translation for a new component using existing translations (translation memory).

参见：

跨组件保持翻译一致

1.3.11 频次限制

To avoid abuse of the interface, there is rate limiting applied to several operations like searching, sending contact form or translating. In case you are hit by this, you are blocked for a certain period until you can perform the operation again.

The default limits are described in the administrative manual in 频次限制, but can be tweaked by configuration.

1.3.12 搜索并替换

In case you want to change a terminology or perform some bulk fixing of the strings, *Search and replace* is a feature for you. You can find it in the *Tools* menu.

提示: Don't worry about messing up the strings. This is a two step process which will show you a preview of the edits before the actual change is done.

1.3.13 批量编辑

Bulk edit allows you to perform operation on number of strings. You define search strings and operation to perform and all matching strings are updated. Following operations are supported:

- Changing string state (for example to approve all strings waiting for review)
- Adjust translation flags (see 定制行为)
- Adjust string labels (see labels)

提示: This tool is called *Bulk edit* accessible in the *Tools* menu for each project, component or translation.

参见:

Bulk edit addon

1.4 Downloading and uploading translations

You can export files from a translation, make changes, and import them again. This allows working offline, and then merging changes back into the existing translation. This works even if it has been changed in the meantime.

注解: The available options might be limited by 访问控制.

1.4.1 Downloading translations

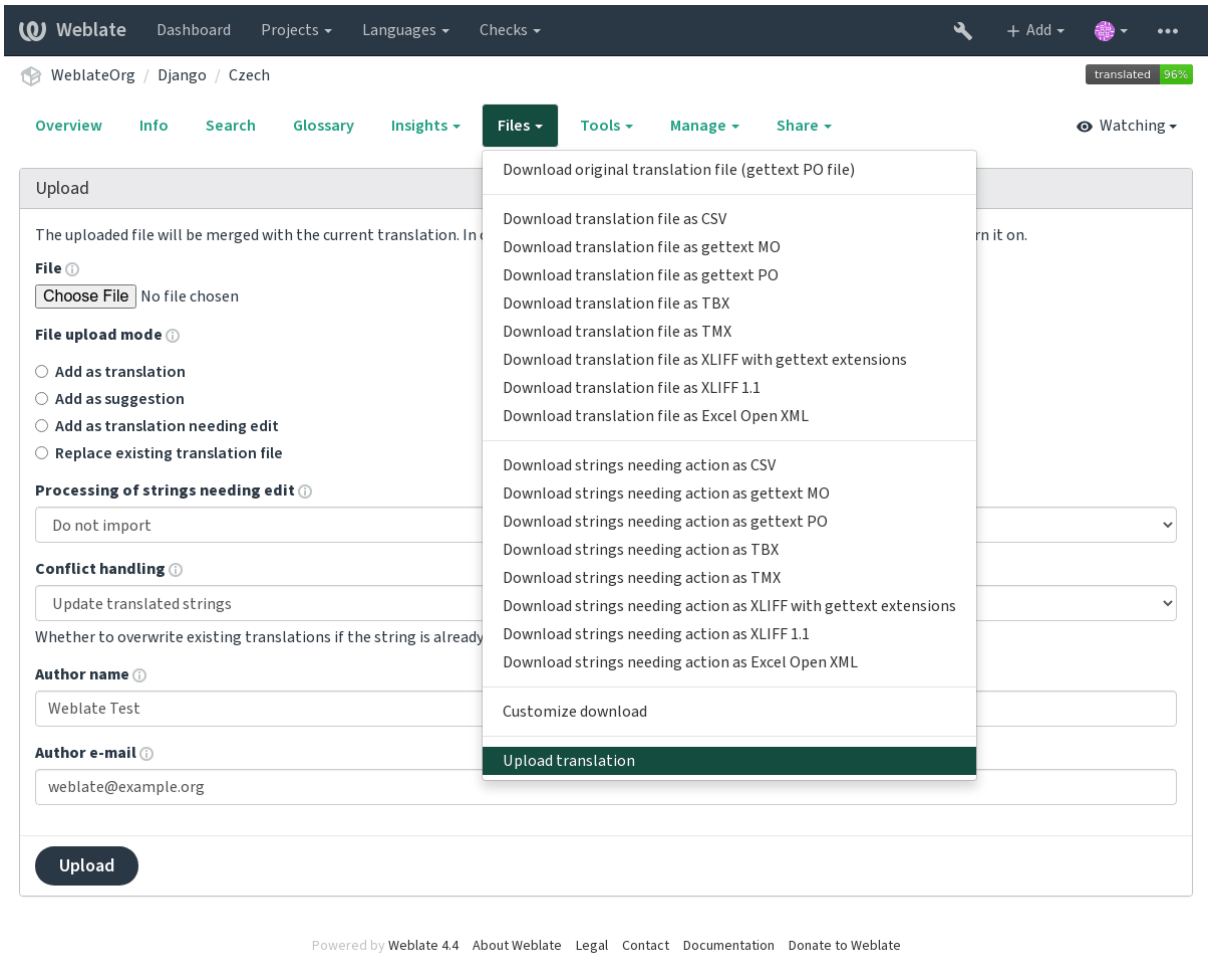
From the project or component dashboard, translatable files can be downloaded using the *Download original translation file* in the *Files* menu, producing a copy of the original file as it is stored in the upstream Version Control System.

You can also download the translation converted into one of widely used localization formats. The converted files will be enriched with data provided in Weblate such as additional context, comments or flags.

Several file formats are available, including a compiled file to use in your choice of application (for example `.mo` files for GNU Gettext) using the *Files* menu.

1.4.2 Uploading translations

When you have made your changes, use *Upload translation* in the *Files* menu.



支持的文件格式

Any file in a supported file format can be uploaded, but it is still recommended to use the same file format as the one used for translation, otherwise some features might not be translated properly.

参见:

支持的文件格式

The uploaded file is merged to update the translation, overwriting existing entries by default (this can be turned off or on in the upload dialog).

Import methods

These are the choices presented when uploading translation files:

添加为翻译 (`translate`) Imported translations are added as translations. This is the most common usecase, and the default behavior.

添加为建议 (`suggest`) Imported translations are added as suggestions, do this when you want to have your uploaded strings reviewed.

添加为需要编辑的翻译 (`fuzzy`) Imported translations are added as translations needing edit. This can be useful when you want translations to be used, but also reviewed.

替换现有翻译文件 (`replace`) Existing file is replaced with new content. This can lead to loss of existing translations, use with caution.

更新源字符串 (`source`) Updates source strings in bilingual translation file. This is similar to what *更新 PO 文件以匹配 POT 文件 (msgmerge)* does.

参见:

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

Conflicts handling

Defines how to deal with uploaded strings which are already translated.

Strings needing edit

There is also an option for how to handle strings needing edit in the imported file. Such strings can be handle in one of the three following ways: "Do not import", "Import as string needing edit", or "Import as translated".

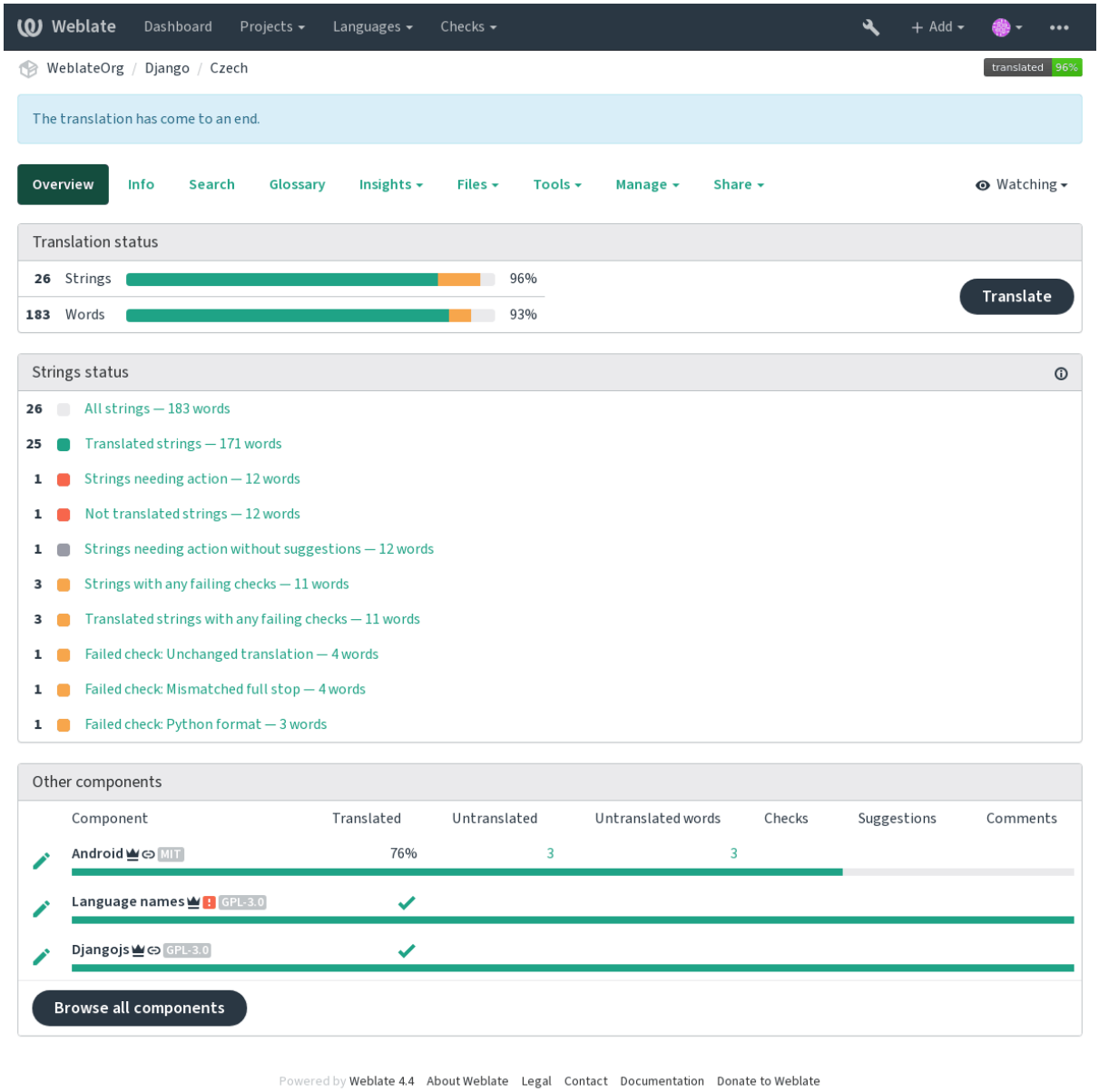
Overriding authorship

With admin permissions, you can also specify authorship of uploaded file. This can be useful in case you've received the file in another way and want to merge it into existing translations while properly crediting the actual author.

1.5 检查并修正

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。

提交未通过检查的翻译后，将立即向用户显示：



1.5.1 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

1.5.2 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, *定制行为*

1.5.3 翻译检查

在每次翻译更改时执行，帮助翻译人员保持高质量的翻译。

BBcode 标记

翻译中的 BBcode 与来源不符

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。

注解: 当前检测 BBcode 的方法非常简单，因此此检查可能会产生误报。

连续重复的单词

文本连续两次包含相同的单词:

4.1 新版功能.

检查翻译中是否没有连续重复的单词。这通常表示翻译错误。

提示: 此检查包括特定于语言的规则，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*。

双空格

翻译包含双空格

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。

当在源中找到双空格时，检查为假，这意味着故意使用双空格。

格式化字符串

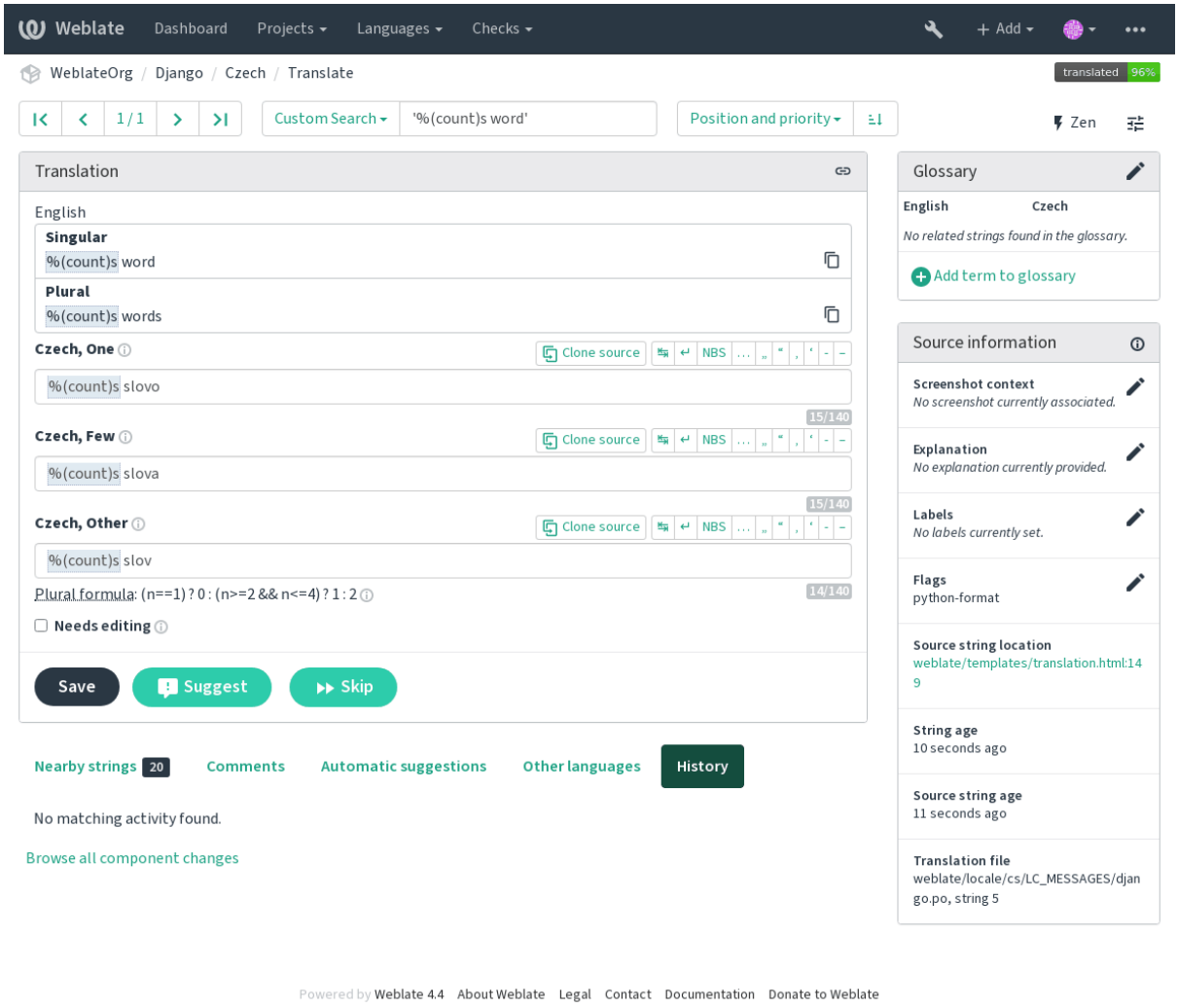
检查字符串格式是否在源和翻译之间复制。在翻译中省略格式字符串通常会导致严重的问题，因此字符串中的格式通常应与源匹配。

Weblate 支持检查几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*组件配置* 中完成此操作。为每个组件定义它比较简单，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

AngularJS interpolation strings do not match source

Named format string	`Your balance is {{amount}} {{ currency }}`
Flag to enable	*angularjs-format*

参见:

AngularJS: API: $interpolate

C 格式

C format string does not match source

Simple format string	`There are %d apples`
Position format string	`Your balance is %1$d %2$s`
Flag to enable	*c-format*

参见:

C format strings, C printf format

C# 格式

C# format string does not match source

Position format string	`There are {0} apples`
Flag to enable	*c-sharp-format*

参见:

C# String Format

ECMAScript 模板文字

ECMAScript 模板文字与源不匹配

Interpolation	`There are ${number} apples`
Flag to enable	*es-format*

参见:

Template literals

i18next 插补

The i18next interpolation does not match source

4.0 新版功能.

Interpolation	`There are {{number}} apples`
Nesting	`There are $t(number) apples`
Flag to enable	*i18next-interpolation*

参见:

i18next interpolation

Java 格式

Java format string does not match source

Simple format string	`There are %d apples`
Position format string	`Your balance is %1$d %2$s`
Flag to enable	*java-format*

参见:

Java Format Strings

Java MessageFormat

Java MessageFormat string does not match source

Position format string	`There are {0} apples`
Flag to enable	*java-messageformat* enables the check unconditionally
	auto-java-messageformat enables check only if there is a format string in the source

参见:

Java MessageFormat

JavaScript 格式

JavaScript format string does not match source

Simple format string	`There are %d apples`
Flag to enable	*javascript-format*

参见:

JavaScript formatting strings

百分比占位符

The percent placeholders do not match source

4.0 新版功能.

Simple format string	`There are %number% apples`
Flag to enable	*percent-placeholders*

Perl 格式

Perl format string does not match source

Simple format string	`There are %d apples`
Position format string	`Your balance is %1$d %2$s`
Flag to enable	*perl-format*

参见:

Perl sprintf, Perl Format Strings

PHP 格式

PHP format string does not match source

Simple format string	`There are %d apples`
Position format string	`Your balance is %1$d %2$s`
Flag to enable	*php-format*

参见:

PHP sprintf documentation, PHP Format Strings

Python brace 格式

Python brace format string does not match source

Simple format string	`There are {} apples`
Named format string	`Your balance is {amount} {currency}`
Flag to enable	*python-brace-format*

参见:

Python brace format, Python Format Strings

Python 格式

Python format string does not match source

Simple format string	`There are %d apples`
Named format string	`Your balance is %(amount) %(currency)`
Flag to enable	*python-format*

参见:

Python string formatting, Python Format Strings

Qt 格式

Qt format string does not match source

Position format string	`There are %1 apples`
Flag to enable	*qt-format*

参见:

Qt QString::arg()

Qt 复数格式

Qt plural format string does not match source

Plural format string	`There are %Ln apple(s)`
Flag to enable	*qt-plural-format*

参见:

Qt i18n guide

Ruby 格式

Ruby format string does not match source

Simple format string	`There are %d apples`
Position format string	`Your balance is %1$f %2$s`
Named format string	`Your balance is %+.2<amount>f %<currency>s`
Named template string	`Your balance is %{amount} %{currency}`
Flag to enable	*ruby-format*

参见:

Ruby Kernel#sprintf

Vue I18n 格式

The Vue I18n formatting does not match source

已命名格式	`There are {count} apples`
Rails i18n 格式	`There are %{count} apples`
连结的语言环境消息	`@:message.dio @:message.the_world!`
Flag to enable	*vue-format*

参见:

Vue I18n Formatting, Vue I18n Linked locale messages

已被翻译

This string has been translated in the past

Means a string has been translated already. This can happen when the translations have been reverted in VCS or lost otherwise.

不一致的

This string has more than one translation in this project or is not translated in some components.

Weblate checks translations of the same string across all translation within a project to help you keep consistent translations.

The check fails on differing translations of one string within a project. This can also lead to inconsistencies in displayed checks. You can find other translations of this string on the *Other occurrences* tab.

注解: This check also fires in case the string is translated in one component and not in another. It can be used as a quick way to manually handle strings which are not translated in some components just by clicking on the *Use this translation* button displayed on each line in the *Other occurrences* tab.

You can use 自动化翻译 addon to automate translating of newly added strings which are already translated in another component.

参见:

跨组件保持翻译一致

已使用 Kashida 字符

The decorative kashida letters should not be used

3.5 新版功能.

The decorative Kashida letters should not be used in translation. These are also known as Tatweel.

参见:

Kashida on Wikipedia

Markdown 链接

Markdown links do not match source

3.5 新版功能.

Markdown links do not match source.

参见:

Markdown links

Markdown 引用

Markdown link references do not match source

3.5 新版功能.

Markdown link references do not match source.

参见:

Markdown links

Markdown 语法

Markdown syntax does not match source

3.5 新版功能.

Markdown 语法与原文不匹配

参见:

Markdown span elements

译文最大长度

Translation should not exceed given length

检查翻译的长度是否可匹配可用的空间。这只检查翻译字符的长度。

Unlike the other checks, the flag should be set as a `key:value` pair like `max-length:100`.

提示: This check looks at number of chars, what might not be the best metric when using proportional fonts to render the text. The 最大翻译大小 check does check actual rendering of the text.

The `replacements:` flag might be also useful to expand placeables before checking the string.

最大翻译大小

Translation rendered text should not exceed given size

3.7 新版功能.

Translation rendered text should not exceed given size. It renders the text with line wrapping and checks if it fits into given boundaries.

This check needs one or two parameters - maximal width and maximal number of lines. In case the number of lines is not provided, one line text is considered.

You can also configure used font by `font-*` directives (see 定制行为), for example following translation flags say that the text rendered with ubuntu font size 22 should fit into two lines and 500 pixels:

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

提示: You might want to set `font-*` directives in 组件配置 to have the same font configured for all strings within a component. You can override those values per string in case you need to customize it per string.

The `replacements:` flag might be also useful to expand placeables before checking the string.

参见:

管理字型, 定制行为, 译文最大长度

不匹配 \n

译文中的 n 数量和原文不一致

Usually escaped newlines are important for formatting program output. Check fails if the number of `\n` literals in translation do not match the source.

不匹配的冒号

Source and translation do not both end with a colon

Checks that colons are replicated between both source and translation. The presence of colons is also checked for various languages where they do not belong (Chinese or Japanese).

参见:

Colon on Wikipedia

不匹配的省略号

Source and translation do not both end with an ellipsis

Checks that trailing ellipses are replicated between both source and translation. This only checks for real ellipsis (…) not for three dots (`...`).

An ellipsis is usually rendered nicer than three dots in print, and sounds better with text-to-speech.

参见:

Ellipsis on Wikipedia

不匹配的感叹号

Source and translation do not both end with an exclamation mark

Checks that exclamations are replicated between both source and translation. The presence of exclamation marks is also checked for various languages where they do not belong (Chinese, Japanese, Korean, Armenian, Limbu, Myanmar or Nko).

参见:

Exclamation mark on Wikipedia

不匹配的句号

Source and translation do not both end with a full stop

Checks that full stops are replicated between both source and translation. The presence of full stops is checked for various languages where they do not belong (Chinese, Japanese, Devanagari or Urdu).

参见:

Full stop on Wikipedia

不匹配的问号

源文件和译文没有都以问号结尾

Checks that question marks are replicated between both source and translation. The presence of question marks is also checked for various languages where they do not belong (Armenian, Arabic, Chinese, Korean, Japanese, Ethiopic, Vai or Coptic).

参见:

Question mark on Wikipedia

不匹配的分号

Source and translation do not both end with a semicolon

Checks that semicolons at the end of sentences are replicated between both source and translation. This can be useful to keep formatting of entries such as desktop files.

参见:

Semicolon on Wikipedia

换行符不符

Number of new lines in translation does not match source

Usually newlines are important for formatting program output. Check fails if the number of `\n` literals in translation do not match the source.

缺少复数形式

Some plural forms are not translated

Checks that all plural forms of a source string have been translated. Specifics on how each plural form is used can be found in the string definition.

Failing to fill in plural forms will in some cases lead to displaying nothing when the plural form is in use.

占位符

Translation is missing some placeholders:

3.9 新版功能.

在 4.3 版更改: 你可以使用正则表达式作为占位符。

Translation is missing some placeholders. These are either extracted from the translation file or defined manually using `placeholders` flag, more can be separated with colon, strings with space can be quoted:

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

In case you have some syntax for placeholders, you can use an regular expression:

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

参见:

定制行为

标点间距

Missing non breakable space before double punctuation sign

3.9 新版功能.

Checks that there is non breakable space before double punctuation sign (exclamation mark, question mark, semicolon and colon). This rule is used only in a few selected languages like French or Breton, where space before double punctuation sign is a typographic rule.

参见:

French and English spacing on Wikipedia

正则表达式

Translation does not match regular expression:

3.9 新版功能.

Translation does not match regular expression. The expression is either extracted from the translation file or defined manually using `regex` flag:

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

相同复数

Some plural forms are translated in the same way

Check that fails if some plural forms are duplicated in the translation. In most languages they have to be different.

换行开头

Source and translation do not both start with a newline

Newlines usually appear in source strings for good reason, omissions or additions can lead to formatting problems when the translated text is put to use.

参见:

换行结尾

空格开头

Source and translation do not both start with same number of spaces

A space in the beginning of a string is usually used for indentation in the interface and thus important to keep.

换行结尾

Source and translation do not both end with a newline

Newlines usually appear in source strings for good reason, omissions or additions can lead to formatting problems when the translated text is put to use.

参见:

换行开头

空格结尾

Source and translation do not both end with a space

Checks that trailing spaces are replicated between both source and translation.

Trailing space is usually utilized to space out neighbouring elements, so removing it might break layout.

未更改的翻译

Source and translation are identical

Happens if the source and corresponding translation strings is identical, down to at least one of the plural forms. Some strings commonly found across all languages are ignored, and various markup is stripped. This reduces the number of false positives.

This check can help find strings mistakenly untranslated.

The default behavior of this check is to exclude words from the built-in blacklist from the checking. These are words which are frequently not being translated. This is useful to avoid false positives on short strings, which consist only of single word which is same in several languages. This blacklist can be disabled by adding `strict-same` flag to string or component.

参见:

组件配置, 定制行为

不安全的 HTML 网站

The translation uses unsafe HTML markup

3.9 新版功能.

The translation uses unsafe HTML markup. This check has to be enabled using `safe-html` flag (see 定制行为). There is also accompanied autofixer which can automatically sanitize the markup.

参见:

The HTML check is performed by the Bleach library developed by Mozilla.

URL

The translation does not contain an URL

3.5 新版功能.

The translation does not contain an URL. This is triggered only in case the unit is marked as containing URL. In that case the translation has to be a valid URL.

XML 标记

XML tags in translation do not match source

This usually means the resulting output will look different. In most cases this is not a desired result from changing the translation, but occasionally it is.

Checks that XML tags are replicated between both source and translation.

XML 语法

The translation is not valid XML

2.8 新版功能.

The XML markup is not valid.

零宽空格

Translation contains extra zero-width space character

Zero-width space (<U+200B>) characters are used to break messages within words (word wrapping).

As they are usually inserted by mistake, this check is triggered once they are present in translation. Some programs might have problems when this character is used.

参见:

Zero width space on Wikipedia

1.5.4 Source checks

Source checks can help developers improve the quality of source strings.

省略号

The string uses three dots (…) instead of an ellipsis character (…)

This fails when the string uses three dots (`...`) when it should use an ellipsis character (…).

Using the Unicode character is in most cases the better approach and looks better rendered, and may sound better with text-to-speech.

参见:

Ellipsis on Wikipedia

长期未翻译

The string has not been translated for a long time

4.1 新版功能.

When the string has not been translated for a long time, it is can indicate problem in a source string making it hard to translate.

多项检查失败

The translations in several languages have failing checks

Numerous translations of this string have failing quality checks. This is usually an indication that something could be done to improve the source string.

This check failing can quite often be caused by a missing full stop at the end of a sentence, or similar minor issues which translators tend to fix in translation, while it would be better to fix it in the source string.

多个未命名的变量

There are multiple unnamed variables in the string, making it impossible for translators to reorder them

4.1 新版功能.

There are multiple unnamed variables in the string, making it impossible for translators to reorder them.

Consider using named variables instead to allow translators to reorder them.

未复数化

The string is used as plural, but not using plural forms

The string is used as a plural, but does not use plural forms. In case your translation system supports this, you should use the plural aware variant of it.

For example with Gettext in Python it could be:

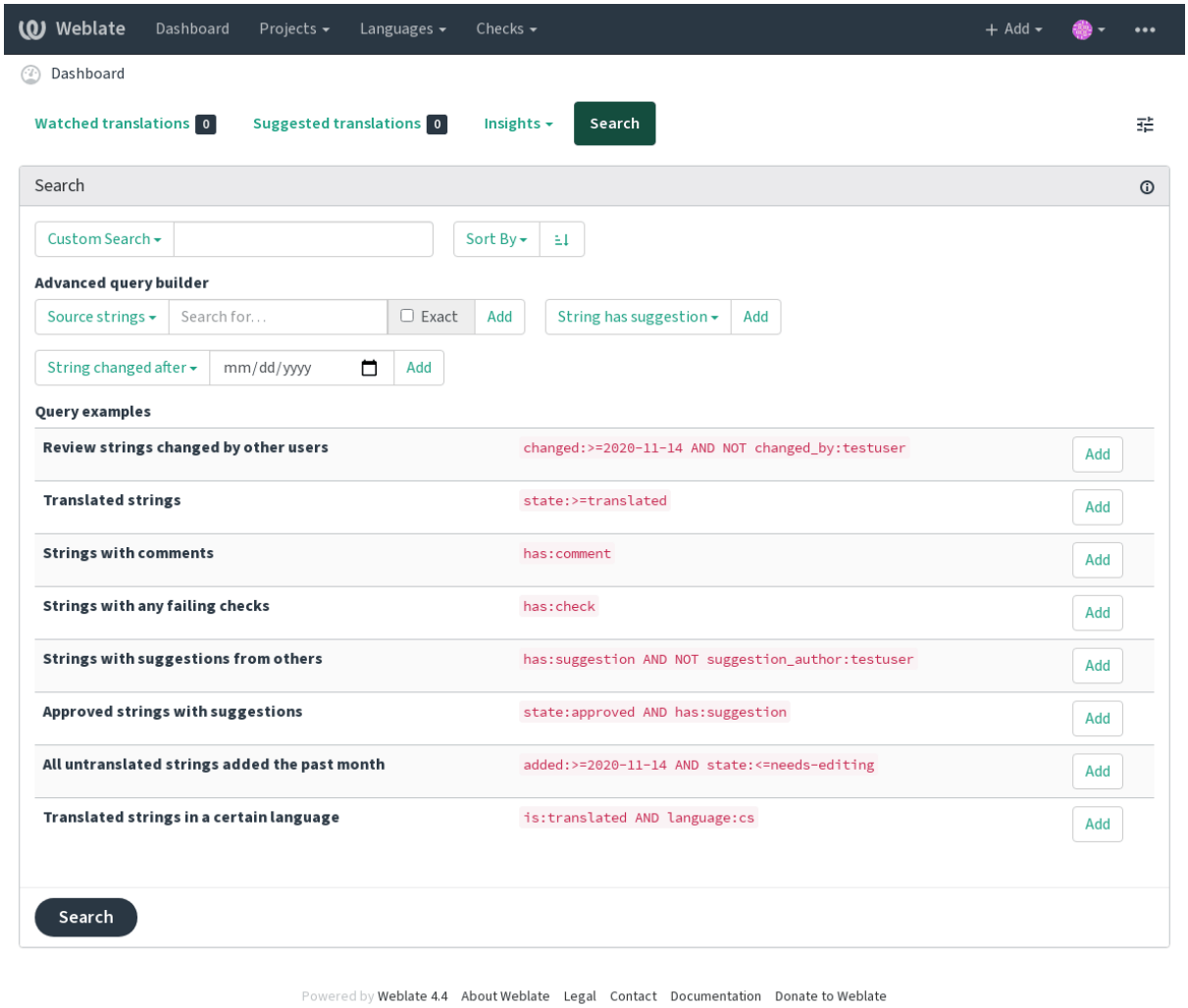
```
from gettext import ngettext

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

1.6 Searching

3.9 新版功能.

Advanced queries using boolean operations, parentheses, or field specific lookup can be used to find the strings you want.

When no field is defined, the lookup happens on *Source*, *Translate* and *Context* fields.



1.6.1 Simple search

Any phrase typed into the search box is split into words. Strings containing any of them are shown. To look for an exact phrase, put "the searchphrase" into quotes (both single (') and double (") quotes will work): `"this is a quoted string"` or `'another quoted string'`.

1.6.2 Fields

`source:TEXT` Source string case insensitive search.

`target:TEXT` Target string case insensitive search.

`context:TEXT` Context string case insensitive search.

`key:TEXT` Key string case insensitive search.

`note:TEXT` Comment string case insensitive search.

`location:TEXT` Location string case insensitive search.

`priority:NUMBER` String priority.

`added:DATETIME` Timestamp for when the string was added to Weblate.

`state:TEXT` State search (`approved`, `translated`, `needs-editing`, `empty`, `read-only`), supports *Field operators*.

pending:BOOLEAN String pending for flushing to VCS.

has:TEXT Search for string having attributes - `plural`, `context`, `suggestion`, `comment`, `check`, `dismissed-check`, `translation`, `variant`, `screenshot` (works only on source strings).

is:TEXT Search for string states (`pending`, `translated`, `untranslated`).

language:TEXT String target language.

组件:TEXT 组件标识串，请参见组件标识串。

项目:TEXT Project slug, see 项目标识串.

changed_by:TEXT String was changed by author with given username.

changed:DATETIME String content was changed on date, supports *Field operators*.

change_time:DATEIME String was changed on date, supports *Field operators*, unlike `changed` this includes event which don't change content and you can apply custom action filtering using `change_action`.

change_action:TEXT Filters on change action, useful together with `change_time`. Accepts English name of the change action, either quoted and with spaces or lowercase and spaces replaced by dash. See 搜索更改中 for examples.

check:TEXT String has failing check.

dismissed_check:TEXT String has dismissed check.

comment:TEXT Search in user comments.

comment_author:TEXT Filter by comment author.

suggestion:TEXT Search in suggestions.

suggestion_author:TEXT Filter by suggestion author.

1.6.3 Boolean operators

You can combine lookups using `AND`, `OR`, `NOT` and parentheses to form complex queries. For example: `state:translated AND (source:hello OR source:bar)`

1.6.4 Field operators

You can specify operators, ranges or partial lookups for date or numeric searches:

state:>=translated State is `translated` or better (`approved`).

changed:2019 Changed in year 2019.

changed:[2019-03-01 to 2019-04-01] Changed between two given dates.

1.6.5 Exact operators

You can do an exact match query on different string fields using = operator. For example, to search for all source strings exactly matching `hello world`, use: `source:="hello world"`. For searching single word expressions, you can skip quotes. For example, to search for all source strings matching `hello`, you can use: `source:=hello`.

1.6.6 搜索更改中

4.4 新版功能.

Searching for history events can be done using `change_action` and `change_time` operators.

For example, searching for strings marked for edit in 2018 can be entered as `change_time:2018 AND change_action:marked-for-edit` or `change_time:2018 AND change_action:"Marked for edit"`.

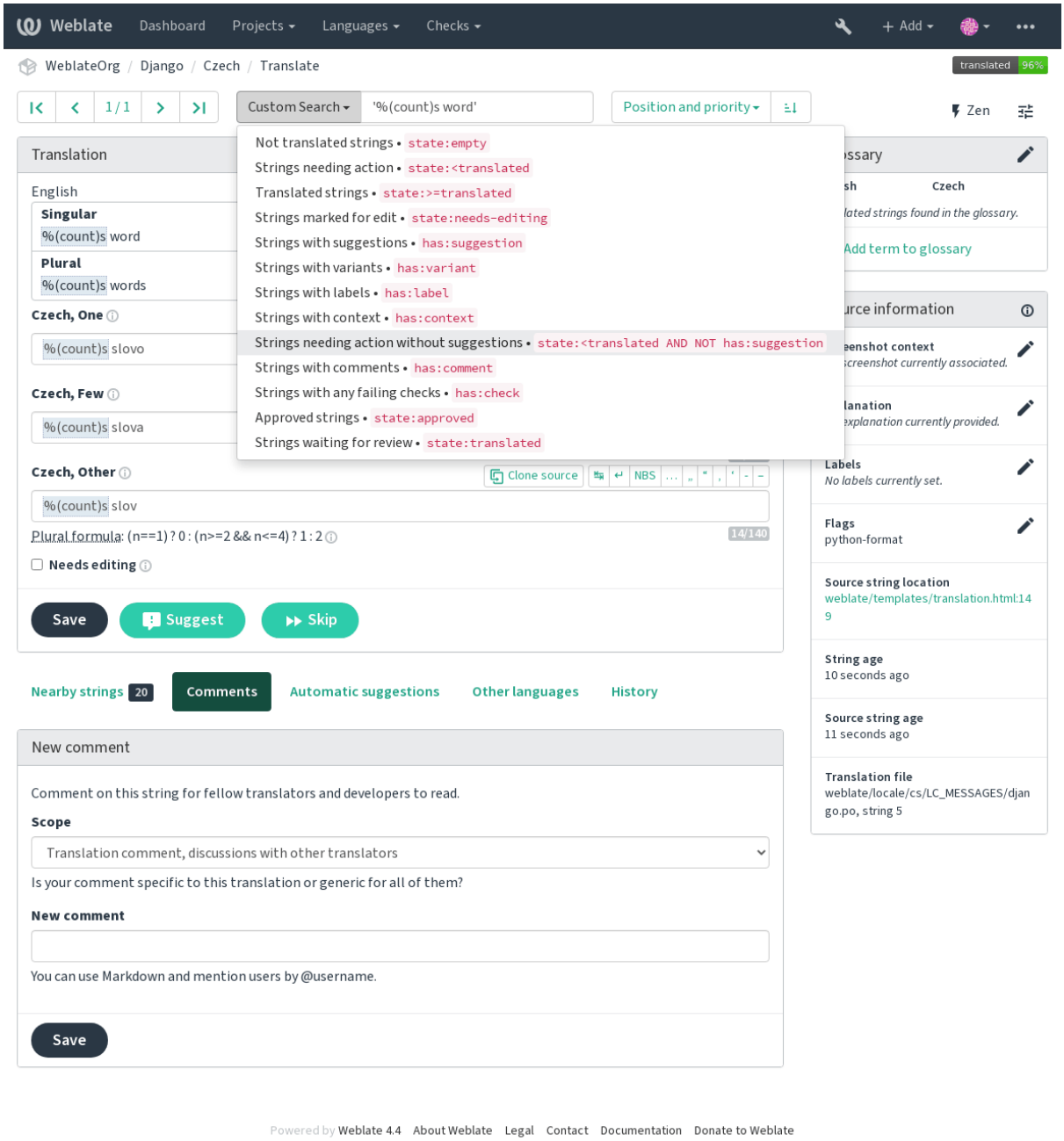
1.6.7 Regular expressions

Anywhere text is accepted you can also specify a regular expression as `r"regexp"`.

For example, to search for all source strings which contain any digit between 2 and 5, use `source:r"[2-5]"`.

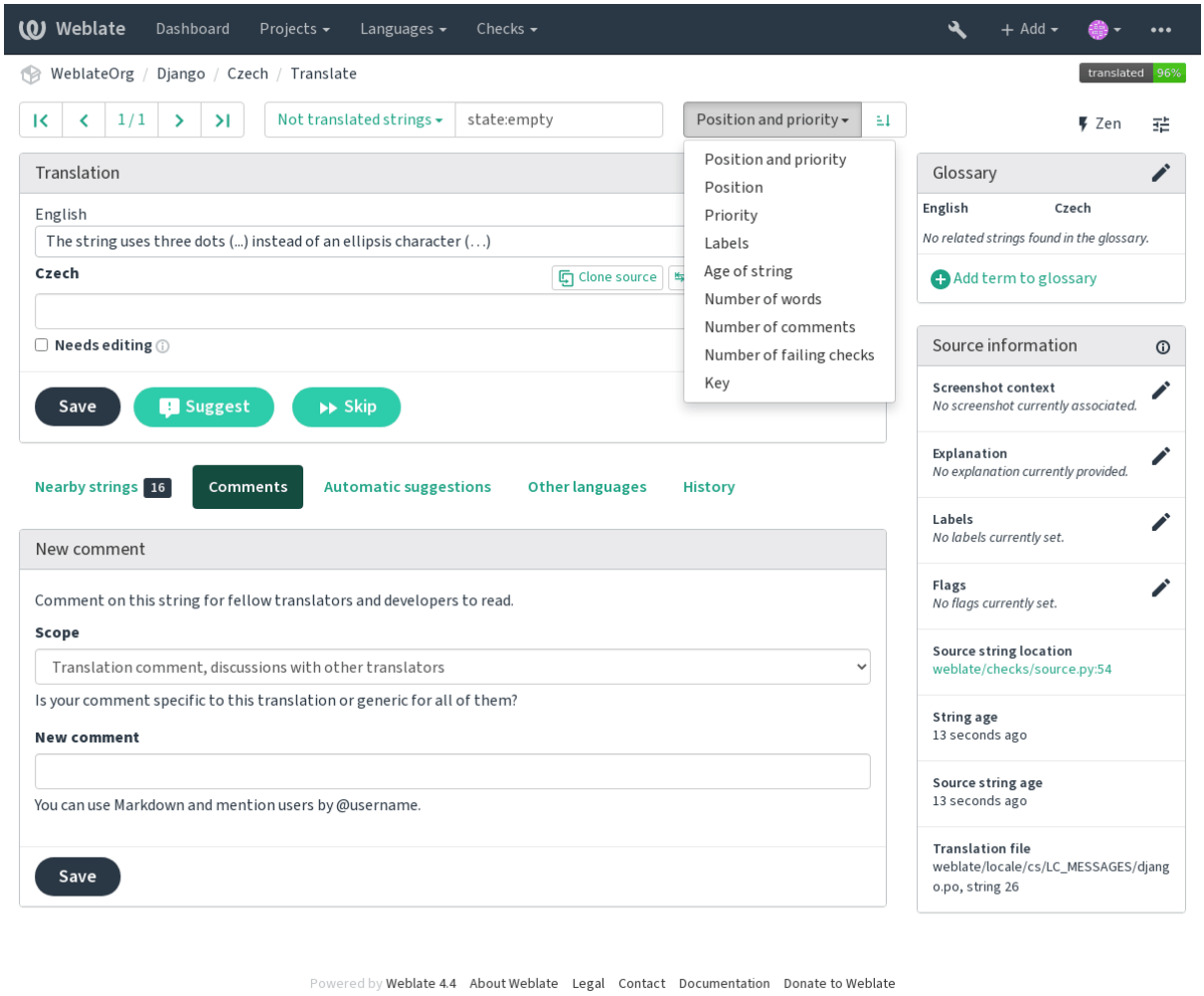
1.6.8 Predefined queries

You can select out of predefined queries on the search page, this allows you to quickly access the most frequent searches:



1.6.9 Ordering the results

There are many options to order the strings according to your needs:



1.7 翻译工作流

Using Weblate is a process that brings your users closer to you, by bringing you closer to your translators. It is up to you to decide how many of its features you want to make use of.

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。

1.7.1 翻译访问

访问控制 在工作流中没有太多讨论，因为每个访问控制选项都可以应用于任何工作流。请查阅该文档以获取有关如何管理对翻译的访问的信息。

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 *Translate* 权限的用户。

1.7.2 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，它可能被标志为需要编辑（例如，当它在 Gettext 文件中获得一个 fuzzy 标记）。

等待复查 翻译已完成，但未进行审核。它作为有效翻译存储在文件中。

已批准 翻译已在审核中得到批准。翻译者不能再更改它，只能由审阅者更改。译者只能向其中添加建议。

建议 建议仅存储在 Weblate 中，而不存储在翻译文件中。

1.7.3 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。
- 当翻译者不确定更改时，建议是建议更改的可选方法。

设置	值	备注
启用复查	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	
翻译组	用户	或使用访问控制 进行 翻译。
审核者组	不可用	不曾用过。

1.7.4 同行评审

使用此工作流程，任何人都可以添加建议，并且需要其他成员的同意才能被接受为翻译。

- 任何用户都可以添加建议。
- 任何用户都可以对建议投票。
- 当给定预定数量的投票时，建议就变成翻译。

设置	值	备注
启用复查	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。
翻译组	用户	或使用访问控制 进行 翻译。
审核者组	不可用	未使用，所有翻译员都审阅。

1.7.5 专门的审核者

2.18 新版功能: 从 Weblate 2.18 开始，支持正确的审核工作流。

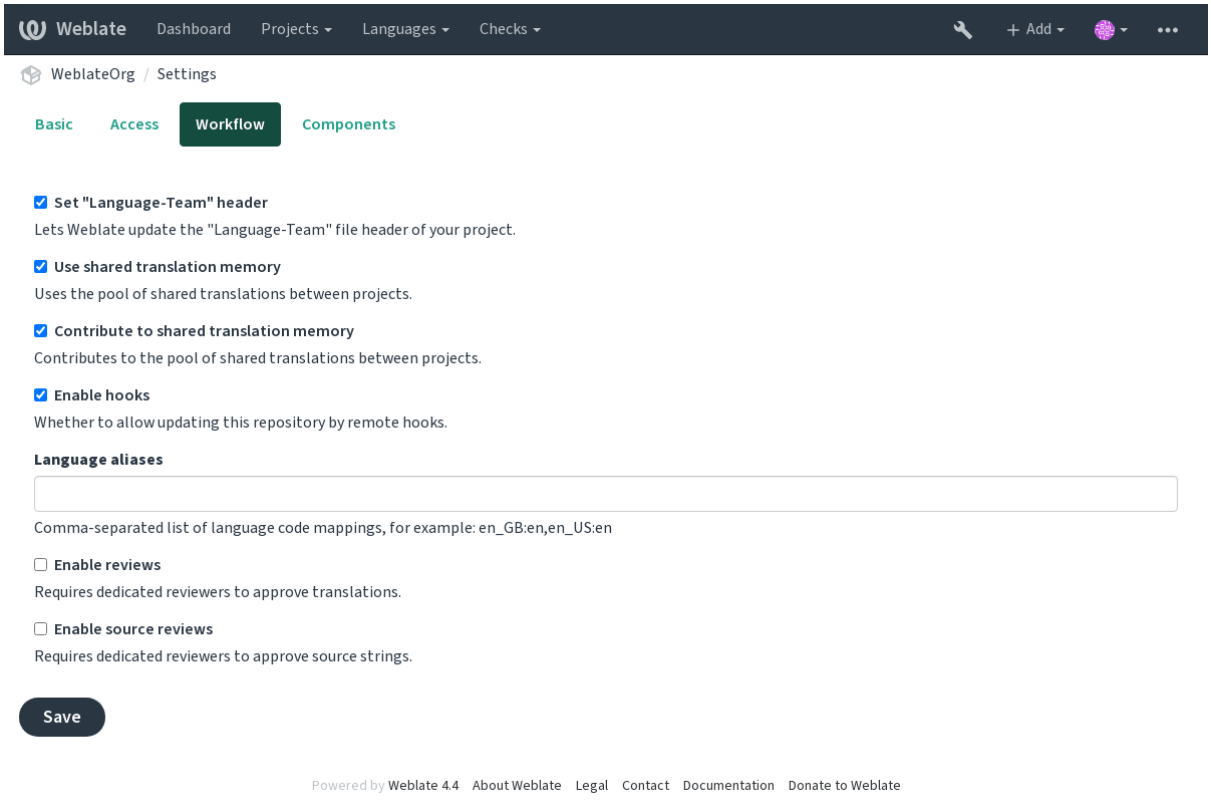
使用专门的审核者，您有两组用户，一组可以提交翻译，而另一组可以审核它们以确保翻译一致且质量良好。

- 任何用户都可以编辑未批准的翻译。
- *Reviewer* 可以批准/否决字符串。
- 审核者可以编辑所有翻译（包括批准的翻译）。
- 建议还可以用于建议更改已批准的字符串。

设置	值	备注
启用复查	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	
翻译组	用户	或使用访问控制 进行 翻译。
审核者组	校对	或使用访问控制 进行 审核。

1.7.6 打开审核

可以在项目设置的 *Workflow* 子页面中的项目配置中打开审核（位于 *Manage* → *Settings* 菜单中）:

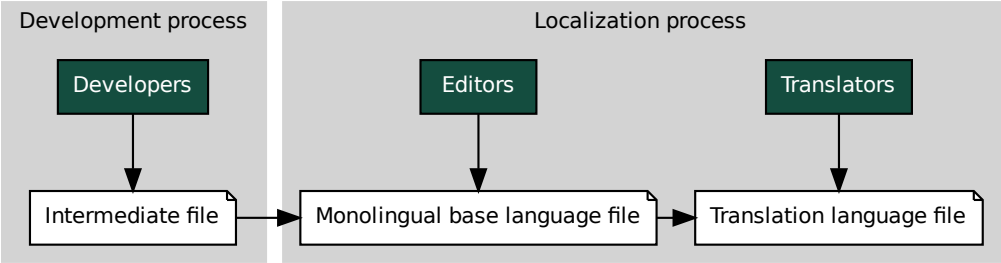


注解: 根据 Weblate 配置的不同，该设置可能对您不可用。例如，在 Hosted Weblate 上，这不适用于免费托管的项目。

1.7.7 Quality gateway for the source strings

In many cases the original source language strings are coming from developers, because they write the code and provide initial strings. However developers are often not a native speakers in the source language and do not provide desired quality of the source strings. The intermediate translation can help you in addressing this - there is additional quality gateway for the strings between developers and translators and users.

By setting 中间语言文件, this file will be used as source for the strings, but it will be edited to source language to polish it. Once the string is ready in the source language, it will be also available for translators to translate into additional languages.



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

1.7.8 源字符串复查

通过允许启用来源评论，复查过程可以应用到源字符串上。一旦允许，用户可以汇报源字符串种的情况。实际过程依赖于使用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核者 相似——一旦源字符串汇报了情况，就会被标记为 *Needs editing* 。

双语言格式不允许直接编辑源字符串（他们典型地是从源代码种直接提取的）。在这种情况下，*Source needs review* 标签会贴到翻译者回到的字符串上。可以复查这样的字符串，并在源中编辑或删除标签。

参见:

双语和单语格式, 专门的审核者, labels

1.8 常见问题

1.8.1 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. 新建您的 Git 仓库，来告知 Weblate 何时有任何更改，请参阅通知钩子 以获取有关如何执行此操作的信息。
2. 在 Weblate 中的组件配置 配置中设置推送 URL，这使 Weblate 可以将更改推送到仓库。

3. 在 Weblate 中打开项目配置 配置上的 push-on-commit，这将使 Weblate 在 Weblate 发生更改时将更改推送到仓库。

参见:

持续本地化集成, 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*Accessing repositories* 。

如何修复翻译中的合并冲突？

在翻译文件在 Weblate 和上游仓库同时更改时，合并冲突不时发生。通常可以通过在对翻译文件进行更改前（例如在运行 msgmerge 前）合并仓库来避免发生这个情况。只要告诉 Weblate 执行挂起所有的翻译（可以在 *Manage* 菜单中的 *Repository maintenance* 中去做），并合并仓库即可（如果自动推送没有打开的话）。

如果已经进入了合并冲突，那么最容易的方式是在您自己的工作站上本地解决所有冲突——就是简单地将 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动作。

注解: 依赖于您的设置，访问 Weblate 仓库会需要认证。当在 Weblate 中使用*Git 导出器* 中的构建时，要以用户名和 API 密钥认证。

```
# 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:
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/master

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

# 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
```

(下页继续)

(续上页)

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

# Merge master branch:
git checkout master
git merge weblates-second/master
... # 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/master

# 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？，*持续本地化集成*，*避免合并冲突*

如何立刻翻译几个分支？

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组件配置 （默认行为），所作的更改自动传递给其它组件。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，就会不得不合并这些分支（依赖于您的开发工作流程），而丢弃差异：

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

参见:

跨组件保持翻译一致

如何翻译多平台项目？

Weblate 支持大范围的文件格式（请参见支持的文件格式），而最容易的方式时对每个平台使用本地格式。

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

参见：

跨组件保持翻译一致

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？

这严重依赖于您的设置，Weblate 在这个领域是非常灵活的。这里是使用 Weblate 的一些工作流程的例子：

- 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 只访问翻译，而不向它暴露源代码？

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. 以您的文艺文件创建仓库。
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。

为什么所有提价由 Weblate <noreply@weblate.org> 执行？

这是默认的提交者的名称，当创建翻译组件时配置。可以在任何时间在管理时将其更改。

（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。

参见：

组件配置

1.8.2 用法

如何复查其他人的翻译？

- 可以订阅通知 中做出的任何更改，然后检查当它们通过电子邮件进入时检查其他人的贡献。
- 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。

如何提供源字符串的反馈？

在翻译下面的文本标签上，可以使用 *Comments* 标签来提供源字符串的反馈，或者与其他翻译者讨论。

参见：

report-source, 注释

翻译时如何使用现有的翻译？

- 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 事件中翻译所有相关项目，这样同样可以从其它项目中自动地拾取翻译。

参见：

机器翻译, 自动建议

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。

对于单语言文件（请参见支持的文件格式），Weblate 会将没有出现的新翻译的字符串添加在 *template* 中，而不是在实际的翻译中。然而它不会对任何旧的字符串执行自动清理，因为那会具有难以预料的结果。如果想做的话，请安装适当的插件，来根据您的要求处理清理工作。

Weblate 也不会尝试以任何方式更新双语言文件，所以如果需要 `po` 文件从 `pot` 更新的话，需要您自己去做或使用插件。

参见：

用脚本处理仓库，清理翻译文件，移除空白字符串，更新 *RESX* 文件，更新 *PO* 文件以匹配 *POT* 文件 *(msgmerge)*

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。

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 除了允许翻译者翻译之外，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。

注解： 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
```

在想要进行自动更新的情况下，可以安装插件*更新 PO 文件以匹配 POT 文件 (msgmerge)*。

1.8.3 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，请更改组件设置中的:ref:*component-source_language*。
- 如果不需要源语言的翻译文件，请从存储库中将其删除。
- 如果需要源语言的翻译文件，但 Weblate 应该忽略它，请调整:ref: *component-language_regex* 来排除它。

1.8.4 功能

Weblate 支持 Git 和 Mercurial 意外的 VCS 吗？

Weblate 当前不支持*Git* （扩展支持*GitHub*、*Gerrit* 和*Subversion*）和*Mercurial* 以外的任何 VCSes，但能够对其它 VCSes 写后端。

还可以在 Git 中使用*Git remote helpers* 来访问其它 VCSes。

Weblate 还支持无 VCS 的操作，请参见*Local files*。

注解： 为了本地支持其它 VCSes，Weblate 需要使用分布式 VCS，并可能能够调整来与 Git 和 Mercurial 以外的其它任何 VCSes 工作，但必须有人应用这项支持。

参见：

版本控制集成

Weblate 如何记录翻译者？

Weblate 中所做的每个更改都将以翻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，则文件头会更新为包含翻译者的名称。

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 以每个 PO 文件表似乎成单一组件来设计。这对翻译者有好处，使他们指导真正在翻译什么。如果您认为自己的项目因作为一个整体来翻译，那么考虑合并这些 PO 文件。对那些即使不使用 Weblate 的翻译者也更容易一些。

注解： 在对这个特性有大量需求的情况下，它可能会应用与将来的版本中。

Weblate 为什么使用 sr_Latn 或 zh_Hant 这样的语言编码？

这些语言编码由 **RFC 4646** 定义，来更好地指示它们是真正不同的语言而不是之前错误使用修饰符（`@latin` 变体）或国家代码（对于中文）。

Weblate 仍然理解传统的语言编码并将它们映射到当前的便马上——例如 `sr@latin` 将被处理为 `sr_Latn` 或者 `zh@CN` 处理为 `zh_Hans`。

1.9 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，可能会出现未经良好测试的格式问题。

参见：

Translation Related File Formats

注解： 为您的应用程序选择文件格式时，最好在您使用的工具箱/平台中保留一些公认的格式。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做出贡献。

1.9.1 双语和单语格式

支持 monolingual 和 bilingual 格式。双语格式在单个文件中存储两种语言——源和翻译（典型示例是*GNU gettext*，*XLIFF* 或*Apple iOS strings* 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（通常是*Android string resources*）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 要求访问一个包含完整字符串列表的文件，以与其源一起翻译——该文件在 Weblate 中称为单语言译文模版语言文件，尽管命名方式可能会有所不同。

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。

1.9.2 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）。

1.9.3 翻译类型功能

所有受支持格式的功能：

格式	语言能力[1]	复数[2]	注释[3]	语境[4]	位置[5]	标记[8]	附加状态[6]
GNU gettext	双语	支持	支持	支持	支持	是[9]	需要编辑
单语 gettext	单一	支持	支持	支持	支持	是[9]	需要编辑
XLIFF	二者	支持	支持	支持	支持	是[10]	需要编辑，已同意
Java 属性	二者	不支持	支持	不支持	不支持	不支持	
GWT properties	单一	支持	支持	不支持	不支持	不支持	
Joomla translations	单一	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	是[10]	需要编辑
Android string resources	单一	支持	是[7]	不支持	不支持	是[10]	
Apple iOS strings	双语	不支持	支持	不支持	不支持	不支持	
PHP 字符串	单一	否[11]	支持	不支持	不支持	不支持	
JSON files	单一	不支持	不支持	不支持	不支持	不支持	
JSON i18next files	单一	支持	不支持	不支持	不支持	不支持	
go-i18n JSON files	单一	支持	不支持	不支持	不支持	不支持	
ARB File	单一	支持	支持	不支持	不支持	不支持	
WebExtension JSON	单一	支持	支持	不支持	不支持	不支持	
.XML resource files	单一	不支持	支持	不支持	不支持	是[10]	
CSV 文件	二者	不支持	支持	支持	支持	不支持	需要编辑
YAML files	单一	不支持	支持	不支持	不支持	不支持	
Ruby YAML files	单一	支持	支持	不支持	不支持	不支持	
DTD files	单一	不支持	不支持	不支持	不支持	不支持	
Flat XML	单一	不支持	不支持	不支持	不支持	是[10]	
Windows RC files	单一	不支持	支持	不支持	不支持	不支持	
Excel Open XML	单一	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	单一	不支持	不支持	不支持	不支持	不支持	
Subtitle files	单一	不支持	不支持	不支持	支持	不支持	
HTML files	单一	不支持	不支持	不支持	不支持	不支持	
OpenDocument Format	单一	不支持	不支持	不支持	不支持	不支持	

下页继续

表 1 – 续上页

格式	语言能力[1]	复数[2]	注释[3]	语境[4]	位置[5]	标记[8]	附加状态[6]
IDML Format	单一	不支持	不支持	不支持	不支持	不支持	
INI translations	单一	不支持	不支持	不支持	不支持	不支持	
Inno Setup INI 翻译	单一	不支持	不支持	不支持	不支持	不支持	

1.9.4 GNU gettext

翻译自由软件用得最广泛的格式。

通过调整文件头或链接到相应的源文件，可以支持存储在文件中的语境信息。

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 组件配置	
文件掩码	`po/*.po`
单语言译文模版语言文件	*Empty*
新翻译的译文模版	`po/messages.pot`
文件格式	*Gettext PO file*

参见:

devel/gettext、devel/sphinx、Gettext on Wikipedia、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] 将带有不同数量的字符串正确本地化时复数是必要的。
[3] 注释能够用户传递要翻译的字符串的附加信息。
[4] 文本用于却别不同范围中使用的相同字符串（例如 'Sun'可以用作“Sunday”名称的缩写，或用作我们最近的恒星）。
[5] 源代码中字符串的位置会帮助熟练的译者识别出字符串如何使用。
[8] 请参见*定制行为*
[6] 除了“Not translated”（未翻译）和“Translated”（已翻译），文件格式支持的其它状态。
[9] gettext 类型的注释用作标记。
[10] 对于基于格式所有 XML，标记从非标准的属性 `weblate-flags` 中提取。此外，`max-length:N` 通过 `maxwidth` *attribute <http://docs.oasis-open.org/xliff/v1.2/os/xliff-core.html#maxwidth>'_ 如在 XLIFF 标准中定义的，请参见:ref:'xliff-flags*。
[7] 放在 `<string>` 元素前的 XML 注释，解析为开发者的注释。
[11] 只对于 Laravel 来支持复数，在字符串语法中使用来定义它们，请参见 Localization in Laravel。

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，尽管在将组件导入 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 组件配置	
文件掩码	`po/*.po`
单语言译文模版语言文件	`po/en.po`
新翻译的译文模版	`po/messages.pot`
文件格式	*Gettext PO* 文件（单语）

1.9.5 XLIFF

创建基于 XML 的格式来标准化翻译文件，但最终它是该领域中 许多标准 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。

参见：

XML Localization Interchange File Format (XLIFF) 规范

翻译状态

在 3.3 版更改: Weblate 忽略 3.3 发布版本之前的状态属性。

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"`，那么它还将被导入 Weblate 作为 "Approved"（同意的），任何其它内容被导入作为 "Waiting for review"（等待复核，这与 XLIFF 规范匹配）。

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` 属性只在字符串已经被复查的情况下添加。
- 在其它情况下不添加属性，但在它们出现的情况下更新。

那意味着当使用 XLIFF 格式时，强烈推荐打开 Weblate 复查过程，从而能看到并更改字符串的同意状态。

请参见专门的审核者。

相似地在导入这样的文件时（在上传表格中），应该选择 *Processing of strings needing edit* 之下的 *Import as translated*。

XLIFF 中的空白字符和新行符

在 XML 格式中不区分通常类型或量的空白字符。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。字符集中的任何空白字符被转换为下划线，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）。

单元键或上下文

Weblate 通过 `resname` 属性在它出线的情况下识别 XLIFF 文件中的单元，并退回到 `id`（如果出现的话与 `file` 标签一起）。

`resname` 属性被认为是人类友好的单元识别符，对 Weblate 比 `id` 更适于显示。在整个 XLIFF 文件中 `resname` 具有一致性。这是 Weblate 需要的，并且不被 XLIFF 标准覆盖——它不在这个属性上放入任何独特性限制。

用于双语言 XLIFF 的典型 Weblate 组件配置	
文件掩码	`localizations/*.xliff`
单语言译文模版语言文件	*Empty*
新翻译的译文模版	`localizations/en-US.xliff`
文件格式	*XLIFF Translation File*

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

参见:

XLIFF on Wikipedia、XLIFF、font attribute in XLIFF 1.2、maxwidth attribute in XLIFF 1.2

1.9.6 Java 属性

用于翻译的本地 Java 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。

注解: 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 组件配置	
文件掩码	`src/app/Bundle_*.properties`
单语言译文模版语言文件	`src/app/Bundle.properties`
新翻译的译文模版	*Empty*
文件格式	*Java Properties (ISO-8859-1)*

参见:

Java properties on Wikipedia, Mozilla and Java properties files, 格式化 *Java* 属性文件, 清理翻译文件

1.9.7 GWT properties

Native GWT format for translations.

GWT properties are usually used as monolingual translations.

典型的 Weblate 组件配置	
文件掩码	`src/app/Bundle_*.properties`
单语言译文模版语言文件	`src/app/Bundle.properties`
新翻译的译文模版	*Empty*
文件格式	*GWT Properties*

参见:

GWT localization guide Mozilla and Java properties files, *格式化 Java 属性文件*, *清理翻译文件*

1.9.8 INI translations

4.1 新版功能.

INI file format for translations.

INI translations are usually used as monolingual translations.

典型的 Weblate 组件配置	
文件掩码	`language/*.ini`
单语言译文模版语言文件	`language/en.ini`
新翻译的译文模版	*Empty*
文件格式	*INI File*

注解: Weblate only extracts keys from sections within a INI file. In case your INI file lacks sections, you might want to use *Joomla translations* or *Java 属性* instead.

参见:

INI Files, *Java 属性*, *Joomla translations*, *Inno Setup INI 翻译*

1.9.9 Inno Setup INI 翻译

4.1 新版功能.

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。

注解: The only notable difference to *INI translations* is in supporting `%n` and `%t` placeholders for line break and tab.

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

注解： Only Unicode files (`.islu`) are currently supported, ANSI variant (`.isl`) is currently not supported.

参见：

INI Files, *Joomla translations*, *INI translations*

1.9.10 Joomla translations

2.12 新版功能.

Native Joomla format for translations.

Joomla translations are usually used as monolingual translations.

典型的 Weblate 组件配置	
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`
新翻译的译文模版	*Empty*
文件格式	*Joomla Language File*

参见：

Specification of Joomla language files, Mozilla and Java properties files, *INI translations*, *Inno Setup INI 翻译*

1.9.11 Qt Linguist .ts

Translation format used in Qt based applications.

Qt Linguist files are used as both bilingual and monolingual translations.

Typical Weblate 组件配置 when using as bilingual	
文件掩码	`i18n/app.*.ts`
单语言译文模版语言文件	*Empty*
新翻译的译文模版	`i18n/app.de.ts`
文件格式	*Qt Linguist Translation File*

Typical Weblate 组件配置 when using as monolingual	
文件掩码	`i18n/app.*.ts`
单语言译文模版语言文件	`i18n/app.en.ts`
新翻译的译文模版	`i18n/app.en.ts`
文件格式	*Qt Linguist Translation File*

参见：

Qt Linguist manual, Qt .ts, *双语和单语格式*

1.9.12 Android string resources

Android specific file format for translating applications.

Android string resources are monolingual, the 单语言译文模版语言文件 file is stored in a different location from the others `res/values/strings.xml`.

典型的 Weblate 组件配置	
文件掩码	`res/values-*/strings.xml`
单语言译文模版语言文件	`res/values/strings.xml`
新翻译的译文模版	*Empty*
文件格式	*Android String Resource*

参见:

Android string resources documentation, Android string resources

---

注解: Android *string-array* structures are not currently supported. To work around this, you can break your string arrays apart:

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

become:

```
<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>
```

The *string-array* that points to the *string* elements should be stored in a different file, and not be made available for translation.

This script may help pre-process your existing strings.xml files and translations: https://gist.github.com/paour/11291062

1.9.13 Apple iOS strings

Apple specific file format for translating applications, used for both iOS and iPhone/iPad application translations.

Apple iOS strings are usually used as bilingual translations.

典型的 Weblate 组件配置	
文件掩码	`Resources/*.lproj/Localizable.strings`
单语言译文模版语言文件	`Resources/en.lproj/Localizable.strings` or `Resources/Base.lproj/Localizable.strings`
新翻译的译文模版	*Empty*
文件格式	*iOS Strings (UTF-8)*

参见:

Apple "strings files" documentation, Mac OSX strings

1.9.14 PHP 字符串

PHP translations are usually monolingual, so it is recommended to specify a base file with (what is most often the) English strings.

Example file:

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

典型的 Weblate 组件配置	
文件掩码	`lang/*/texts.php`
单语言译文模版语言文件	`lang/en/texts.php`
新翻译的译文模版	`lang/en/texts.php`
文件格式	*PHP strings*

Laravel PHP 字符串

在 4.1 版更改.

The Laravel PHP localization files are supported as well with plurals:

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

参见:

PHP, Localization in Laravel

1.9.15 JSON files

2.0 新版功能.

在 2.16 版更改: Since Weblate 2.16 and with translate-toolkit at-least 2.2.4, nested structure JSON files are supported as well.

在 4.3 版更改: The structure of JSON file is properly preserved even for complex situations which were broken in prior releases.

JSON format is used mostly for translating applications implemented in JavaScript.

Weblate currently supports several variants of JSON translations:

- Simple key / value files, used for example by *vue-i18n* or *react-intl*.
- Files with nested keys.
- *JSON i18next files*
- *go-i18n JSON files*
- *WebExtension JSON*
- *ARB File*

JSON translations are usually monolingual, so it is recommended to specify a base file with (what is most often the) English strings.

Example file:

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

Nested files are supported as well (see above for requirements), such a file can look like:

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

提示: The *JSON file* and *JSON nested structure file* can both handle same type of files. The only difference between them is when adding new strings. The nested variant tries to parse the key and insert the new string into the matching structure.

典型的 Weblate 组件配置	
文件掩码	`langs/translation-*.json`
单语言译文模版语言文件	`langs/translation-en.json`
新翻译的译文模版	*Empty*
文件格式	*JSON nested structure file*

参见:

JSON, 自定义 *JSON* 输出, 清理翻译文件,

1.9.16 JSON i18next files

在 2.17 版更改: Since Weblate 2.17 and with translate-toolkit at-least 2.2.5, i18next JSON files with plurals are supported as well.

i18next is an internationalization framework written in and for JavaScript. Weblate supports its localization files with features such as plurals.

i18next translations are monolingual, so it is recommended to specify a base file with (what is most often the) English strings.

注解: Weblate supports the i18next JSON v3 format. The v2 and v1 variants are mostly compatible, with exception of how plurals are handled.

Example file:

```
{
  "hello": "Hello",
  "apple": "I have an apple",
  "apple_plural": "I have {{count}} apples",
```

(下页继续)

(续上页)

```
  "apple_negative": "I have no apples"
}
```

典型的 Weblate 组件配置	
文件掩码	`langs/*.json`
单语言译文模版语言文件	`langs/en.json`
新翻译的译文模版	*Empty*
文件格式	*i18next JSON file*

参见:

JSON, i18next JSON Format, 自定义 *JSON* 输出, 清理翻译文件

1.9.17 go-i18n JSON files

4.1 新版功能.

go-i18n translations are monolingual, so it is recommended to specify a base file with (what is most often the) English strings.

注解: Weblate supports the go-i18n JSON v1 format, for flat JSON formats please use *JSON files*. The v2 format with hash is currently not supported.

典型的 Weblate 组件配置	
文件掩码	`langs/*.json`
单语言译文模版语言文件	`langs/en.json`
新翻译的译文模版	*Empty*
文件格式	*go-i18n JSON file*

参见:

JSON, go-i18n, 自定义 *JSON* 输出, 清理翻译文件,

1.9.18 ARB File

4.1 新版功能.

ARB translations are monolingual, so it is recommended to specify a base file with (what is most often the) English strings.

典型的 Weblate 组件配置	
文件掩码	`lib/l10n/intl_*.arb`
单语言译文模版语言文件	`lib/l10n/intl_en.arb`
新翻译的译文模版	*Empty*
文件格式	*ARB file*

参见:

JSON, Application Resource Bundle Specification, Internationalizing Flutter apps, 自定义 *JSON* 输出, 清理翻译文件

1.9.19 WebExtension JSON

2.16 新版功能: This is supported since Weblate 2.16 and with translate-toolkit at-least 2.2.4.

File format used when translating extensions for Mozilla Firefox or Google Chromium.

注解: While this format is called JSON, its specification allows to include comments, which are not part of JSON specification. Weblate currently does not support file with comments.

Example file:

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

典型的 Weblate 组件配置	
文件掩码	`_locales/*/messages.json`
单语言译文模版语言文件	`_locales/en/messages.json`
新翻译的译文模版	*Empty*
文件格式	*WebExtension JSON file*

参见:

JSON, Google chrome.i18n, Mozilla Extensions Internationalization

1.9.20 .XML resource files

2.3 新版功能.

A .XML resource (.resx) file employs a monolingual XML file format used in Microsoft .NET applications. It is interchangeable with .resw, when using identical syntax to .resx.

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

参见:

.NET Resource files (.resx), 清理翻译文件,

1.9.21 CSV 文件

2.4 新版功能.

CSV 文件可以包含源和翻译的简单列表。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:guilabel:`CSV file` 作为一种文件格式。
- 具有两个字段的文件——源和翻译（按此顺序），选择 *Simple CSV file* 作为文件格式
- 无标头文件，字段顺序由 translate-toolkit 定义：location，source，target，ID，fuzzy，context，translator_comments，developer_comments; 挑选:guilabel:`CSV file` 作为一种文件格式。
- Remember to define 单语言译文模版语言文件 when your files are monolingual (see 双语和单语格式).

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，并且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。作为一种解决方法，推荐省略引用的字符。

Example file:

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

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

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

参见:

CSV

1.9.22 YAML files

2.9 新版功能.

The plain YAML files with string keys and values. Weblate also extract strings from lists or dictionaries.

Example of a YAML file:

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

典型的 Weblate 组件配置	
文件掩码	`translations/messages.*.yml`
单语言译文模版语言文件	`translations/messages.en.yml`
新翻译的译文模版	*Empty*
文件格式	*YAML file*

参见:

YAML, *Ruby YAML files*

1.9.23 Ruby YAML files

2.9 新版功能.

Ruby i18n YAML files with language as root node.

Example Ruby i18n YAML file:

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

典型的 Weblate 组件配置	
文件掩码	`translations/messages.*.yml`
单语言译文模版语言文件	`translations/messages.en.yml`
新翻译的译文模版	*Empty*
文件格式	*Ruby YAML file*

参见:

YAML, *YAML files*

1.9.24 DTD files

2.18 新版功能.

Example DTD file:

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

典型的 Weblate 组件配置	
文件掩码	`locale/*.dtd`
单语言译文模版语言文件	`locale/en.dtd`
新翻译的译文模版	*Empty*
文件格式	*DTD file*

参见:

Mozilla DTD format

1.9.25 Flat XML files

3.9 新版功能.

Example of a flat XML file:

```
<?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 组件配置	
文件掩码	`locale/*.xml`
单语言译文模版语言文件	`locale/en.xml`
新翻译的译文模版	*Empty*
文件格式	*Flat XML file*

参见:

Flat XML

1.9.26 Windows RC files

在 4.1 版更改: Support for Windows RC files has been rewritten.

注解: Support for this format is currently in beta, feedback from testing is welcome.

Example Windows RC file:

```
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 组件配置	
文件掩码	`lang/*.rc`
单语言译文模版语言文件	`lang/en-US.rc`
新翻译的译文模版	`lang/en-US.rc`
文件格式	*RC file*

参见:

Windows RC files

1.9.27 应用商店元数据文件

3.5 新版功能.

Metadata used for publishing apps in various app stores can be translated. Currently the following tools are compatible:

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

The metadata consists of several textfiles, which Weblate will present as separate strings to translate.

典型的 Weblate 组件配置	
文件掩码	`fastlane/android/metadata/*`
单语言译文模版语言文件	`fastlane/android/metadata/en-US`
新翻译的译文模版	`fastlane/android/metadata/en-US`
文件格式	*App store metadata files*

提示: In case you don't want to translate certain strings (for example changelogs), mark them read-only (see 定制行为). This can be automated by the 批量编辑.

1.9.28 Subtitle files

3.7 新版功能.

Weblate 可以翻译多个字幕文件:

- SubRip subtitle file (`*.srt`)
- MicroDVD subtitle file (`*.sub`)
- Advanced Substation Alpha subtitles file (`*.ass`)
- Substation Alpha subtitle file (`*.ssa`)

典型的 Weblate 组件配置	
文件掩码	`path/*.srt`
单语言译文模版语言文件	`path/en.srt`
新翻译的译文模版	`path/en.srt`
文件格式	*SubRip subtitle file*

参见:

Subtitles

1.9.29 Excel Open XML

3.2 新版功能.

Excel Open XML (.xlsx) files can be imported and exported.

When uploading XLSX files for translation, be aware that only the active worksheet is considered, and there must be at least a column called `source` (which contains the source string) and a column called `target` (which contains the translation). Additionally there should be the column called `context` (which contains the context path of the translation string). If you use the XLSX download for exporting the translations into an Excel workbook, you already get a file with the correct file format.

1.9.30 HTML files

4.1 新版功能.

注解： Support for this format is currently in beta, feedback from testing is welcome.

The translatable content is extracted from the HTML files and offered for the translation.

参见：

HTML

1.9.31 OpenDocument Format

4.1 新版功能.

注解： Support for this format is currently in beta, feedback from testing is welcome.

The translatable content is extracted from the OpenDocument files and offered for the translation.

参见：

OpenDocument Format

1.9.32 IDML Format

4.1 新版功能.

注解： Support for this format is currently in beta, feedback from testing is welcome.

The translatable content is extracted from the Adobe InDesign Markup Language files and offered for the translation.

1.9.33 其它

Most formats supported by translate-toolkit which support serializing can be easily supported, but they did not (yet) receive any testing. In most cases some thin layer is needed in Weblate to hide differences in behavior of different translate-toolkit storages.

参见：

Translation Related File Formats

1.9.34 只读字符串

3.10 新版功能.

Read-only strings from translation files will be included, but can not be edited in Weblate. This feature is natively supported by few formats (*XLIFF* and *Android string resources*), but can be emulated in others by adding a `read-only` flag, see 定制行为.

1.10 版本控制集成

Weblate currently supports *Git* (with extended support for *GitHub*, *Gerrit* and *Subversion*) and *Mercurial* as version control backends.

1.10.1 Accessing repositories

The VCS repository you want to use has to be accessible to Weblate. With a publicly available repository you just need to enter the correct URL (for example `https://github.com/WeblateOrg/weblate.git`), but for private repositories or for push URLs the setup is more complex and requires authentication.

Accessing repositories from Hosted Weblate

For Hosted Weblate there is a dedicated push user registered on GitHub, Bitbucket, Codeberg and GitLab (with username *weblate* named *Weblate push user*). You need to add this user as a collaborator and give it appropriate permission to your repository (read only is okay for cloning, write is required for pushing). Depending on service and your organization settings, this happens immediately or requires confirmation from Weblate side.

The invitations on GitHub are accepted automatically within five minutes, on other services manual processing might be needed, so please be patient.

Once the *weblate* user is added, you can configure *源代码库* and *代码库推送 URL* using SSH protocol (for example `git@github.com:WeblateOrg/weblate.git`).

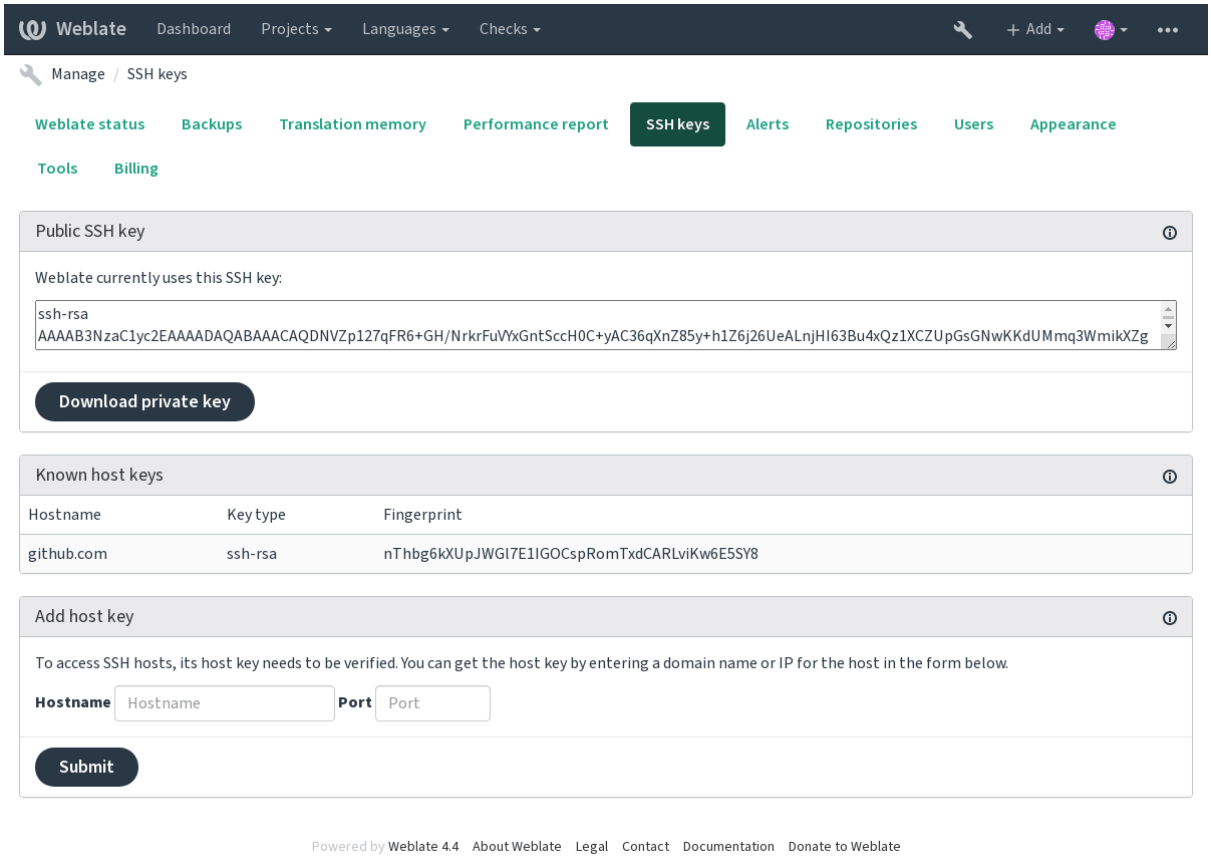
SSH 仓库

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，每个密钥只能添加到一个存储库中，请参阅*GitHub repositories* 和*Accessing repositories from Hosted Weblate*。

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 *About* 页面的所有用户可见。

管理员可以在管理界面登录页面的连接部分（从 *SSH keys*）生成或显示 Weblate 当前使用的公共密钥。

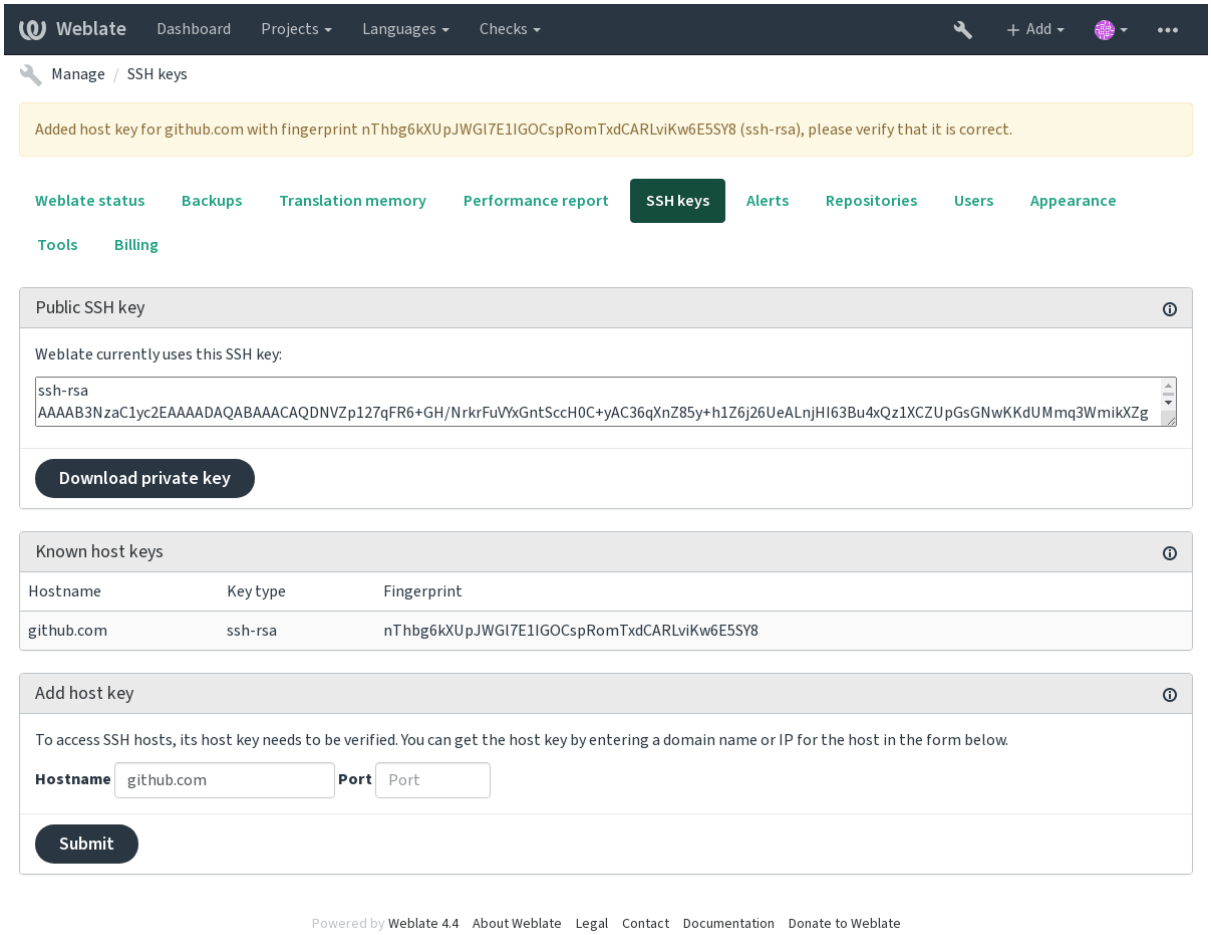
注解： 相应的私有 SSH 密钥当前无法使用密码，因此请确保已受到良好的保护。

提示： 对生成的私有 Weblate SSH 密钥进行备份。

验证 SSH 主机密钥

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

如果要在连接到仓库之前对其进行验证，请在管理界面的同一部分中的 *Add host key* 中验证要访问的服务器的 SSH 主机密钥。输入您要访问的主机名（例如 `gitlab.com`），然后按 *Submit*。验证其指纹与您添加的服务器匹配。它们显示在确认消息中：



GitHub repositories

Access via SSH is possible (see *SSH 仓库*), but in case you need to access more than one repository, you will hit a GitHub limitation on allowed SSH key usage (since one key can be used only for one repository).

In case the 推送分支 is not set, the project is forked and changes pushed through a fork. In case it is set, changes are pushed to the upstream repository and chosen branch.

For smaller deployments, use HTTPS authentication with a personal access token and your GitHub account, see Creating an access token for command-line use.

For bigger setups, it is usually better to create a dedicated user for Weblate, assign it the public SSH key generated in Weblate (see *Weblate SSH 密钥*) and grant it access to all the repositories you want to translate. This approach is also used for Hosted Weblate, there is dedicated *weblate* user for that.

参见:

Accessing repositories from Hosted Weblate

Weblate internal URLs

To share one repository between different components you can use a special URL like `weblate://project/component`. This way, the component will share the VCS repository configuration with the referenced component (`project/component` in the example).

Weblate automatically adjusts repository URL when creating component when it finds component with matching repository setup. You can override this in last step of component configuration.

Reasons to use this:

- Saves disk space on the server, the repository is stored just once.
- Makes the updates faster, only one repository is updated.
- There is just single exported repository with Weblate translations (see *Git 导出器*).
- Some addons can operate on more components sharing single repository, for example *压缩 Git 提交*.

HTTPS repositories

To access protected HTTPS repositories, include the username and password in the URL. Don't worry, Weblate will strip this info when the URL is shown to users (if even allowed to see the repository URL at all).

For example the GitHub URL with authentication added might look like: `https://user:your_access_token@github.com/WeblateOrg/weblate.git`.

注解: If your username or password contains special characters, those have to be URL encoded, for example `https://user%40example.com:%24password%23@bitbucket.org/…`.

Using proxy

If you need to access HTTP/HTTPS VCS repositories using a proxy server, configure the VCS to use it.

This can be done using the `http_proxy`, `https_proxy`, and `all_proxy` environment variables, (as described in the cURL documentation) or by enforcing it in the VCS configuration, for example:

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

注解: The proxy configuration needs to be done under user running Weblate (see also *文件系统权限*) and with `HOME=$DATA_DIR/home` (see *DATA_DIR*), otherwise Git executed by Weblate will not use it.

参见:

The cURL manpage, Git config documentation

1.10.2 Git

参见:

See *Accessing repositories* for info on how to access different kinds of repositories.

Git 强制推送

This behaves exactly like Git itself, the only difference being that it always force pushes. This is intended only in the case of using a separate repository for translations.

> **警告：** Use with caution, as this easily leads to lost commits in your upstream repository.

Customizing Git configuration

Weblate invokes all VCS commands with `HOME=$DATA_DIR/home` (see *DATA_DIR*), therefore editing the user configuration needs to be done in `DATA_DIR/home/.git`.

Git remote helpers

You can also use Git remote helpers for additionally supporting other version control systems, but be prepared to debug problems this may lead to.

At this time, helpers for Bazaar and Mercurial are available within separate repositories on GitHub: git-remote-hg and git-remote-bzr. Download them manually and put somewhere in your search path (for example `~/bin`). Make sure you have the corresponding version control systems installed.

Once you have these installed, such remotes can be used to specify a repository in Weblate.

To clone the `gnuhello` project from Launchpad using Bazaar:

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

For the `hello` repository from selenic.com using Mercurial:

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

> **警告：** The inconvenience of using Git remote helpers is for example with Mercurial, the remote helper sometimes creates a new tip when pushing changes back.

1.10.3 GitHub

2.3 新版功能.

This adds a thin layer atop *Git* using the Github API to allow pushing translation changes as pull requests, instead of pushing directly to the repository.

Git pushes changes directly to a repository, while *GitHub* creates pull requests. The latter is not needed for merely accessing Git repositories.

参见：

推送 Weblate 的更改

Pushing changes to GitHub as pull requests

If not wanting to push translations to a GitHub repository, they can be sent as either one or many pull requests instead.

You need to configure API credentials to make this work.

参见:

GITHUB_USERNAME, *GITHUB_TOKEN*, *GITHUB_CREDENTIALS*

1.10.4 GitLab

3.9 新版功能.

This just adds a thin layer atop *Git* using the GitLab API to allow pushing translation changes as merge requests instead of pushing directly to the repository.

There is no need to use this to access Git repositories, ordinary *Git* works the same, the only difference is how pushing to a repository is handled. With *Git* changes are pushed directly to the repository, while *GitLab* creates merge request.

参见:

推送 Weblate 的更改

Pushing changes to GitLab as merge requests

If not wanting to push translations to a GitLab repository, they can be sent as either one or many merge requests instead.

You need to configure API credentials to make this work.

参见:

GITLAB_USERNAME, *GITLAB_TOKEN*, *GITLAB_CREDENTIALS*

1.10.5 Pagure

4.3.2 新版功能.

This just adds a thin layer atop *Git* using the Pagure API to allow pushing translation changes as merge requests instead of pushing directly to the repository.

There is no need to use this to access Git repositories, ordinary *Git* works the same, the only difference is how pushing to a repository is handled. With *Git* changes are pushed directly to the repository, while *Pagure* creates merge request.

参见:

推送 Weblate 的更改

将作为合并请求的更改推送到 Pagure

If not wanting to push translations to a Pagure repository, they can be sent as either one or many merge requests instead.

You need to configure API credentials to make this work.

参见:

PAGURE_USERNAME, *PAGURE_TOKEN*, *PAGURE_CREDENTIALS*

1.10.6 Gerrit

2.2 新版功能.

Adds a thin layer atop *Git* using the git-review tool to allow pushing translation changes as Gerrit review requests, instead of pushing them directly to the repository.

The Gerrit documentation has the details on the configuration necessary to set up such repositories.

1.10.7 Mercurial

2.1 新版功能.

Mercurial is another VCS you can use directly in Weblate.

注解: It should work with any Mercurial version, but there are sometimes incompatible changes to the command-line interface which breaks Weblate integration.

参见:

See *Accessing repositories* for info on how to access different kinds of repositories.

1.10.8 Subversion

2.8 新版功能.

Weblate uses git-svn to interact with subversion repositories. It is a Perl script that lets subversion be used by a Git client, enabling users to maintain a full clone of the internal repository and commit locally.

注解: Weblate tries to detect Subversion repository layout automatically - it supports both direct URLs for branch or repositories with standard layout (branches/, tags/ and trunk/). More info about this is to be found in the git-svn documentation. If your repository does not have a standard layout and you encounter errors, try including the branch name in the repository URL and leaving branch empty.

在 2.19 版更改: Before this, there was only support for standard layout repositories.

Subversion credentials

Weblate expects you to have accepted the certificate up-front and if needed, your credentials. It will look to insert them into the DATA_DIR directory. Accept the certificate by using *svn* once with the *$HOME* environment variable set to the DATA_DIR:

```
# 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

1.10.9 Local files

3.8 新版功能.

Weblate can also operate without a remote VCS. The initial translations are imported by uploading them. Later you can replace individual files by file upload, or add translation strings directly from Weblate (currently available only for monolingual translations).

In the background Weblate creates a Git repository for you and all changes are tracked in. In case you later decide to use a VCS to store the translations, you already have a repository within Weblate can base your integration on.

1.11 Weblate 的 REST API

2.6 新版功能: 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 framework。你可以直接使用或参考 *Weblate 客户端*。

1.11.1 身份验证和通用参数

公共项目 API 不需要身份验证就可用，尽管没有身份验证的请求导致严重的瓶颈（默认每天 100 个请求），所以推荐使用身份验证。身份验证使用令牌，这可以在你的简介中得到。在 `Authorization` 标头中使用它：

ANY /

对于 API 的普通请求行为，标头、状态编码和参数在这里也应用于所有端点。

查询参数

- **format** –响应格式（覆盖了 Accept）。可能的值依赖于 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。

请求标头

- Accept –相应内容的类型依赖于 Accept 标头
- Authorization –进行身份验证的可选令牌

响应标头

- Content-Type –这依赖于请求的标头 Accept
- Allow –对象允许的 HTTP 方法的列表

响应 JSON 对象

- **detail** (*string*) –失败的详细描述（对于 200 OK 以外的 HTTP 状态编码）
- **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

状态编码

- 200 OK –当请求被正确地处理时
- 400 Bad Request –当缺少表格参数时
- 403 Forbidden –当访问被拒绝时

- 429 Too Many Requests –当出现瓶颈时

身份验证的例子

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，参数可以指定为表格提交（*application/x-www-form-urlencoded*）或 JSON（*application/json*）。

表格请求示例：

```
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 。

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 版更改: 添加速率限制状态的标头。

参见:

频次限制, 频次限制

1.11.2 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/",
```

(下页继续)

(续上页)

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

1.11.3 用户

4.0 新版功能.

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:

用户对象属性被归档在*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）

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
- **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,
    "date_joined": "2020-03-29T18:42:42.617681Z",
```

(下页继续)

(续上页)

```
    "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
- **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
- **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

表格参数

- **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*) –用户上传的数量
- **commented** (*int*) –用户注释的数量
- **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*) –来自可用选择的通知范围
- **frequency** (*int*) –通知的频率选择

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*) –来自可用选择的通知范围
- **frequency** (*int*) –通知的频率选择

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*) –来自可用选择的通知范围
- **frequency** (*int*) –通知的频率选择

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

1.11.4 群组

4.0 新版功能.

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:

群组对象属性归档在*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

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*) –相关联组件的连接；请参见*GET /api/components/(string:project)/(string:component)/*
- **componentlist** (*array*) –相关联组件列表的连接；请参见*GET /api/component-lists/(str:slug)/*

示例 JSON 数据:

```
{
    "name": "Guests",
    "project_selection": 3,
    "language_selection": 1,
    "url": "http://example.com/api/groups/1/",
    "roles": [
        "http://example.com/api/roles/1/",
        "http://example.com/api/roles/2/"
```

(下页继续)

(续上页)

```
    ],
    "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

表格参数

- **string role_id** –唯一的角色 ID

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

将组件与群组关联。

参数

- **id** (*int*) –群组的 ID

表格参数

- **string component_id** –唯一的组件 ID

DELETE /api/groups/(**int:** *id*)**/components/**
int: *component_id* 从群组删除组件。

参数

- **id**(*int*) –群组的 ID
- **component_id**(*int*) –唯一的组件 ID

POST /api/groups/(**int:** *id*)**/projects/**
将项目与群组关联。

参数

- **id**(*int*) –群组的 ID

表格参数

- **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

表格参数

- **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/**
将组件列表与群组关联。

参数

- **id**(*int*) –群组的 ID

表格参数

- **string component_list_id** –唯一的组件列表 ID

DELETE /api/groups/(**int:** *id*)**/componentlists/**
int: *component_list_id* 从群组删除组件列表。

参数

- **id**(*int*) –群组的 ID
- **component_list_id**(*int*) –唯一的组件列表 ID

1.11.5 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:

角色对象属性归档在*GET /api/roles/(int:id)/*。

POST /api/roles/

创建新角色。

参数

- **name** (*string*) –角色名称
- **permissions** (*array*) –权限编码名称的列表

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*) –权限编码名称的列表

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*) –权限编码名称的列表

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*) –权限编码名称的列表

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

删除角色。

参数

- **id** (*int*) –角色的 ID

1.11.6 语言

GET /api/languages/
返回所有语言的列表。

参见:

语言对象属性存档在*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*) –语言方向
- **plural** (*object*) –语言复数形式与数字

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

示例 JSON 数据:

```
{
    "code": "en",
    "direction": "ltr",
    "name": "English",
    "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*) –语言的编码

请求 JSON 对象

- **name** (*string*) –语言名称
- **direction** (*string*) –语言方向
- **plural** (*object*) –语言复数的细节

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

更改语言参数。

参数

- **language** (*string*) –语言的编码

请求 JSON 对象

- **name** (*string*) –语言名称
- **direction** (*string*) –语言方向
- **plural** (*object*) –语言复数的细节

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

删除语言。

参数

- **language** (*string*) –语言的编码

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*) –语言的上一次更改
- **recent_changes** (*int*) –更改的总数
- **translated** (*int*) –已翻译的字符串数量
- **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*) –模糊字符串（标记为需要编辑）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

1.11.7 项目

GET /api/projects/

返回所有项目的列表。

参见:

项目对象的属性存档在*GET /api/projects/(string:project)/*。

POST /api/projects/

3.9 新版功能.

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 标识串

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*) –组件列表的 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*) –启用复查
- **source_review** (*boolean*) –启用来源评论
- **set_language_team** (*boolean*) –设置 *Language-Team* 头
- **enable_hooks** (*boolean*) –启用钩子
- **instructions** (*string*) –翻译说明
- **mail** (*string*) –邮件列表
- **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*)/

4.3 新版功能.

通过 PATCH 请求来编辑一个项目。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

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

4.3 新版功能.

通过 PUT 请求来编辑一个项目。

参数

- **project** (*string*) –项目 URL 标识串

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

3.9 新版功能.

删除项目。

参数

- **project** (*string*) –项目 URL 标识串

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

返回项目更改的列表。这本质上是仔细检查的项目*GET /api/changes/*接收相同的参数。

参数

- **project** (*string*) –项目 URL 标识串

响应 JSON 对象

- **results** (*array*) – 组件对象的矩阵；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 标识串

响应 JSON 对象

- **needs_commit** (*boolean*) –是否有待定的更改要提交
- **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 标识串

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/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
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组件列表。

参数

- **project** (*string*) –项目 URL 标识串

响应 JSON 对象

- **results** (*array*) –组件对象的矩阵；请参见*GET /api/components/(string:project)/(string:component)/*

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

3.9 新版功能.

在 4.3 版更改: 现在对于没有版本控制系统（VCS）的组件接受参数 zipfile 和 docfile，请参见*Local files*。

在给定的项目中新建翻译组件。

提示: 多数组件的新建发生在后台。检查新建组件的 task_url 属性，并按照那里的步骤进行。

参数

- **project** (*string*) –项目 URL 标识串

请求 JSON 对象

- **zipfile** (*file*) –上传到 Weblate 用于翻译初始化的 ZIP 文件
- **docfile** (*file*) –要翻译的文档

响应 JSON 对象

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

CURL 示例：

```
curl \
    --data-binary '{
        "branch": "master",
        "file_format": "po",
        "filemask": "po/*.po",
        "git_export": "",
        "license": "",
        "license_url": "",
        "name": "Weblate",
        "slug": "weblate",
        "repo": "file:///home/nijel/work/weblate-hello",
        "template": "",
        "new_base": "",
        "vcs": "git"
    }' \
    -H "Content-Type: application/json" \
    -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

{
    "branch": "master",
    "file_format": "po",
    "filemask": "po/*.po",
    "git_export": "",
    "license": "",
    "license_url": "",
    "name": "Weblate",
    "slug": "weblate",
    "repo": "file:///home/nijel/work/weblate-hello",
    "template": "",
    "new_base": "",
    "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
```

(下页继续)

(续上页)

```
Content-Language: en
Allow: GET, POST, HEAD, OPTIONS

{
    "branch": "master",
    "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",
            "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。

3.8 新版功能.

参数

- **project** (*string*) –项目 URL 标识串

响应 JSON 对象

- **results** (*array*) –翻译统计数据对象的矩阵
- **language** (*string*) –语言名称
- **code** (*string*) –语言代码
- **total** (*int*) –字符串的总数
- **translated** (*int*) –已翻译的字符串数量
- **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。

3.8 新版功能.

参数

- **project** (*string*) –项目 URL 标识串

响应 JSON 对象

- **total** (*int*) –字符串的总数
- **translated** (*int*) –已翻译的字符串数量
- **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

1.11.8 组件

GET /api/components/
返回翻译组件的列表。

参见:

组件对象属性存档在*GET /api/components/(string:project)/(string:component)/*。

GET /api/components/(string: *project*)/string: *component*/ 返回翻译组件的信息。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

响应 JSON 对象

- **project** (*object*) – 翻译项目；请参见*GET /api/projects/(string:project)/*
- **name** (*string*) –组件名称
- **slug** (*string*) –组件标识串
- **vcs** (*string*) –版本控制系统
- **repo** (*string*) –源代码库
- **git_export** (*string*) –已导出代码库 *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*) –新翻译的译文模版
- **file_format** (*string*) –文件格式
- **license** (*string*) –翻译许可证
- **agreement** (*string*) –贡献者协议
- **new_lang** (*string*) –添加新翻译

- **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*) –代码库推送 *URL*
- **check_flags** (*string*) –翻译标记
- **priority** (*string*) –优先权
- **enforced_checks** (*string*) –强制检查
- **restricted** (*string*) –受限制的访问
- **repoweb** (*string*) –代码库浏览器
- **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、合并以及插件消息
- **add_message** (*string*) –提交、添加、删除、合并以及插件消息
- **delete_message** (*string*) –提交、添加、删除、合并以及插件消息
- **merge_message** (*string*) –提交、添加、删除、合并以及插件消息
- **addon_message** (*string*) –提交、添加、删除、合并以及插件消息
- **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*) –对变更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": "master",
    "file_format": "po",
    "filemask": "po/*.po",
    "git_export": "",
```

（下页继续）

(续上页)

```
    "license": "",
    "license_url": "",
    "name": "Weblate",
    "slug": "weblate",
    "project": {
        "name": "Hello",
        "slug": "hello",
        "source_language": {
            "code": "en",
            "direction": "ltr",
            "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",
        "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 请求编辑一个组件。

参数

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

请求 JSON 对象

- **name** (*string*) –组件名称
- **slug** (*string*) –组件的标识串
- **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
```

(下页继续)

(续上页)

```
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": "master",
    "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",
            "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/"
}
```

PUT /api/components/(string: *project*)/string: *component*/ 通过 PUT 请求编辑一个组件。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

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*) –组件名称
- **slug** (*string*) –组件的标识串
- **repo** (*string*) –版本控制系统（VCS）仓库的 URL
- **template** (*string*) –但语言翻译的译文模板文件
- **new_base** (*string*) –用于添加新翻译的译文模板文件
- **vcs** (*string*) –版本控制系统

DELETE /api/components/(**string:** *project*)**/**
string: *component***/** 3.9 新版功能.

删除组件。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

GET /api/components/(**string:** *project*)**/**
string: *component***/changes/** 返回组件更改的列表。这本质上是仔细检查的组件:http:get:'/api/changes/'接相同参数。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

响应 JSON 对象

- **results** (*array*) – 组件对象的矩阵；请参见*GET /api/changes/(int:id)/*

GET /api/components/(**string:** *project*)**/**
string: *component***/screenshots/** 返回组件屏幕截图的列表。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

响应 JSON 对象

- **results** (*array*) –组件屏幕截图的矩阵；请参见*GET /api/screenshots/(int:id)/*

GET /api/components/(**string:** *project*)**/**
string: *component***/lock/** 返回组件锁定状态。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

响应 JSON 对象

- **locked** (*boolean*) –组件是否因更新而锁定

示例 JSON 数据：

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

POST /api/components/(string: *project*)/
string: *component*/lock/ 设置组件锁定状态。

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

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

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/* 的相同。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

响应 JSON 对象

- **needs_commit** (*boolean*) –是否有待定的更改要提交
- **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*) –远程提交信息
- **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 标识串
- **component** (*string*) –组件 URL 标识串

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/ 为单语言翻译下载译文模板文件。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

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 标识串
- **component** (*string*) –组件 URL 标识串

GET /api/components/(string: *project*)/
string: *component*/translations/ 返回给定组件中翻译对象的列表。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

响应 JSON 对象

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

POST /api/components/(string: *project*)/
string: *component*/translations/ 在给定组件中新建新的翻译。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

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,
```

(下页继续)

(续上页)

```
    "fuzzy_percent": 0.0,
    "fuzzy_words": 0,
    "have_comment": 0,
    "have_suggestion": 0,
    "is_template": false,
    "is_source": false,
    "language": {
        "code": "cs",
        "direction": "ltr",
        "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/ 对组件内所有的翻译返回分页的统计数据。

2.7 新版功能.

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串

响应 JSON 对象

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

1.11.9 翻译

GET /api/translations/
返回翻译的列表。

参见:

翻译对象属性存档在*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 标识串
- **component** (*string*) –组件 URL 标识串
- **language** (*string*) –翻译语言代码

响应 JSON 对象

- **component** (*object*) – 组件对象；请参见*GET /api/components/(string:project)/(string:component)/*
- **failing_checks** (*int*) –未通过检查的字符串数目
- **failing_checks_percent** (*float*) –未通过检查的字符串比重
- **failing_checks_words** (*int*) –未通过检查的单词数目
- **filename** (*string*) –翻译文件名
- **fuzzy** (*int*) –模糊字符串（标记为需要编辑）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*) –带有注释的字符串数量
- **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*) –最后更改的时间标签
- **revision** (*string*) –文件的修订哈希值
- **share_url** (*string*) –用于分享导向约定页面的 URL
- **total** (*int*) –字符串的总数
- **total_words** (*int*) –词的总数
- **translate_url** (*string*) –翻译的 URL
- **translated** (*int*) –已翻译的字符串数量
- **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": "master",
        "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",
                "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",
        "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",
        "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*/ 3.9 新版功能.

删除翻译。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串
- **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。这本质上是仔细检查的翻译:http:get:'/api/changes/'接受相同参数。

参数

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

响应 JSON 对象

- **results** (*array*) – 组件对象的矩阵；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 标识串
- **component** (*string*) –组件 URL 标识串
- **language** (*string*) –翻译语言代码
- **q** (*string*) –搜索查询字符串:ref:'Searching'（可选）

响应 JSON 对象

- **results** (*array*) –组件对象的矩阵；请参见*GET /api/units/(int:id)/*

POST /api/translations/(string: *project*)/
string: *component*/string: *language*/units/ 添加新的单语言单元。

参数

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

请求 JSON 对象

- **key** (*string*) –翻译单元的名称
- **value** (*string*) –翻译单元值

POST /api/translations/(string: *project*)/
string: *component*/string: *language*/autotranslate/ 触发自动翻译。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串
- **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*) –自动翻译来源
- **component** (*string*) –为项目打开对共享翻译记忆库的贡献，以访问其他组件。
- **engines** (*string*) –机器翻译引擎
- **threshold** (*string*) –匹配分数阈值

GET /api/translations/(string: *project*)/string: *component*/string: *language*/file/ 下载当前翻译文件存储在版本控制系统（VCS）中（没有 `format` 参数），或转换为标准搁置（当前支持：Gettext PO、MO、XLIFF 和 TBX）。

注解： 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`, `csv`, `xlsx`, `json`, `aresource`, `strings`

参数

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

表格参数

- **string conflicts** –如何处理冲突（`ignore`, `replace-translated` or `replace-approved`）
- **file file** –上传文件
- **string email** –作者邮箱
- **string author** –作者姓名
- **string method** –上传方法（`translate`, `approve`, `suggest`, `fuzzy`, `replace`, `source`），请参见*Import methods*
- **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:http:get:'/api/components/(string:project)/(string:component)/repository/'的相同。

参数

- **project** (*string*) –项目 URL 标识串
- **component** (*string*) –组件 URL 标识串
- **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 标识串
- **component** (*string*) –组件 URL 标识串
- **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。

2.7 新版功能.

参数

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

响应 JSON 对象

- **code** (*string*) –语言代码
- **failing** (*int*) –检查失败的数量
- **failing_percent** (*float*) –检查失败的百分比
- **fuzzy** (*int*) –模糊字符串（标记为需要编辑）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*) –上次更改的日期
- **name** (*string*) –语言名称
- **total** (*int*) –字符串的总数
- **translated** (*int*) –已翻译的字符串数量

- **translated_percent** (*float*) –已翻译字符串的百分比
- **url** (*string*) –访问翻译的 URL （约定的 URL）
- **url_translate** (*string*) –访问翻译的 URL （真实翻译的 URL）

1.11.10 单元

2.10 新版功能.

GET /api/units/

返回翻译单元的列表。

参见:

单元对象属性存档在*GET /api/units/(int:id)/*。

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

在 4.3 版更改: `target` 和 `source` 现在是矩阵，来适当的处理多个字符串。

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*) –单元的唯一识别文字
- **content_hash** (*string*) –源字符串的唯一识别文字
- **location** (*string*) –源代码中单元的位置
- **context** (*string*) –翻译单元的语境
- **note** (*string*) –翻译单元的注解
- **flags** (*string*) –翻译单元的标记
- **state** (*int*) –单元状态，0——未翻译，10——需要编辑，20——已翻译，30——以通过，100——只读
- **fuzzy** (*boolean*) –是否该单元是模糊的或标记为需要检查
- **translated** (*boolean*) –单元是否被翻译
- **approved** (*boolean*) –翻译是否被核准
- **position** (*int*) –翻译文件中的单元位置
- **has_suggestion** (*boolean*) –单元是否有翻译建议
- **has_comment** (*boolean*) –单元是否有注释
- **has_failing_check** (*boolean*) –单元是否有未通过检查的翻译
- **num_words** (*int*) –源词汇的数量
- **priority** (*int*) –翻译优先级；100 为默认值
- **id** (*int*) –单元识别问题
- **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，请参见定制行为
- **web_url** (*string*) –单元可以被编辑的 URL
- **souce_unit** (*string*) –源单元链接；请参见*GET /api/units/(int:id)/*

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

4.3 新版功能.

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 - 已批准 (需启用审核工作流, 见专门的审核者)
- **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，请参见定制行为

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

4.3 新版功能.

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 - 已批准 (需启用审核工作流, 见专门的审核者)
- **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，请参见定制行为

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

4.3 新版功能.

删除一个翻译单元。

参数

- **id** (*int*) –单元 ID

1.11.11 修改

2.10 新版功能.

GET /api/changes/

在 4.1 版更改: 更改的筛选在 4.1 版本引入。

返回翻译更改的列表。

参见:

更改对象的属性存档在*GET /api/changes/(int:id)/*。

查询参数

- **user** (*string*) –筛选用户的用户名
- **action** (*int*) –筛选的动作，可以多次使用
- **timestamp_after** (*timestamp*) –ISO 8601 格式的时间标签，列出此时间之后的更改
- **timestamp_before** (*timestamp*) –ISO 8601 格式的时间标签，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*) –相关组件对象的 URL
- **glossary_term** (*string*) –相关术语表对象的 URL
- **user** (*string*) –相关用户对象的 URL
- **author** (*string*) –相关作者对象的 URL
- **timestamp** (*timestamp*) –时间的时间标签
- **action** (*int*) –动作的几种识别
- **action_name** (*string*) –动作的文本描述
- **target** (*string*) –更改的事件的文本或细节
- **id** (*int*) –更改的识别文字

1.11.12 截图

2.14 新版功能.

GET /api/screenshots/

返回屏幕截图字符串信息的列表。

参见:

屏幕截图对象的属性存档在*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*) –屏幕截图的 ID

响应 JSON 对象

- **name** (*string*) –屏幕截图的名称
- **component** (*string*) –相关组件对象的 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*) –屏幕截图的 ID

POST /api/screenshots/(int: *id*)/file/

替换屏幕截图。

参数

- **id** (*int*) –屏幕截图的 ID

表格参数

- **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*) –屏幕截图的 ID

表格参数

- **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*) –屏幕截图的 ID
- **unit_id** –源字符串单元 ID

POST /api/screenshots/

新建新的屏幕截图。

表格参数

- **file image** –上传文件
- **string name** –截图名称
- **string project_slug** –项目标识串
- **string component_slug** –组件标识串
- **string language_code** –语言代码

响应 JSON 对象

- **name** (*string*) –屏幕截图的名称
- **component** (*string*) –相关组件对象的 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*) –屏幕截图的 ID

响应 JSON 对象

- **name** (*string*) –屏幕截图的名称
- **component** (*string*) –相关组件对象的 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*) –屏幕截图的 ID

响应 JSON 对象

- **name** (*string*) –屏幕截图的名称
- **component** (*string*) –相关组件对象的 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*) –屏幕截图的 ID

1.11.13 附加组件

4.4.1 新版功能.

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*) –组件名称
- **component** (*string*) –相关组件对象的 URL
- **configuration** (*object*) –最优附加组件配置

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*) –组件标识串

请求 JSON 对象

- **name** (*string*) –组件名称
- **configuration** (*object*) –最优附加组件配置

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*) –最优附加组件配置

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*) –最优附加组件配置

DELETE /api/addons/(int: *id*)/
删除附加组件。

参数

- **id** (*int*) –附加组件 ID

1.11.14 组件列表

4.0 新版功能.

GET /api/component-lists/
返回组件列表的列表。

参见:

组件列表对象属性存档在*GET /api/component-lists/(str:slug)/*。

GET /api/component-lists/(str: *slug*)/
返回组件列表的信息。

参数

- **slug** (*string*) –组件列表的标识串

响应 JSON 对象

- **name** (*string*) –组件列表的名称
- **slug** (*string*) –组件列表的表示串
- **show_dashboard** (*boolean*) –是否在控制台上显示
- **components** (*array*) –相关联组件的连接；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*)/
更改组件列表参数。

参数

- **slug** (*string*) –组件列表的标识串

请求 JSON 对象

- **name** (*string*) –组件列表的名称
- **slug** (*string*) –组件列表的表示串
- **show_dashboard** (*boolean*) –是否在控制台上显示

PATCH /api/component-lists/(str: *slug*)/
更改组件列表参数。

参数

- **slug** (*string*) –组件列表的标识串

请求 JSON 对象

- **name** (*string*) –组件列表的名称
- **slug** (*string*) –组件列表的表示串
- **show_dashboard** (*boolean*) –是否在控制台上显示

DELETE /api/component-lists/(str: *slug*)/
删除组件列表。

参数

- **slug** (*string*) –组件列表的标识串

POST /api/component-lists/(str: *slug*)/components/
使组件与组件列表相关。

参数

- **slug** (*string*) –组件列表的标识串

表格参数

- **string component_id** –组件 ID

DELETE /api/component-lists/(str: *slug*)/components/
str: *component_slug* 将组件与组件列表接触相关性。

参数

- **slug** (*string*) –组件列表的标识串
- **component_slug** (*string*) –组件标识串

1.11.15 词汇表

GET /api/glossary/
将与用户访问的项目相关的所有词汇表的列表返回。

参见:

语言对象属性存档在*GET /api/languages/(string:language)/*。

GET /api/glossary/(int: *id*)/
返回关于一条词汇表的信息。

参数

- **id** (*int*) –词汇表 id

响应 JSON 对象

- **name** (*string*) –语言代码
- **color** (*string*) –文字方向
- **source_language** (*object*) –语言复数信息的对象
- **projects** (*array*) –相关联项目的连接；请参见*GET /api/projects/(string:project)/*

示例 JSON 数据:

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

(下页继续)

(续上页)

```
    },
    "project": {
        "name": "Hello",
        "slug": "hello",
        "id": 1,
        "source_language": {
            "code": "en",
            "name": "English",
            "plural": {
                "id": 75,
                "source": 0,
                "number": 2,
                "formula": "n != 1",
                "type": 1
            },
            "aliases": [
                "english",
                "en_en",
                "base",
                "source",
                "eng"
            ],
            "direction": "ltr",
            "web_url": "http://example.com/languages/en/",
            "url": "http://example.com/api/languages/en/",
            "statistics_url": "http://example.com/api/languages/en/statistics/"
        },
        "web_url": "http://example.com/projects/demo1/",
        "url": "http://example.com/api/projects/demo1/",
        "components_list_url": "http://example.com/api/projects/demo1/
↪components/",
        "repository_url": "http://example.com/api/projects/demo1/repository/",
        "statistics_url": "http://example.com/api/projects/demo1/statistics/",
        "changes_list_url": "http://example.com/api/projects/demo1/changes/",
        "languages_url": "http://example.com/api/projects/demo1/languages/"
    },
    "projects_url": "http://example.com/api/glossary/7/projects/",
    "terms_url": "http://example.com/api/glossary/7/terms/",
    "url": "http://example.com/api/glossary/7/"
}
```

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

更改词汇表参数。

参数

- **id** (*int*) –词汇表 id

请求 JSON 对象

- **name** (*string*) –语言名称
- **color** (*string*) –语言方向
- **source_language** (*object*) –语言复数的细节

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

更改词汇表参数。

参数

- **id** (*int*) –词汇表 id

请求 JSON 对象

- **name** (*string*) –语言名称

- **color** (*string*) –语言方向
- **source_language** (*object*) –语言复数的细节

DELETE /api/glossary/(**int:** *id*)**/**
删除该术语。

参数

- **id** (*int*) –词汇表 id

GET /api/glossary/(**int:** *id*)**/projects/**
返回与词汇表连接的项目。

参数

- **id** (*int*) –词汇表 id

响应 JSON 对象

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

POST /api/glossary/(**int:** *id*)**/projects/**
将项目与词汇表相关。

参数

- **id** (*int*) –词汇表 id

表格参数

- **string project_slug** –项目标识串

DELETE /api/glossary/(**int:** *id*)**/projects/**
删除项目与词汇表的关联性。

参数

- **id** (*int*) –词汇表 id

表格参数

- **string project_slug** –项目标识串

GET /api/glossary/(**int:** *id*)**/terms/**
列出词汇表的项。

参数

- **id** (*int*) –词汇表 id

POST /api/glossary/(**int:** *id*)**/terms/**
将术语与词汇表关联。

参数

- **id** (*int*) –词汇表 id

请求 JSON 对象

- **language** (*object*) –术语的语言
- **source** (*string*) –术语的源字符串
- **target** (*string*) –术语的目标字符串

GET /api/glossary/(**int:** *id*)**/terms/**
int: *term_id***/** 获取与词汇表相关联的术语。

参数

- **id** (*int*) –词汇表 id
- **term_id** (*int*) –术语的 ID

PUT /api/glossary/(int: *id*)/terms/
int: *term_id*/ 编辑与词汇表关联的术语。

参数

- **id** (*int*) –词汇表 id
- **term_id** (*int*) –术语的 ID

请求 JSON 对象

- **language** (*object*) –术语的语言
- **source** (*string*) –术语的源字符串
- **target** (*string*) –术语的目标字符串

PATCH /api/glossary/(int: *id*)/terms/
int: *term_id*/ 编辑与词汇表关联的术语。

参数

- **id** (*int*) –词汇表 id
- **term_id** (*int*) –术语的 ID

请求 JSON 对象

- **language** (*object*) –术语的语言
- **source** (*string*) –术语的源字符串
- **target** (*string*) –术语的目标字符串

DELETE /api/glossary/(int: *id*)/terms/
int: *term_id*/ 删除与词汇表相关的术语。

参数

- **id** (*int*) –词汇表 id
- **term_id** (*int*) –术语的 ID

1.11.16 任务

4.4 新版功能.

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

1.11.17 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、组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*/ 2.6 版后已移除: 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。

触发组件的更新（从版本控制系统 VCS 拉取并扫描翻译的更改）。

GET /hooks/update/(string: *project*)/
2.6 版后已移除: 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。

触发项目中所有组件的更新（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组件的特殊钩子。

注解: 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/free-pro-team@latest/github/extending-github/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组件的特殊钩子。

参见:

从 *GitLab* 自动接收更改 关于设置 GitLab 集成的指示

https://docs.gitlab.com/ce/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组件的特殊钩子。

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/
3.3 新版功能.

处理 Pagure 通知并自动更新匹配的组件的特殊钩子。

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/

3.8 新版功能.

处理 Azure Repos 通知并自动更新匹配的组件的特殊钩子。

参见:

从 *Azure Repos* 自动接收更改 关于设置 Azure 集成的指示

https://docs.microsoft.com/en-us/azure/devops/service-hooks/services/webhooks?view=azure-devops 关于 Azure Repos Web Hooks 的一般信息

ENABLE_HOOKS 关于对整个 Weblate 启动钩子

POST /hooks/gitea/

3.9 新版功能.

处理 Gitea Webhook 通知并自动更新匹配的组件的特殊钩子。

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/

3.9 新版功能.

处理 Gitee Webhook 通知并自动更新匹配的组件的特殊钩子。

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

1.11.18 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`

2.6 版后已移除: 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。

为给定的组件以给定的格式检索统计数据。

请求的示例:

```
GET /exports/stats/weblate/master/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

响应的示例：

```
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/master/cs/
↪"
    },
    {
        "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/master/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/",
```

(下页继续)

(续上页)

```
        "url_translate": "http://hosted.weblate.org/projects/weblate/master/el/
↪"
    }
]
```

1.11.19 RSS 频道

翻译的更改导出到 RSS 频道。

GET /exports/rss/(**string:** *project*)**/**
string: *component*/**string:** *language*/ 用翻译近期的更改检索 RSS 频道。

GET /exports/rss/(**string:** *project*)**/**
string: *component*/ 用组件的近期更改检索 RSS 频道。

GET /exports/rss/(**string:** *project*)**/**
用项目的近期更改检索 RSS 频道。

GET /exports/rss/language/(**string:** *language*)**/**
用语言的近期更改检索 RSS 频道。

GET /exports/rss/
用 Weblate 事件的近期更改检索 RSS 频道。

参见:

RSS on wikipedia

1.12 Weblate 客户端

2.7 新版功能: 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。

1.12.1 安装

Weblate 客户端是分开上市的，包括 Python 模块。要使用下面的命令，您需要安装`wlc`:

```
pip3 install wlc
```

1.12.2 Docker 用法

The Weblate Client is also available as a Docker image.

The image is published on Docker Hub: https://hub.docker.com/r/weblate/wlc

正在安装:

```
docker pull weblate/wlc
```

The Docker container uses Weblate’s default settings and connects to the API deployed in localhost. The API URL and API_KEY can be configured through the arguments accepted by Weblate.

The command to launch the container uses the following syntax:

```
docker run --rm weblate/wlc [WLC_ARGS]
```

例:

```
docker run --rm weblate/wlc --url https://hosted.weblate.org/api/ list-projects
```

You might want to pass your 配置文件 to the Docker container, the easiest approach is to add your current directory as `/home/weblate` volume:

```
docker run --volume $PWD:/home/weblate --rm weblate/wlc show
```

1.12.3 入门

wlc 配置存储在 `~/.config/weblate` 中（其他位置参见 ref:*wlc-config*），请创建它以匹配您的环境：

```
[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

1.12.4 概要

```
wlc [arguments] <command> [options]
```

命令实际上指示应该执行哪个操作。

1.12.5 描述

Weblate 客户端是一个 Python 库和命令行实用程序，可使用 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 中的组件。

list-translations
列出 Weblate 中的翻译。

show
显示 Weblate 对象（翻译，组件或项目）。

ls
列出 Weblate 对象（翻译，组件或项目）。

commit
提交在 Weblate 对象（翻译，组件或项目）中所做的更改。

pull
拉取远程仓库的更改到 Weblate 对象中（翻译，组件或项目）。

push
将 Weblate 对象更改推送到远程仓库（翻译，组件或项目）。

reset
0.7 新版功能: 自 wlc 0.7 起受支持。

重置 Weblate 对象中的更改以匹配远程存储库（翻译，组件或项目）。

cleanup
0.9 新版功能: 从 wlc 0.9 开始受支持。

删除 Weblate 对象中所有未跟踪的更改以匹配远程仓库（翻译，组件或项目）。

repo
显示给定 Weblate 对象（翻译，组件或项目）的仓库状态。

statistics
显示给定 Weblate 对象（翻译，组件或项目）的详细统计数据。

lock-status
0.5 新版功能: 自 wlc 0.5 起受支持。

显示锁定状态。

lock
0.5 新版功能: 自 wlc 0.5 起受支持。

锁定组件以防止在 Weblate 中进一步翻译。

unlock
0.5 新版功能: 自 wlc 0.5 起受支持。

解锁 Weblate 组件的翻译。

changes
0.7 新版功能: 从 wlc 0.7 和 Weblate 2.10 开始受支持。

显示给定对象的更改。

download

0.7 新版功能: 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

0.9 新版功能: 从 wlc 0.9 开始受支持。

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。

提示: You can get more detailed information on invoking individual commands by passing `--help`, for example: `wlc ls --help`.

1.12.6 配置文件

.weblate, .weblate.ini, weblate.ini 在 1.6 版更改: The files with *.ini* extension are accepted as well.

每个项目的配置文件

C:\Users\NAME\AppData\weblate.ini 1.6 新版功能.

在 Windows 上使用配置文件。

~/.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——组件或项目。

配置文件是一个 INI 文件，例如:

```
[weblate]
url = https://hosted.weblate.org/api/
key = APIKEY
translation = weblate/master
```

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。

1.12.7 例子

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/
```

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/master

$ wlc show
branch: master
file_format: po
source_language: en
filemask: weblate/locale/*/LC_MESSAGES/django.po
git_export: https://hosted.weblate.org/git/weblate/master/
license: GPL-3.0+
license_url: https://spdx.org/licenses/GPL-3.0+
name: master
new_base: weblate/locale/django.pot
project: weblate
repo: git://github.com/WeblateOrg/weblate.git
slug: master
template:
url: https://hosted.weblate.org/api/components/weblate/master/
vcs: git
web_url: https://hosted.weblate.org/projects/weblate/master/
```

通过此设置，可以轻松地提交当前项目中待定的更改：

```
$ wlc commit
```

1.13 Weblate’s Python API

1.13.1 安装

The Python API is shipped separately, you need to install the *Weblate 客户端*: (wlc) to have it.

```
pip install wlc
```

1.13.2 wlc

WeblateException

exception wlc.**WeblateException**

Base class for all exceptions.

Weblate

class wlc.**Weblate**(*key=''*, *url=None*, *config=None*)

参数

- **key** (*str*) –User key
- **url** (*str*) –API server URL, if not specified default is used
- **config** (wlc.config.WeblateConfig) –Configuration object, overrides any other parameters.

Access class to the API, define API key and optionally API URL.

get(*path*)

参数 **path** (*str*) –Request path

返回类型 object

Performs a single API GET call.

post(*path*, ***kwargs*)

参数 **path** (*str*) –Request path

返回类型 object

Performs a single API GET call.

1.13.3 wlc.config

WeblateConfig

class wlc.config.**WeblateConfig**(*section='wlc'*)

参数 **section** (*str*) –Configuration section to use

Configuration file parser following XDG specification.

load(*path=None*)

参数 **path** (*str*) –Path from which to load configuration.

Loads configuration from a file, if none is specified, it loads from the *wlc* configuration file (`~/.config/wlc`) placed in your XDG configuration path (`/etc/xdg/wlc`).

1.13.4 wlc.main

`wlc.main.main`(*settings=None*, *stdout=None*, *args=None*)

参数

- **settings** (*list*) –Settings to override as list of tuples
- **stdout** (*object*) –stdout file object for printing output, uses `sys.stdout` as default
- **args** (*list*) –Command-line arguments to process, uses `sys.args` as default

Main entry point for command-line interface.

`@wlc.main.register_command`(*command*)

Decorator to register *Command* class in main parser used by *main()*.

Command

class `wlc.main.Command`(*args*, *config*, *stdout=None*)

Main class for invoking commands.

CHAPTER 2

Administrator docs

2.1 配置手册

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 应该可以在所有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：

- 2 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。对于数百个翻译组件，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。

注解： 根据 Weblate 中管理的翻译大小，安装 Weblate 的实际要求差异很大。

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 存储库：

```
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
```

注解: 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。

提供的例子使 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 版更改: 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 版更改: 在 2019 年 7 月（从 3.7.1-6 标签开始）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

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

3.8-3 新版功能.

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;

    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:
```

（下页继续）

(续上页)

```
      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。

您可以通过坚持使用现有的 docker-compose，并且只是拉取最新映像，然后重新启动，来执行此操作：

```
docker-compose stop
docker-compose pull
docker-compose up
```

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。

注解： Weblate 不支持跨 3.0 的升级。如果您使用的是 2.x 系列，并且要升级到 3.x，请首先升级到最新的 3.0.1-x（在撰写本文时为 `3.0.1-7`）镜像，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，请注意 PostgreSQL 版本的更改，因为升级数据库并不容易，请参见 GitHub issue 以获取更多信息。

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*

Docker 环境变量

可以使用环境变量在 Docker 容器中设置许多 Weblate 的配置：

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_SITE_TITLE

更改所有页面页眉上显示的站点标题。

WEBLATE_SITE_DOMAIN

配置网站域名.

提示: 在没有设置的情况下，使用 *WEBLATE_ALLOWED_HOSTS* 的第一项。

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_NAME*, *WEBLATE_ADMIN_EMAIL*

WEBLATE_SERVER_EMAIL

WEBLATE_DEFAULT_FROM_EMAIL

配置外发电子邮件的地址。

参见:

配置电子邮件发送的设置

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* 创建新帐户的身份验证方法。

示例:

```
environment:
  WEBLATE_REGISTRATION_OPEN: 0
  WEBLATE_REGISTRATION_ALLOW_BACKENDS: azuread-oauth2,azuread-tenant-
↪oauth2
```

WEBLATE_TIME_ZONE

在 Weblate 中配置使用的时区，请参阅 `TIME_ZONE`。

注解: 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。

注解: 这不会使 Weblate 容器接受 HTTPS 连接，您同样需要配置它，例子请参见*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_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* 。

注解: 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 Webate 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 *GOOGLE_ANALYTICS_ID* 来配置用于 Google Analytics 的 ID 。

WEBLATE_GITHUB_USERNAME

通过更改 *GITHUB_USERNAME* 来配置用于 GitHub 拉取请求的 GitHub 用户名。

参见:

GitHub

WEBLATE_GITHUB_TOKEN

4.3 新版功能.

通过更改*GITHUB_TOKEN*，为 Github 通过 API 的拉取请求来配置 GitHub 的个人访问令牌。

参见:

GitHub

WEBLATE_GITLAB_USERNAME

通过更改*GITLAB_USERNAME* 来配置用于 GitLab 合并请求的 GitLab 用户名

参见:

GitLab

WEBLATE_GITLAB_TOKEN

通过更改*GITLAB_TOKEN* ，为 GitLab 通过 API 的合并请求来配置 GitLab 的个人访问令牌

参见:

GitLab

WEBLATE_PAGURE_USERNAME

通过更改 PARGUE_USERNAME 来配置用于 Pagure 合并请求的 Pagure 用户名

参见:

Pagure

WEBLATE_PAGURE_TOKEN

通过更改*PAGURE_TOKEN* ，为通过 API 的 Pagure 合并请求配置 Pagure 个人访问令牌

参见:

Pagure

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

为新组件的*受限制的访问* 配置默认值，请参见*DEFAULT_RESTRICTED_COMPONENT* 。

WEBLATE_DEFAULT_TRANSLATION_PROPAGATION

为新组件的*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_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:

签署 GnuPG 的 Git 承诺

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_HIDE_VERSION

配置*HIDE_VERSION*。

WEBLATE_BASIC_LANGUAGES

配置*BASIC_LANGUAGES*.

机器翻译设置

WEBLATE_MT_APERTIUM_APY

启用*Apertium* 机器翻译并设置*MT_APERTIUM_APY*

WEBLATE_MT_AWS_REGION

WEBLATE_MT_AWS_ACCESS_KEY_ID

WEBLATE_MT_AWS_SECRET_ACCESS_KEY

配置*AWS* 机器翻译。

```
environment:
  WEBLATE_MT_AWS_REGION: us-east-1
  WEBLATE_MT_AWS_ACCESS_KEY_ID: AKIAIOSFODNN7EXAMPLE
  WEBLATE_MT_AWS_SECRET_ACCESS_KEY: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

WEBLATE_MT_DEEPL_KEY

允许*DeepL* 机器翻译，并设置*MT_DEEPL_KEY*

WEBLATE_MT_DEEPL_API_VERSION

配置使用的*DeepL* API 版本，请参见*MT_DEEPL_API_VERSION* 。

WEBLATE_MT_GOOGLE_KEY

允许*谷歌翻译* 并设置*MT_GOOGLE_KEY*

WEBLATE_MT_MICROSOFT_COGNITIVE_KEY

允许*微软认知服务翻译工具* 并设置*MT_MICROSOFT_COGNITIVE_KEY*

WEBLATE_MT_MICROSOFT_ENDPOINT_URL

设置*MT_MICROSOFT_ENDPOINT_URL* ，请注意，这应该只包含域名。

WEBLATE_MT_MICROSOFT_REGION

设置*MT_MICROSOFT_REGION*

WEBLATE_MT_MICROSOFT_BASE_URL

设置*MT_MICROSOFT_BASE_URL*

WEBLATE_MT_MODERNMT_KEY

允许*ModernMT* 并设置*MT_MODERNMT_KEY* 。

WEBLATE_MT_MYMEMORY_ENABLED

允许*MyMemory* 机器翻译，并将*MT_MYMEMORY_EMAIL* 设置到 *WEBLATE_ADMIN_EMAIL* 。

示例：

```
environment:
  WEBLATE_MT_MYMEMORY_ENABLED: 1
```

WEBLATE_MT_GLOSBE_ENABLED

允许*Glosbe* 机器翻译。

```
environment:
  WEBLATE_MT_GLOSBE_ENABLED: 1
```

WEBLATE_MT_MICROSOFT_TERMINOLOGY_ENABLED

允许*微软术语服务* 机器翻译。

```
environment:
  WEBLATE_MT_MICROSOFT_TERMINOLOGY_ENABLED: 1
```

WEBLATE_MT_SAP_BASE_URL

WEBLATE_MT_SAP_SANDBOX_APIKEY

WEBLATE_MT_SAP_USERNAME

WEBLATE_MT_SAP_PASSWORD

WEBLATE_MT_SAP_USE_MT

配置*SAP 翻译中心* 机器翻译。

```
environment:
    WEBLATE_MT_SAP_BASE_URL: "https://example.hana.ondemand.com/translationhub/
↪api/v1/"
    WEBLATE_MT_SAP_USERNAME: "user"
    WEBLATE_MT_SAP_PASSWORD: "password"
    WEBLATE_MT_SAP_USE_MT: 1
```

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_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。

直接绑定的例子：

```
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
```

搜索与绑定的例子：

```
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
```

联合搜索与绑定的例子：

```
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
```

针对活动目录（Active Directory）的搜索与绑定的例子：

```
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

允许*GitHub 身份验证*。

Bitbucket

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* 。

Azure Active Directory

WEBLATE_SOCIAL_AUTH_AZUREAD_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_AZUREAD_OAUTH2_SECRET
允许 Azure 活动目录身份验证，请参见*微软 Azure Active Directory* 。

带有租户支持的 Azure 活动目录

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_KEY

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_SECRET

WEBLATE_SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_TENANT_ID
允许带有租户支持的 Azure 活动目录身份验证，请参见*微软 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
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* 。

SAML

在第一次启动容器时自动产生自签名的 SAML 密钥。在您想要使用自己的密钥的情况下，将证书和私钥放置在 `/app/data/ssl/saml.crt` 和:file:'/app/data/ssl/saml.key'中。

WEBLATE_SAML_IDP_ENTITY_ID

WEBLATE_SAML_IDP_URL

WEBLATE_SAML_IDP_X509CERT
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。

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_USER
PostgreSQL 用户名。

POSTGRES_DATABASE
PostgreSQL 数据库名。

POSTGRES_HOST
PostgreSQL 服务器主机名成或 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*。

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 事例。

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_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，您需要提供一个邮件服务器。

TLS 配置例子:

```
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
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
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_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*。

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*。

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

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

4.2.1 新版功能.

: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
```

注解： 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*

更改允许的 app 、检查、插件或自动修复

3.8-5 新版功能.

可以通过后面的变量来调整允许的检查、插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

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`）。

并发 worker 的数量默认与处理的数量匹配（除了被认为只运行一次的备份 worker）。

示例：

```
environment:
  CELERY_MAIN_OPTIONS: --concurrency 16
```

参见：

Celery worker options, *使用 Celery 的后台任务*

UWSGI_WORKERS

配置应该执行多少个 uWSGI workers 。

默认为处理器的数量 + 1 。

示例：

```
environment:
  UWSGI_WORKERS: 32
```

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:
  UWSGI_WORKERS: 4
  CELERY_MAIN_OPTIONS: --concurrency 2
  CELERY_NOTIFY_OPTIONS: --concurrency 1
  CELERY_TRANSLATE_OPTIONS: --concurrency 1
```

Docker 容器卷

Weblate 容器导出单个数据卷。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。在容器中，它挂载为 `/app/data`。

参见：

Docker 卷文档

进一步的配置定制

您可以在数据卷中进一步自定义 Weblate 安装，请参阅*Docker 容器卷*。

定制配置文件

您还可以在 `/app/data/settings-override.py` 中覆盖配置（请参阅*Docker 容器卷*）。加载所有环境设置后将执行此操作，因此完全新建它，并可用于自定义任何内容。

替换 logo 和其它静态文件

3.8-5 新版功能.

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 。

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
```

添加自己的 Python 模块

3.8-5 新版功能.

您可以将自己的 Python 模块放置在 `/app/data/python/` 中（请参阅*Docker 容器卷*），然后可以由 Weblate 加载它们，很可能是使用*定制配置文件*。

参见：

定制 Weblate

选择您的机器——本地或云提供商

使用 Docker Machine，您可以在本地计算机上或在任何数量的基于云的部署，例如 Amazon AWS，Greenhost 和许多其他提供商上创建 Weblate 部署。

在 Debian 和 Ubantu 上安装

硬件要求

Weblate 应该可以在所有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：

- 2 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。对于数百个翻译组件，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。

注解： 根据 Weblate 中管理的翻译大小，安装 Weblate 的实际要求差异很大。

安装

系统要求

安装所需的依赖包，来建立 Python 模块（参见软件要求）：

```
apt install \
   libxml2-dev libxslt-dev libfreetype6-dev libjpeg-dev libz-dev libyaml-dev \
   libcairo-dev gir1.2-pango-1.0 libgirepository1.0-dev libacl1-dev libssl-dev \
   build-essential python3-gdbm python3-dev python3-pip python3-virtualenv␣
↪virtualenv git
```

根据您想要使用的特性来安装想要的可选依赖包（参见可选依赖性 ）：

```
apt install tesseract-ocr libtesseract-dev libleptonica-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组件。

本地安装的使用说明：

```
# Web server option 1: NGINX and uWSGI
apt install nginx uwsgi uwsgi-plugin-python3

# Web server option 2: Apache with ``mod_wsgi``
apt install apache2 libapache2-mod-wsgi

# Caching backend: Redis
apt install redis-server

# Database server: PostgreSQL
apt install postgresql postgresql-contrib

# SMTP server
apt install exim4
```

Python 模块

提示: 我们使用 virtualenv 在与您的系统隔开的环境安装 Weblate。如果您不熟悉，查看 virtualenv User Guide 。

1. 为 Weblate 新建 virtualenv：

```
virtualenv --python=python3 ~/weblate-env
```

2. 为 Weblate 激活 virtualevn ：

```
. ~/weblate-env/bin/activate
```

3. 安装 Weblate ，包括所有依赖包：

```
pip install Weblate
```

4. 安装数据库驱动：

```
pip install psycopg2-binary
```

5. 根据您想要使用的特性，来装所所需要的可选依赖包（一些会需要另外的系统库，查看可选依赖性 ）：

```
pip install ruamel.yaml aeidon boto3 zeep chardet tesserocr
```

配置 Weblate

注解: 后面的步骤假定 Weblate 使用的 virtualevn 已经激活（可以通过 `. ~/weblate-env/bin/activate` 来实现）。如果不是这种情况，您必须指定到 **weblate** 命令的完全路径为 `~/weblate-env/bin/weblate` 。

1. 将文件 `~/weblate-env/lib/python3.7/site-packages/weblate/settings_example.py` 复制为 `~/weblate-env/lib/python3.7/site-packages/weblate/settings.py` 。
2. 将新的 `settings.py` 文件中的值调整为您所需要的。您可以跟随一起上市的例子，来用于测试，但您还会希望更改来用于生产设置，参见：调整配置 。
3. 为 Weblate 新建数据库及其结构（例子中的设置使用 PostgreSQL ，已经准备好的生产设置请查看*Weblate 的数据库设置* ）：

```
weblate migrate
```

4. 新建管理员用户账户，并将输出的密码复制到剪贴板，同时将它存储供以后使用：

```
weblate createadmin
```

5. 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，请参见压缩客户资产 ）：

```
weblate compress
```

7. 启动 Celery workers 。当用于开发目的是不需要这步，但仍然强烈推荐。更多信息请参见*使用 Celery 的后台任务*：

```
~/weblate-env/lib/python3.7/site-packages/weblate/examples/celery start
```

8. 启动开发服务器（生产设置请参见*运行服务器*）：

```
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 。
- 使用安装时得到的管理管理员证明来登录，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 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. 新建组件，它是翻译的真实对象——它执行版本控制系统（VCS ）仓库，并用于选择那个文件被翻译。更多细节请参见*组件配置* 。

   这里重要的字段是：组件名称、版本控制系统（VCS ）仓库地址和找到的翻译文件的掩码。Weblate 支持大范围的格式，包括 PO 文件、安卓资源字符串、IOS 字符串属性，Java 属性或 Qt Linguist 文件，更多细节请参看：*支持的文件格式* 。

3. 一旦完成上面的工作（根据您的版本控制系统 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 应该可以在所有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：

- 2 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。对于数百个翻译组件，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。

注解: 根据 Weblate 中管理的翻译大小，安装 Weblate 的实际要求差异很大。

安装

系统要求

安装所需的依赖包，来建立 Python 模块（参见软件要求）:

```
zypper install \
   libxslt-devel libxml2-devel freetype-devel libjpeg-devel zlib-devel libyaml-
↪devel \
   cairo-devel typelib-1_0-Pango-1_0 gobject-introspection-devel libacl-devel \
   python3-pip python3-virtualenv python3-devel git
```

根据您想要使用的特性来安装想要的可选依赖包（参见可选依赖性）:

```
zypper install tesseract-ocr tesseract-devel leptonica-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组件。

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 在与您的系统隔开的环境安装 Weblate。如果您不熟悉，查看 virtualenv User Guide 。

1. 为 Weblate 新建 virtualenv:

   ```
   virtualenv --python=python3 ~/weblate-env
   ```

2. 为 Weblate 激活 virtualevn :

   ```
   . ~/weblate-env/bin/activate
   ```

3. 安装 Weblate ，包括所有依赖包:

   ```
   pip install Weblate
   ```

4. 安装数据库驱动：

```
pip install psycopg2-binary
```

5. 根据您想要使用的特性，来装所所需要的可选依赖包（一些会需要另外的系统库，查看可选依赖性）：

```
pip install ruamel.yaml aeidon boto3 zeep chardet tesserocr
```

配置 Weblate

注解： 后面的步骤假定 Weblate 使用的 virtualevn 已经激活（可以通过 `. ~/weblate-env/bin/activate` 来实现）。如果不是这种情况，您必须指定到 **weblate** 命令的完全路径为 `~/weblate-env/bin/weblate` 。

1. 将文件 `~/weblate-env/lib/python3.7/site-packages/weblate/settings_example.py` 复制为 `~/weblate-env/lib/python3.7/site-packages/weblate/settings.py` 。
2. 将新的 `settings.py` 文件中的值调整为您所需要的。您可以跟随一起上市的例子，来用于测试，但您还会希望更改来用于生产设置，参见：调整配置 。
3. 为 Weblate 新建数据库及其结构（例子中的设置使用 PostgreSQL ，已经准备好的生产设置请查看*Weblate 的数据库设置* ）：

```
weblate migrate
```

4. 新建管理员用户账户，并将输出的密码复制到剪贴板，同时将它存储供以后使用：

```
weblate createadmin
```

5. 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，请参见压缩客户资产 ）：

```
weblate compress
```

7. 启动 Celery workers 。当用于开发目的是不需要这步，但仍然强烈推荐。更多信息请参见*使用 Celery 的后台任务* ：

```
~/weblate-env/lib/python3.7/site-packages/weblate/examples/celery start
```

8. 启动开发服务器（生产设置请参见运行服务器 ）：

```
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 。
- 使用安装时得到的管理管理员证明来登录，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 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. 新建组件，它是翻译的真实对象——它执行版本控制系统（VCS ）仓库，并用于选择那个文件被翻译。更多细节请参见组件配置 。

   这里重要的字段是：组件名称、版本控制系统（VCS ）仓库地址和找到的翻译文件的掩码。Weblate 支持大范围的格式，包括 PO 文件、安卓资源字符串、IOS 字符串属性，Java 属性或 Qt Linguist 文件，更多细节请参看：支持的文件格式 。

3. 一旦完成上面的工作（根据您的版本控制系统 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 应该可以在所有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：

- 2 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。对于数百个翻译组件，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。

注解： 根据 Weblate 中管理的翻译大小，安装 Weblate 的实际要求差异很大。

安装

系统要求

安装所需的依赖包，来建立 Python 模块（参见软件要求）：

```
dnf install \
   libxslt-devel libxml2-devel freetype-devel libjpeg-devel zlib-devel libyaml-
↪devel \
   cairo-devel pango-devel gobject-introspection-devel libacl-devel \
   python3-pip python3-virtualenv python3-devel git
```

根据您想要使用的特性来安装想要的可选依赖包（参见可选依赖性）：

```
dnf install tesseract-langpack-eng tesseract-devel leptonica-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组件。

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 在与您的系统隔开的环境安装 Weblate。如果您不熟悉，查看 virtualenv User Guide 。

1. 为 Weblate 新建 virtualenv：

   ```
   virtualenv --python=python3 ~/weblate-env
   ```

2. 为 Weblate 激活 virtualevn ：

   ```
   . ~/weblate-env/bin/activate
   ```

3. 安装 Weblate ，包括所有依赖包：

   ```
   pip install Weblate
   ```

4. 安装数据库驱动：

   ```
   pip install psycopg2-binary
   ```

5. 根据您想要使用的特性，来装所所需要的可选依赖包（一些会需要另外的系统库，查看可选依赖性）：

```
pip install ruamel.yaml aeidon boto3 zeep chardet tesserocr
```

配置 Weblate

注解: 后面的步骤假定 Weblate 使用的 virtualevn 已经激活（可以通过 `. ~/weblate-env/bin/activate` 来实现）。如果不是这种情况，您必须指定到 **weblate** 命令的完全路径为 `~/weblate-env/bin/weblate` 。

1. 将文件 `~/weblate-env/lib/python3.7/site-packages/weblate/settings_example.py` 复制为 `~/weblate-env/lib/python3.7/site-packages/weblate/settings.py` 。
2. 将新的 `settings.py` 文件中的值调整为您所需要的。您可以跟随一起上市的例子，来用于测试，但您还会希望更改来用于生产设置，参见：调整配置 。
3. 为 Weblate 新建数据库及其结构（例子中的设置使用 PostgreSQL ，已经准备好的生产设置请查看*Weblate 的数据库设置* ）：

   ```
   weblate migrate
   ```

4. 新建管理员用户账户，并将输出的密码复制到剪贴板，同时将它存储供以后使用：

   ```
   weblate createadmin
   ```

5. 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，请参见压缩客户资产 ）：

   ```
   weblate compress
   ```

7. 启动 Celery workers 。当用于开发目的是不需要这步，但仍然强烈推荐。更多信息请参见*使用 Celery 的后台任务* ：

   ```
   ~/weblate-env/lib/python3.7/site-packages/weblate/examples/celery start
   ```

8. 启动开发服务器（生产设置请参见运行服务器 ）：

   ```
   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 。
- 使用安装时得到的管理管理员证明来登录，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 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. 新建组件，它是翻译的真实对象——它执行版本控制系统（VCS ）仓库，并用于选择那个文件被翻译。更多细节请参见组件配置 。

   这里重要的字段是：组件名称、版本控制系统（VCS ）仓库地址和找到的翻译文件的掩码。Weblate 支持大范围的格式，包括 PO 文件、安卓资源字符串、IOS 字符串属性，Java 属性或 Qt Linguist 文件，更多细节请参看：支持的文件格式 。

3. 一旦完成上面的工作（根据您的版本控制系统 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 应该可以在所有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：

- 2 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。对于数百个翻译组件，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。

注解： 根据 Weblate 中管理的翻译大小，安装 Weblate 的实际要求差异很大。

安装

系统要求

安装所需的依赖包，来建立 Python 模块（参见软件要求）：

```
brew install pango libjpeg python git libyaml gobject-introspection
pip3 install virtualenv
```

确认 pip 能够找到 homebrew 提供的 `libffi` 版本——将在安装编译步骤中需要它。

```
export PKG_CONFIG_PATH="/usr/local/opt/libffi/lib/pkgconfig"
```

根据您想要使用的特性来安装想要的可选依赖包（参见可选依赖性 ）：

```
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组件。

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 在与您的系统隔开的环境安装 Weblate。如果您不熟悉，查看 virtualenv User Guide 。

1. 为 Weblate 新建 virtualenv：

   ```
   virtualenv --python=python3 ~/weblate-env
   ```

2. 为 Weblate 激活 virtualevn ：

   ```
   . ~/weblate-env/bin/activate
   ```

3. 安装 Weblate ，包括所有依赖包：

   ```
   pip install Weblate
   ```

4. 安装数据库驱动：

   ```
   pip install psycopg2-binary
   ```

5. 根据您想要使用的特性，来装所所需要的可选依赖包（一些会需要另外的系统库，查看可选依赖性 ）：

   ```
   pip install ruamel.yaml aeidon boto3 zeep chardet tesserocr
   ```

配置 Weblate

注解: 后面的步骤假定 Weblate 使用的 virtualevn 已经激活（可以通过 `. ~/weblate-env/bin/activate` 来实现）。如果不是这种情况，您必须指定到 **weblate** 命令的完全路径为 `~/weblate-env/bin/weblate` 。

1. 将文件 `~/weblate-env/lib/python3.7/site-packages/weblate/settings_example.py` 复制为 `~/weblate-env/lib/python3.7/site-packages/weblate/settings.py` 。
2. 将新的 `settings.py` 文件中的值调整为您所需要的。您可以跟随一起上市的例子，来用于测试，但您还会希望更改来用于生产设置，参见：调整配置 。
3. 为 Weblate 新建数据库及其结构（例子中的设置使用 PostgreSQL ，已经准备好的生产设置请查看 *Weblate 的数据库设置* ）：

```
weblate migrate
```

4. 新建管理员用户账户，并将输出的密码复制到剪贴板，同时将它存储供以后使用：

```
weblate createadmin
```

5. 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，请参见压缩客户资产 ）：

```
weblate compress
```

7. 启动 Celery workers 。当用于开发目的是不需要这步，但仍然强烈推荐。更多信息请参见使用 *Celery* 的后台任务 ：

```
~/weblate-env/lib/python3.7/site-packages/weblate/examples/celery start
```

8. 启动开发服务器（生产设置请参见运行服务器 ）：

```
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 。
- 使用安装时得到的管理管理员证明来登录，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 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. 新建组件，它是翻译的真实对象——它执行版本控制系统（VCS ）仓库，并用于选择那个文件被翻译。更多细节请参见组件配置 。

   这里重要的字段是：组件名称、版本控制系统（VCS ）仓库地址和找到的翻译文件的掩码。Weblate 支持大范围的格式，包括 PO 文件、安卓资源字符串、IOS 字符串属性，Java 属性或 Qt Linguist 文件，更多细节请参看：支持的文件格式 。

3. 一旦完成上面的工作（根据您的版本控制系统 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。

从源文件安装

1. 请首先按照用于您的系统的安装指示：
   - *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 来抓取最新的 Weblate 资源（或下载 tarball 包并将其解压）：

   ```
   git clone https://github.com/WeblateOrg/weblate.git weblate-src
   ```

   另外，您可以发布档案。从可以从我们的网站 <https://weblate.org/> 来下载。下载是加密签名的，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。您可以跟随一起上市的例子，来用于测试，但您还会希望更改来用于生产设置，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. 建立 Django 表、静态文件和初始数据（参见*填满数据库* 和*填满数据库* ）：

   ```
   weblate migrate
   weblate collectstatic
   weblate compress
   weblate compilemessages
   ```

   注解： 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。

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 事件就应该可用了。

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。

要重置 管理员密码，请在将 “WEBLATE_ADMIN_PASSWORD“设置为相应的 “Secret” 中的新密码的情况下，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 映像。`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，环境变量的完成列表请参见*Docker 环境变量* for full list of environment variables。

在 Kubernetes 上安装

注解： 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/> 找到图标，并且它可以在 <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
```

根据你的设置和经验，为你选择适当的安装方法：

- *使用 Docker 安装*, 推荐用于生产安装。
- Virtualenv 安装，推荐用于产品设置：
  - *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*
- *从源文件安装*, 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

Weblate 已知运行在 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 。但仍然可能工作，并愿意接受补丁。

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 依赖性

Weblate 用 Python 编写，并且支持 Python 3.6 或更新版本。可以使用 pip 或你的发布包来安装依赖性，完全列表可在 `requirements.txt` 中找到。

最重要的依赖性：

Django https://www.djangoproject.com/

Celery https://docs.celeryproject.org/

Translate Toolkit （翻译工具包） https://toolkit.translatehouse.org/

translation-finder https://github.com/WeblateOrg/translation-finder

Python Social Auth https://python-social-auth.readthedocs.io/

Django REST 框架 https://www.django-rest-framework.org/

可选依赖性

后面的模块对 Weblate 的一些特性是必须的。可以在 `requirements-optional.txt` 中找到。

`Mercurial` （对 Mercurial 仓库支持是可选的） https://www.mercurial-scm.org/

`phply` （对 PHP 支持是可选的） https://github.com/viraptor/phply

`tesserocr` （对截屏 OCR 是可选的） https://github.com/sirfz/tesserocr

`akismet` （对建议垃圾邮件保护是可选的） https://github.com/ubernostrum/akismet

`ruamel.yaml` （对*YAML files* 是可选的） https://pypi.org/project/ruamel.yaml/

`Zeep` （对微软术语服务 是可选的） https://docs.python-zeep.org/

`aeidon` （对*Subtitle files* 是可选的） https://pypi.org/project/aeidon/

数据库后端依赖性

Weblate 支持 PostgreSQL 、MySQL 和 MariaDB 数据库，更多细节请参见*Weblate 的数据库设置* 和后端文件。

其他系统要求

后面的依赖性必须安装在系统上：

`Git` https://git-scm.com/

Pango 、Cairo 和相关的头文件与 gir introspection 数据 https://cairographics.org/ , https://pango.gnome.org/ ，请参见*Pango 和 Cairo*

`git-review` （对 Gerrit 支持是可选的） https://pypi.org/project/git-review/

`git-svn` （对 Subversion 的支持是可选的） https://git-scm.com/docs/git-svn

`tesseract` 及其数据（对截屏 OCR 是可选的） https://github.com/tesseract-ocr/tesseract

`licensee` （当新建组件时可选用与删除许可） https://github.com/licensee/licensee

构建时的依赖

为了构建一些*Python 依赖性* ，你可能需要安装其依赖库。这取决于你如何安装它们，因此请参考单独包的文档。如果你使用预编译 `Wheels` 并使用 `pip` 安装或使用分发包时，你不会需要那些。

Pango 和 Cairo

在 3.7 版更改.

Weblate 使用 Pango 和 Cairo 来提供位图 widget（请参参见 promotion ）并提供检查（请参见*管理字型* ）。为了适当地安装 Python 绑定需要首先安装系统库的那些——Cairo 和 Pango 都是需要的，由此需要 Glib 。所有那些需要与开发文件和 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
```

并且可以从 <https://keybase.io/nijel> 得到更多识别信息。

应该验证签名与下载的压缩档案匹配。这种方式可以确保你使用发布的相同编码。还应该核实签名的日期，确定下载了最新的版本。

每个压缩档案伴随 `.asc` 文件在一起，它包含了需要使用的 PGP 签名。一旦它们处于相同的文件夹内，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’s server 下载钥匙链，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可以以这个名称发布密钥。需要确定密钥实际由提到的人所有。GNU 隐私手册在 Validating other keys on your public keyring （‘在你的公共钥匙链上验证其它密钥’）章节涵盖了这个题目。最可靠的方法是与开发者的真人真实交流，并交换密钥指纹，然后还可以依赖于可信任的 Web 。这种方式你可以信任通过其他签名的密钥，而其他人必须接触开发者真人。

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]
```

如果签名非法（压缩的档案被更改），那么会得到清晰的错误，而无论密钥是否可信：

```
$ 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 - setting:*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，见运行服务器 and 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）所有。

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, Databases, 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。

注解: 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 notes

建立 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 -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;
```

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",
        # 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 特性使用*PostgreSQL* 会执行得更好。这包括搜索与翻译记忆库，它们都使用了数据库中的全文本特性，而 PostgreSQL 的实施更胜一筹。

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 notes 和 MariaDB notes 。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` 。

下面是用于 8GB 内存服务器的 /:file:‘etc/my.cnf.d/server.cnf 的例子。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
```

（下页继续）

(续上页)

```
datadir=/var/lib/mysql

log-error=/var/log/mariadb/mariadb.log

innodb_large_prefix=1
innodb_file_format=Barracuda
innodb_file_per_table=1
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。

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。

邮件服务器安装使用这些设置进行配置: `EMAIL_HOST`, `EMAIL_HOST_PASSWORD`, `EMAIL_USE_TLS`, `EMAIL_USE_TLS`, `EMAIL_HOST_USER` and `EMAIL_PORT`. 它们的含义从名字上就看得出来，但是您可以在 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*

在反向代理后面运行

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* 。

参见:

针对垃圾邮件的保护, 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，并且那些从环境中接受代理配置。推荐的方法是在 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:

配置的例子

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

网站管理者的列表，当发生故障时它们接收通知，例如合并失败或 Django 错误的通知。

参见:

ADMINS

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:

Configuring the session engine, SESSION_ENGINE

DATABASES

到数据库服务器的连接性，细节请查看 Django 的文件。

参见:

Weblate 的数据库设置, DATABASES, Databases

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，错误将每封电子邮箱发送到 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

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。

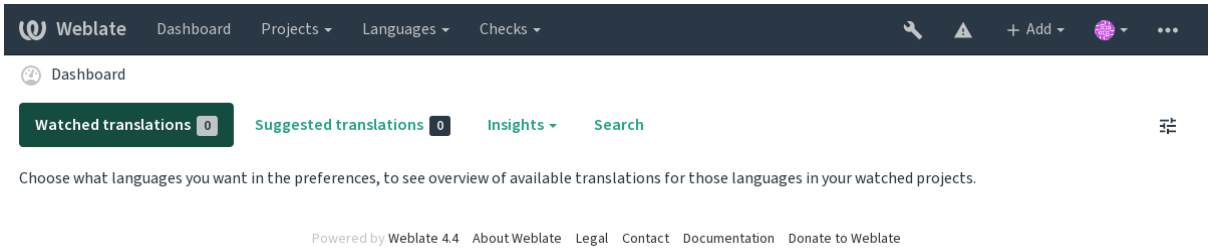
一旦完成，你将可以在管理界面检查 *Performance report* ，它会提示你网站上潜在的非最优的配置。

参见:

配置, 访问控制

2.1.9 生产设置

对于生产设置，可以进行后面的章节中描述的调整。最严格的设置将触发警告，如果超级用户登录的话，警告由顶部条的感叹标记来指示：



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
```

还可以复查来自管理界面 的每个检查表。

参见:

Deployment checklist

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件地址将不工作。这使用*SITE_DOMAIN* 来配置，它应该包含网站域名。

在 4.2 版更改: 在 4.2 版本之前，替代使用了 Django 网站框架，请参见 The "sites" framework 。

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 Strict Transport Security 标头。

警告: 不正确地设置这项会导致你的网站不可逆地（有时）崩溃。请首先阅读 HTTP Strict Transport Security 文件。

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* 。

参见:

Weblate 的数据库设置, *从其它数据库迁移到 PostgreSQL*, *调整配置*, Databases

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’s cache framework

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’s cache framework

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"
```

注解： 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/examples/generate-secret-key`，来产生新的密钥。

参见：

SECRET_KEY

Home 目录

在 2.1 版更改: 这不再需要了，Weblate 现在将所有数据存储在*DATA_DIR* 。

给用户运行 Weblate 的主目录应该存在，并且可以被这个用户写入。如果想要使用 SSH 来访问私有仓库，这是特别重要的，但 Git 也会需要访问这个目录（依赖于你使用的 Git 版本）。

可以在 `settings.py` 中更改 Weblate 使用的目录，例如，将其设置到 Weblate 树下面的 `configuration` 目录中：

```
os.environ["HOME"] = os.path.join(BASE_DIR, "configuration")
```

注解： 在 Linux 和其他类似 UNIX 系统上，到用户主目录的路径在 `/etc/passwd` 中确定。很多发布默认用户使用不可写入目录来提供 Web 内容服务（如 `apache` 、`www-data` 或 `wwwrun` ）。

参见：

Accessing repositories

模板加载

对于 Django 推荐使用缓存模板的加载程序。它将已分析的模板装入，并避免对每个单独的请求多进行分析。可以使用后面的模板来配置它（这里 `loaders` 设置很重要）：

```
TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [
            os.path.join(BASE_DIR, "templates"),
        ],
        "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",
            ],
            "loaders": [
                (
                    "django.template.loaders.cached.Loader",
                    [
                        "django.template.loaders.filesystem.Loader",
                        "django.template.loaders.app_directories.Loader",
                    ],
                ),
            ],
        },
    },
]
```

参见:

`django.template.loaders.cached.Loader`

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）。
- 提交待定的更改（每小时），请参见*惰性提交* 和*commit_pending* 。
- 更新组件警告（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 版更改: 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 发布中这是默认的设置。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"` 。

在一些情况下，各个服务对不同的地区具有不同的设置。例如当使用 Apache 时，你会想要在 `/etc/apache2/envvars` 中设置它：

```
export LANG='en_US.UTF-8'
export LC_ALL='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。

倾向使用的方法是在系统层次进行，更多细节请查看你的发布的文件（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"
```

压缩客户资产

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` （后者已经包括在例子的配置中）：

```
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

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 的配置例子*）
- 用于执行后台任务的 Celery （请参见*使用 Celery 的后台任务*）

注解： 这些服务之间由一些依赖性，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`。

注解： 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 服务器的例子）。

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： 在生产设置中不要使用这个服务器。它还没有通过安全审查或性能检测。还请参见 `runserver` 上的 Django 文件。

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 的配置例子*、*Apache 的配置例子 Apache 和 Gunicorn 的配置例子*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 版更改: 在 2.4 版本之前，Weblate 不能正常使用 Django 静态文件框架，并且设置更复杂。

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:

压缩客户资产, Deploying Django, Deploying static files

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 的配置例子

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/python3.7/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）:

```
# This example assumes Weblate is installed in virtualenv in /home/weblate/weblate-
↪env
# and DATA_DIR is set to /home/weblate/data, please adjust paths to match your␣
↪setup.
server {
    listen 80;
    server_name weblate;
    # Not used
    root /var/www/html;

    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）：

```
# This example assumes Weblate is installed in virtualenv in /home/weblate/weblate-
↪env
# and DATA_DIR is set to /home/weblate/data, please adjust paths to match your␣
↪setup.
[uwsgi]
plugins       = python3
master        = true
protocol      = uwsgi
socket        = 127.0.0.1:8080
wsgi-file     = /home/weblate/weblate-env/lib/python3.7/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

# 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：

How to use Django with uWSGI

Apache 的配置例子

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
#
# This example assumes Weblate is installed in virtualenv in /home/weblate/weblate-
↪env
# and DATA_DIR is set to /home/weblate/data, please adjust paths to match your␣
↪setup.
#
<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/
    <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
    WSGIProcessGroup weblate
    WSGIApplicationGroup %{GLOBAL}

    WSGIScriptAlias / /home/weblate/weblate-env/lib/python3.7/site-packages/
↪weblate/wsgi.py process-group=weblate request-timeout=600
    WSGIPassAuthorization On

    <Directory /home/weblate/weblate-env/lib/python3.7/site-packages/weblate/>
        <Files wsgi.py>
        Require all granted
        </Files>
    </Directory>

</VirtualHost>
```

注解： 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, How to use Django with Apache and mod_wsgi

Apache 和 Gunicorn 的配置例子

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
#
# This example assumes Weblate is installed in virtualenv in /home/weblate/weblate-
↪env
# and DATA_DIR is set to /home/weblate/data, please adjust paths to match your␣
↪setup.
#
<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

    # 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:

How to use Django with Gunicorn

在路径下运行 Weblate

在 1.3 版更改: 从 Weblate 1.3 开始支持。

推荐当 Weblate 使用 WSGI 时使用 prefork MPM。

为 "/weblate" 下的 Weblate 提供服务的 Apache 配置的例子。再次使用 mod_wsgi （还作为 `weblate/examples/apache-path.conf` 获得）:

```
#
# VirtualHost for Weblate, running under /weblate path
#
```

（下页继续）

(续上页)

```
# This example assumes Weblate is installed in virtualenv in /home/weblate/weblate-
↪env
# and DATA_DIR is set to /home/weblate/data, please adjust paths to match your␣
↪setup.
#
<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
    </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
    WSGIProcessGroup weblate
    WSGIApplicationGroup %{GLOBAL}

    WSGIScriptAlias /weblate /home/weblate/weblate-env/lib/python3.7/site-packages/
↪weblate/wsgi.py process-group=weblate request-timeout=600
    WSGIPassAuthorization On

    <Directory /home/weblate/weblate-env/lib/python3.7/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

3.2 新版功能.

Webate 使用 Celery 来处理后台任务。使用 Redis 作为后端的典型安装看起来像这样：

```
CELERY_TASK_ALWAYS_EAGER = False
CELERY_BROKER_URL = "redis://localhost:6379"
CELERY_RESULT_BACKEND = CELERY_BROKER_URL
```

参见:

Redis broker configuration in Celery

为了开发，您会想要使用急切配置，它在运行中处理所有任务，但会冲击 Weblate 的性能：

```
CELERY_TASK_ALWAYS_EAGER = True
CELERY_BROKER_URL = "memory://"
CELERY_TASK_EAGER_PROPAGATES = True
```

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
```

注解： 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。

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` 文件夹一起上市的例子文件。

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"

# 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,
```

（下页继续）

(续上页)

```
# increase concurency 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"

# Internal Weblate variable to indicate we're running inside Celery
CELERY_WORKER_RUNNING="1"
```

Logrotate 配置作为 `/etc/logrotate.d/celery` 放置：

```
/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

可以使用 *celery_queues* 来查看当前 Celery 任务队列的长度。在队列太长的情况下，会在管理界面得到配置错误。

> **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：

Configuration and defaults, Workers Guide, Daemonization, Monitoring and Management Guide, *celery_queues*

2.1.12 监测 Weblate

Weblate 提供 `/healthz/` URL 作为简单的健康检查来使用，例如使用 Kubernetes 。

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": "master",
    "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

依赖于您的数据库后端，会有几个选项来迁移数据库。最直接的是将数据库转储到一个服务器上，并将它导入新的服务器中。另外，在数据库支持的情况下可以使用数据库复制。

最好的方式是使用数据库自带工具，因为它们通常最有效（例如 **mysqldump** 或 **pg_dump** ）。如果您想要在不同的数据库之间迁移，唯一的选项恐怕是使用 Django 管理来转储并导入数据库：

```
# Export current data
weblate dumpdata > /tmp/weblate.dump
# Import dump
weblate loaddata /tmp/weblate.dump
```

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。

其他注释

不要忘记移动 Weblate 会使用的其他服务，如 Redis 、Cron 任务或定制的身份验证后端。

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

注解： 后面的部署不是由 Weblate 团队开发或支持的。部分设置会与本文件中描述的有偏差。

Bitnami Weblate 栈

Bitnami 为很多平台提供 Weblate 栈 <https://bitnami.com/stack/weblate> 。设置在安装过程中调整，更多文件请参见 <https://bitnami.com/stack/weblate/README.txt> 。

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。

Cloudron install

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）：

Install with YUNOHOST

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 映像升级

官方 Docker 映像（请参见*使用 Docker 安装*）已经将所有升级步骤集成了。除了拉取最新的版本外没有手动步骤。

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`）。

在升级前总是查看*与特定版本相关的指示*。在你跳过一些版本的情况下，请遵从在升级中您跳过的所有版本的指示。有时最好升级到一些中间版本，来确保平滑迁移。跨多发行版本的升级应该可以工作，但还没有像单一版本升级一样测试过。

注解： 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，可以通过后面的来实现：

   ```
   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
   ```

3. 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*。
4. 升级数据库架构：

   ```
   weblate migrate --noinput
   ```

5. 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
   ```

6. 压缩 JavaScript 和 CSS 文件（可选步骤，请参见*压缩客户资产* ）：

   ```
   weblate compress
   ```

7. 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
   ```

8. 验证您的设置合理（还请参见*生产设置*）：

   ```
   weblate check --deploy
   ```

9. 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。

显著的配置与依赖性更改:

- 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。
- *DeepL* 机器翻译现在默认为 v2 API，在你当前的 DeepL 订购不支持的情况下，会需要调整*MT_DEEPL_API_VERSION*。

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。

显著的配置与依赖性更改:

- 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，外部组件会需要调整。
- 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。

显著的配置与依赖性更改:

- 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 中暴露的组件。在使用时你会需要更新*Weblate 客户端*。
- 根据翻译的字符串数量，数据库迁移到 4.3 会花费很长时间（期望每 10 万个字符串的迁移时间大约为 1 小时）。
- 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。
- 在使用 **hub`**或**:command:`lab`**与 **GitHub** 或 **GitLab** 集成的情况下，需要重新配置它，请参见**:setting:`GITHUB_CREDENTIALS** 和*GITLAB_CREDENTIALS*。
- **4.3.1 版本变更:** Celery 配置更改为添加 “memory“ 队列。请调整你的启动脚本和 CELERY_TASK_ROUTES 设置。
- **在 4.3.2 版本中更改:** 插件的 “post_update“方法现在采用另外的 “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。

显著的配置与依赖性更改:

- 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*。
- ** 4.4.1 版的修改: ** *单语 gettext* 现在当出现时使用 msgid 和 msgctxt。这将更改这样的文件中翻译字符串的 IDs。请确认在升级前提交将这样文件的更改挂起。
- **Changed in 4.4.1:** Increased minimal required version of translate-toolkit to address several file format issues.

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 3.5 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 Upgrading from Python 2 to Python 3 in the Weblate 3.11.1 documentation。

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，你应该迁移到 PostgreSQL，因为它是 4.0 发布版本唯一支持的数据库后端。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。

建立 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 -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. 添加 PostgreSQL 作为到:file:‘settings.py‘的另外的数据库连接：

```
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
```

(下页继续)

(续上页)

```
        "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

因为 Weblate 开始编写出来替换 Pootle，所以支持从 Pootle 迁移用户账户。你可以将 Pootle 的用户转储，并使用 `importusers` 将他们导入。

2.4 备份和移动 Weblate

2.4.1 使用 BorgBackup 进行的自动化备份

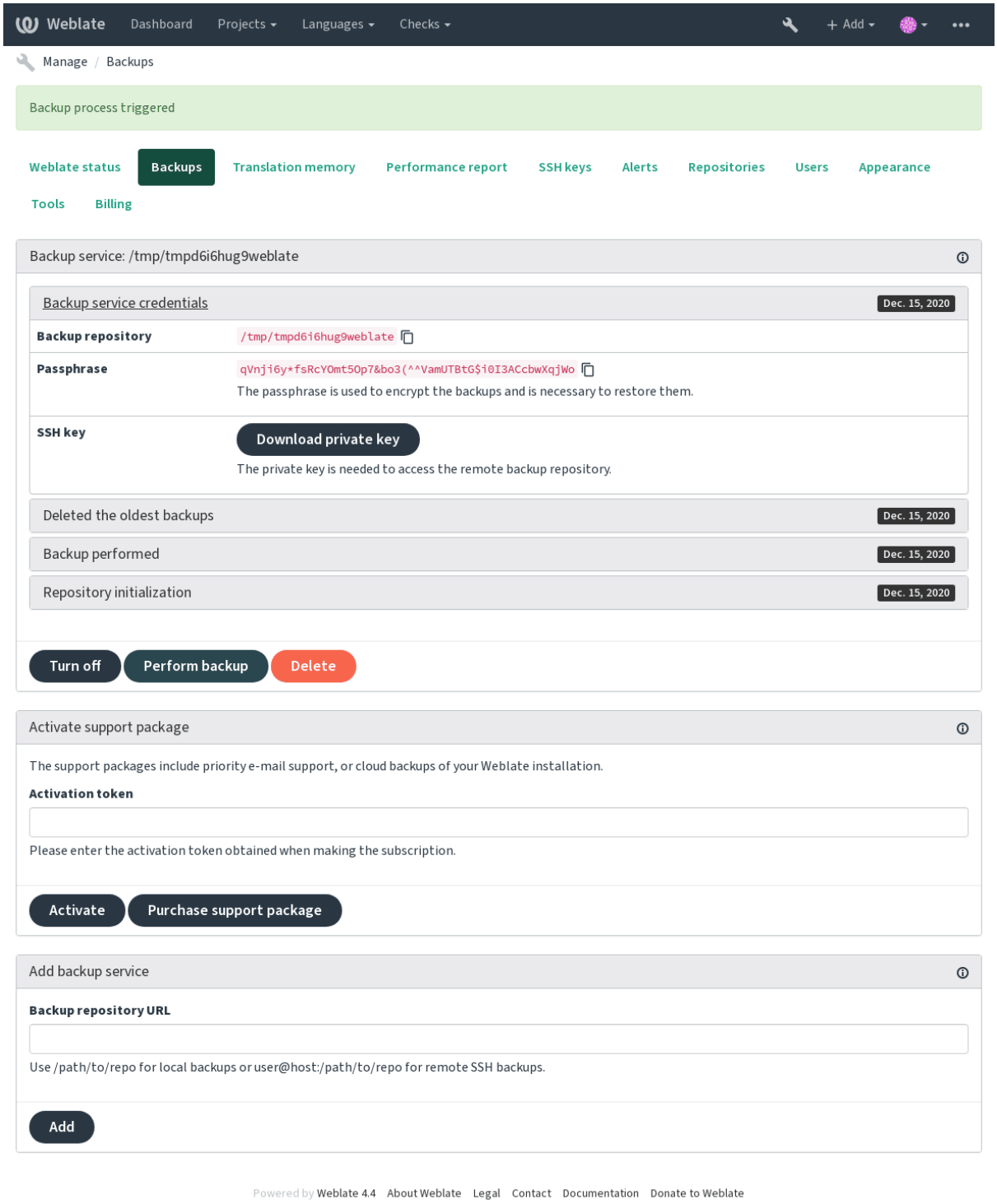
3.9 新版功能.

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。可以在管理界面中的 *Backups* 选项卡上控制备份。

在 4.4.1 版更改: PostgreSQL 和 MySQL/MariaDB 数据库都包括在自动备份中。

使用 Borg 的备份是递增的，Weblate 配置为保留后面的备份：

- 14 个每天备份
- 8 个每周备份
- 6 个每月备份


Weblate
Dashboard
Projects
Languages
Checks
+ Add
Manage / Backups
Backup process triggered
Weblate status
Backups
Translation memory
Performance report
SSH keys
Alerts
Repositories
Users
Appearance
Tools
Billing
Backup service: /tmp/tmpd6i6hug9weblate
Backup service credentials
Dec. 15, 2020
Backup repository
/tmp/tmpd6i6hug9weblate
Passphrase
qVnji6y*fsRcYOmt5Op7&bo3(^^VamUTBtG$i0I3ACcbwXqjWo
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
Dec. 15, 2020
Backup performed
Dec. 15, 2020
Repository initialization
Dec. 15, 2020
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 for remote SSH backups.
Add
Powered by Weblate 4.4
About Weblate
Legal
Contact
Documentation
Donate to Weblate

Borg 加密密钥

BorgBackup 生成加密的备份，没有密码的话就不能恢复备份。当添加备份服务时产生密码，并且应该将其复制并保存在安全的地方。

在使用 *Weblate 提供的备份存储* 的情况下，请同样备份私有 SSH 密钥——它用于访问你的备份。

参见:

borg init

2.4.2 Weblate 提供的备份存储

备份 Weblate 事例最简单的方法是购买 backup service at weblate.org 。激活过程可以分几步来执行:

1. 在 https://weblate.org/support/#backup 上购买备份服务。
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，并得到访问信息来备份。
4. 在 *Backups* 标签打开新的备份配置。
5. 备份 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。

2.4.3 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:

General 在 Borg 文件中

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*。运行 Weblate 的用户必须可写入备份目录（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，但需要权限来执行。

提示: 当在 Docker 中运行 Weblate 时，请确认备份位置显露为 Weblate 容器的卷。

一个选项是将备份防止在现有的卷中。例如，选择 `/app/data/borgbackup`。这是容器中现有的卷。

也可以在 Docker 的编写文件中为备份添加新的容器并使用例如 `/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 会不能将备份写入那里。

远程备份

支持使用 SSH 的远程备份。SSH 服务器需要安装 BorgBackup。Weblate 使用 SSH 密钥连接服务器，请确保 Weblate SSH 密钥被服务器接受（请参见*Weblate SSH 密钥*）。

提示: *Weblate 提供的备份存储* 为你提供自动远程备份。

2.4.4 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. 使用 `borg list REPOSITORY` 列出服务器上存在的备份。
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*。
6. 将整个存储的数据目录复制到在*DATA_DIR* 中配置的位置。

Borg 会话会是这个样子的：

```
$ borg list /tmp/xxx
Enter passphrase for key /tmp/xxx:
2019-09-26T14:56:08                  Thu, 2019-09-26 14:56:08␣
↪[de0e0f13643635d5090e9896bdaceb92a023050749ad3f3350e788f1a65576a5]
$ borg extract /tmp/xxx::2019-09-26T14:56:08
Enter passphrase for key /tmp/xxx:
```

参见:

borg list, borg extract

2.4.5 手动备份

依赖于你想存储什么，Weblate 存储的类型数据备份在各自的位置。

提示: 在进行手动备份时，会想要通过将 `weblate.I028` 添加到 `settings.py` 的 `SILENCED_SYSTEM_CHECKS` 中，或 Docker 的 *WEBLATE_SILENCED_SYSTEM_CHECKS* 中，来关闭 Weblate 的缺少备份警告。

```
SILENCED_SYSTEM_CHECKS.append("weblate.I028")
```

数据库

实际存储位置依赖于数据库的设置。

数据库是最重要的存储。要新建数据库的常规备份，没有的话翻译设置就丢失了。

本地数据库备份

推荐的方式是使用数据库的本地工具如 **pg_dump** 或 **mysqldump** 来备份数据库。这通常比 Django 备份执行得好，并且恢复所有数据的完整表格。

可以在更新版的 Weblate 中恢复备份，当运行 `migrate` 时执行任何必要的迁移。如何在两个版本之间执行升级的更多细节信息请咨询*升级 Weblate* 。

Django 数据库备份

另外，可以使用 Django 的 `dumpdata` 命令备份数据库。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。

在恢复前，需要运行与进行备份使用的 Weblate 完全一致的版本。这是必须的，因为数据库结构在发布版本之间有变化，并且会导致某种方式的数据损坏。在安装相同版本后，使用 `migrate` 来运行所有的数据库迁移。

一旦完成，数据库中就已经建立了一些入口，并且同样建立在数据库备份中。推荐的方法是使用管理 shell （请参见*调用管理命令* ）手动删除这样的入口：

```
weblate shell
>>> from weblate.auth.models import User
>>> User.objects.get(username='anonymous').delete()
```

文件

如果有充足的备份空间，则简单地备份整个*DATA_DIR* 。即使包括一些不想要的文件，这样做也是安全的。后面的部分具体描述了什么应该备份，什么应该跳过。

下载的数据用于备份

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` 。

数据库备份默认存储为纯文本，但也可以通过使用:setting:‘DATABASE_BACKUP‘来压缩的或整个跳过。

版本控制仓库

存储在*DATA_DIR* "/vcs"中。

版本控制仓库包含了带有 Weblate 更改的上游仓库的复制备份。如果对所有翻译组件推进了同意允许，那么所有 Weblate 更改都包括在上游中，并且不必备份 Weblate 侧的仓库。它们可以从上游位置再次克隆，而不会丢失数据。

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。

可以备份用户上传的文件（例如*字符串的可见语境*）。

Celery 任务

Celery 任务队列会包含一些信息，但备份通常不需要它。最多就是丢失了翻译内存还没有处理的更新。推荐在恢复时随便执行全文或仓库更新，这样不会有丢失的问题。

参见:

使用 Celery 的后台任务

手动备份的命令行

使用 crom 任务，可以新建批处理命令，以每天为单位来执行，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选择自己的 xz 选项，例如用于压缩的内存量；请参见 https://linux.die.net/man/1/xz

可以根据需要调整文件夹和文件的列表。例如，为了节省翻译内存（在备份文件夹中），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
```

2.4.6 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
   ```

2.4.7 移动 Weblate 安装

按照上面备份与恢复的说明，将安装重定位到不同系统。

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 身份验证后台

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基于 Django 其他项目的用户数据库（请参见*从 Pootle 迁移* ）。

也可以另外新建 Django，相对于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 Welcome to Python Social Auth’s documentation! ，Weblate 支持很多使用第三方服务的身份验证，如 GitLab 、Ubuntu 、Fedora 等。

请检查 Django Framework 中的通用配置指示的文件。

注解: 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，只需添加一个条目至设置:setting:‘django:AUTHENTICATION_BACKENDS‘即可 (可能还需为一个给定的验证方式添加密钥) 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 无法将功劳归于作出贡献的用户。

参见:

Python Social Auth backend

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"
SOCIAL_AUTH_GITHUB_SECRET = "GitHub Client Secret"
SOCIAL_AUTH_GITHUB_SCOPE = ["user:email"]
```

应该配置 GitHub 具有回调 URL 作为 `https://example.com/accounts/complete/github/` 。

注解： 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.BitbucketOAuth",
    "social_core.backends.email.EmailAuth",
    "weblate.accounts.auth.WeblateUserBackend",
)

# Social auth backends setup
SOCIAL_AUTH_BITBUCKET_KEY = "Bitbucket Client ID"
SOCIAL_AUTH_BITBUCKET_SECRET = "Bitbucket Client Secret"
SOCIAL_AUTH_BITBUCKET_VERIFIED_EMAILS_ONLY = True
```

注解： 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
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = "Client ID"
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = "Client secret"
```

注解： 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"]
```

注解： 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/'
```

注解: 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

微软 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",
)
```

(下页继续)

(续上页)

```
# OAuth2 keys
SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_KEY = ""
SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_SECRET = ""
# Tenant ID
SOCIAL_AUTH_AZUREAD_TENANT_OAUTH2_TENANT_ID = ""
```

注解: 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 = ""
```

注解: 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

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。

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/` 。

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。
- 密码不能包括单个字符或只有空白字符。
- 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

4.1.1 新版功能.

请遵守 Python Social Auth 的指示来配置。显著的差异有：

- Weblate 支持单一 IDP，在 SOCIAL_AUTH_SAML_ENABLED_IDPS 中被称为 weblate 。
- SAML XML 元数据 URL 为 /accounts/metadata/saml/ 。
- 后面的设置自动填入：SOCIAL_AUTH_SAML_SP_ENTITY_ID 、SOCIAL_AUTH_SAML_TECHNICAL_CONTACT 、SOCIAL_AUTH_SAML_SUPPORT_CONTACT

配置的例子：

```
# 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_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",
    }
}
```

参见：

Configuring SAML in Docker, 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
```

警告： 早于 1.3.0 版的 django-auth-ldap ，自动分配组 对新建立的用户无法正常工作。

注解： 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"

# 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
```

注解： 你应当从设置的:setting:*django: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帐户登录，如果你已经创建了它（如，通过使用:djadmin:*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)"
)
```

活动目录集成

```
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 = {
    # 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.
    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

在 3.0 版更改: 在 Weblate 3.0 之前，特权系统基于 Django，但现在是专门为 Weblate 构建的。如果使用的是旧版本，请查阅该版本的文档，此处的信息将不适用。

Weblate 带有细粒度的特权系统，可以为整个实例或在有限范围内分配用户权限。

基于组和角色的权限系统，其中角色定义了一组权限，组将它们分配给用户和翻译，请参阅用户，角色，用户组和权限 以获取更多详细信息。

安装后，将创建一组默认的组，可以使用这些组为整个实例分配用户角色（请参阅默认群组和角色 ）。此外，启用项目访问控制 后，可以将用户分配给特定的翻译项目。使用客户访问控制 可以实现更细粒度的配置。

2.6.1 通用设置

锁定 Weblate

为了完全锁定 Weblate，你可以使用`REQUIRE_LOGIN` 强制用户登录并用`REGISTRATION_OPEN` 来防止新注册。

全网站范围的权限

为了管理整个事例的权限，只通过将用户添加到 *Users* （这通常使用自动分配组 默认实现）、*Reviewers* 和 *Managers* 群组中即可。将所有的项目保持为 *Public* （请参考项目访问控制 ）。

每个每个项目的权限

注解: 此功能对于运行托管自由软件计划的项目不可用。

将项目设置为 *Protected* 或 *Private* ，并在 Weblate 的界面上管理每个项目的用户。

为语言、组件或项目添加语言

注解: 此功能对于运行托管自由软件计划的项目不可用。

可以根据项目、组件或语言向任何用户另外授予权限。为了实现这一目的，对给定的资源建立新的群组（例如 “捷克语译者”）并配置。对于所选的资源，任何指定的权限可以授予群组内的成员。

如果使用每个项目的权限的话，那么无需另外设置即可正常工作。对于整个事例的权限，你可能还想从 *Users* 群组中去掉这些权限，或者更改将所有用户自动指定给哪个群组（请参考自动分配组 ）。

参见:

权限检查

2.6.2 项目访问控制

注解： 通过允许 ACL ，所有用户禁止访问给定项目的任何内容，除非可以将权限授予他们去做。

注解： 此功能对于运行托管自由软件计划的项目不可用。

可以限制用户访问独立的项目。这个特性通过每个单独项目的配置中 *Access control* 来打开。这会为这个项目自动建立几个群组，请参考预定义的群组。

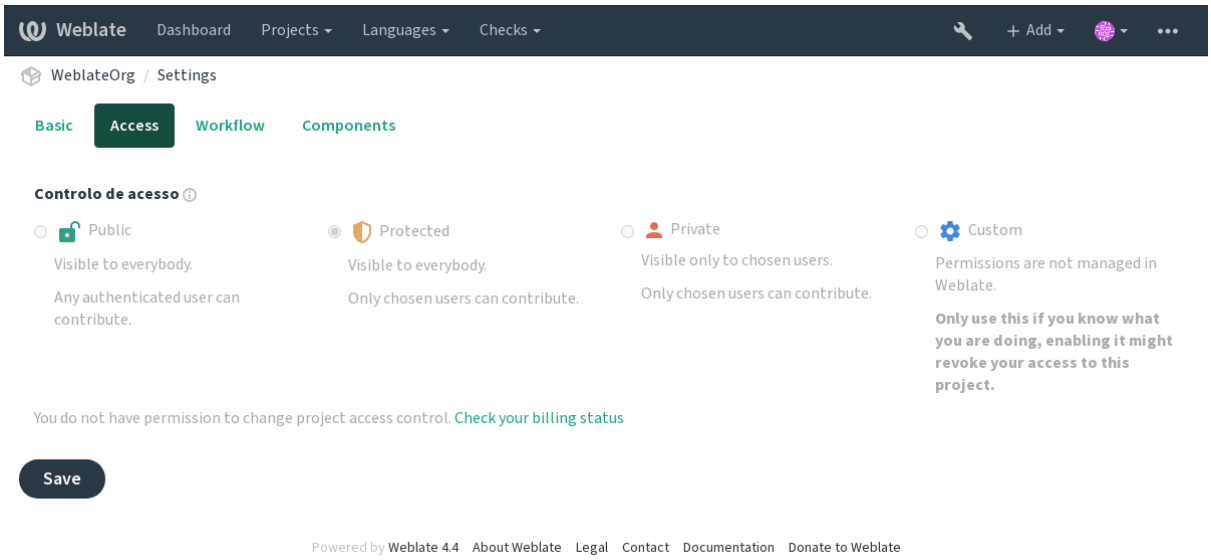
后面的选择是为了 *Access control* ：

公开的 公开可见、可翻译

受保护的 公开可见，但只允许被选择的用户翻译

私有的 公开可见，但只允许被选择的用户翻译

自定义 Weblate 不允许管理用户，请参考客户访问控制 。



为了允许访问这个项目，必须将权限直接添加给特定用户，或在 Django 管理界面上添加给用户所在的群组，或者使用项目页面上的用户管理，如下面的描述管理每个项目的访问控制 。

注解： 即使打开 ACL ，也可以看到项目的一些概要信息：

- 整个事例的统计数据，包括所有项目计数。
- 整个事例的语言概要，包括所有项目的计数。

2.6.3 自动分配组

可以新建 Weblate 根据用户的邮件地址自动将用户添加到用户组中。只有在建立账户时这一自动指定才进行。

可以对每个用户组在 Django 管理界面中新建（在 *Authentication* 部分）。

注解： 对于 *Users* 和 *Viewers* 用户组的自动用户组指定，总是通过 Weblate 在合并时建立，在想将其关闭的时候，简单地设置正则表达式为 `^$` 即可，因为永远不匹配。

2.6.4 用户，角色，用户组和权限

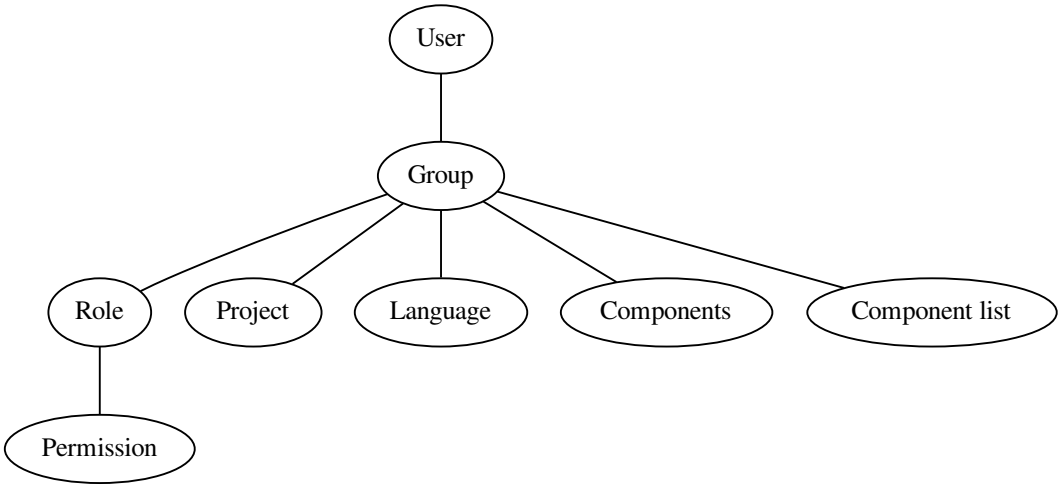
身份验证模型包括几个对象：

权限 Weblate 定义的各自权限。可以不指定各自权限，这可以只通过角色指定实现。

Role 角色定义为一组权限。这能够在几个地方重复使用这些组，并使管理更容易。

User 用户可以使几个用户组的成员。

群组 用户组与角色、用户和身份验证对象（项目、语言和组件列表）联系。



权限检查

任何时候检查权限来决定是否执行给定的动作时，根据范围来执行检查，并且后面的检查依次执行：

1. 群组 *Component list* 与访问的组件或项目匹配（对于项目级的访问）。
2. 群组 *Component* 与访问的组件或项目匹配（对于项目级的访问）。
3. 群组 *Projects* 与访问的项目匹配。

可以看到，为组件授予访问权限同样也会自动地授予用户所包括项目的访问权限。

注解： 只使用第一条规则。所以如果设置所有的 *Component list* 、*Components* 和 *Project* ，只会应用 *Component list* 。

如果检查翻译许可，则会执行另外的步骤：

4. 群组 *Languages* 与访问过的翻译进行匹配，它会被组件或项目级别的访问忽略。

提示： 可以使用 *Language selection* 或 *Project selection* 来自动包括所有语言或项目。

检查项目的访问权限

用于可以成为与项目或其中任何组件相连接的用户组的成员。只有成员即可，不需要特别许可来访问项目（这在默认的 *Viewers* 用户组中使用，请参见默认群组和角色 ）。

检查对组件的访问

一旦用户可以访问包含的项目，就能访问不受限制的组件。通过允许受限制的访问 ，访问组件需要组件（或包含组件列表）的明确授权。

2.6.5 管理用户和群组

可以使用 `/admin/` URL 下面的 Django 管理界面，来管理所有用户和用户组。

管理每个项目的访问控制

注解： 这个特性只对 ACL 控制的项目起作用，请参见项目访问控制 。

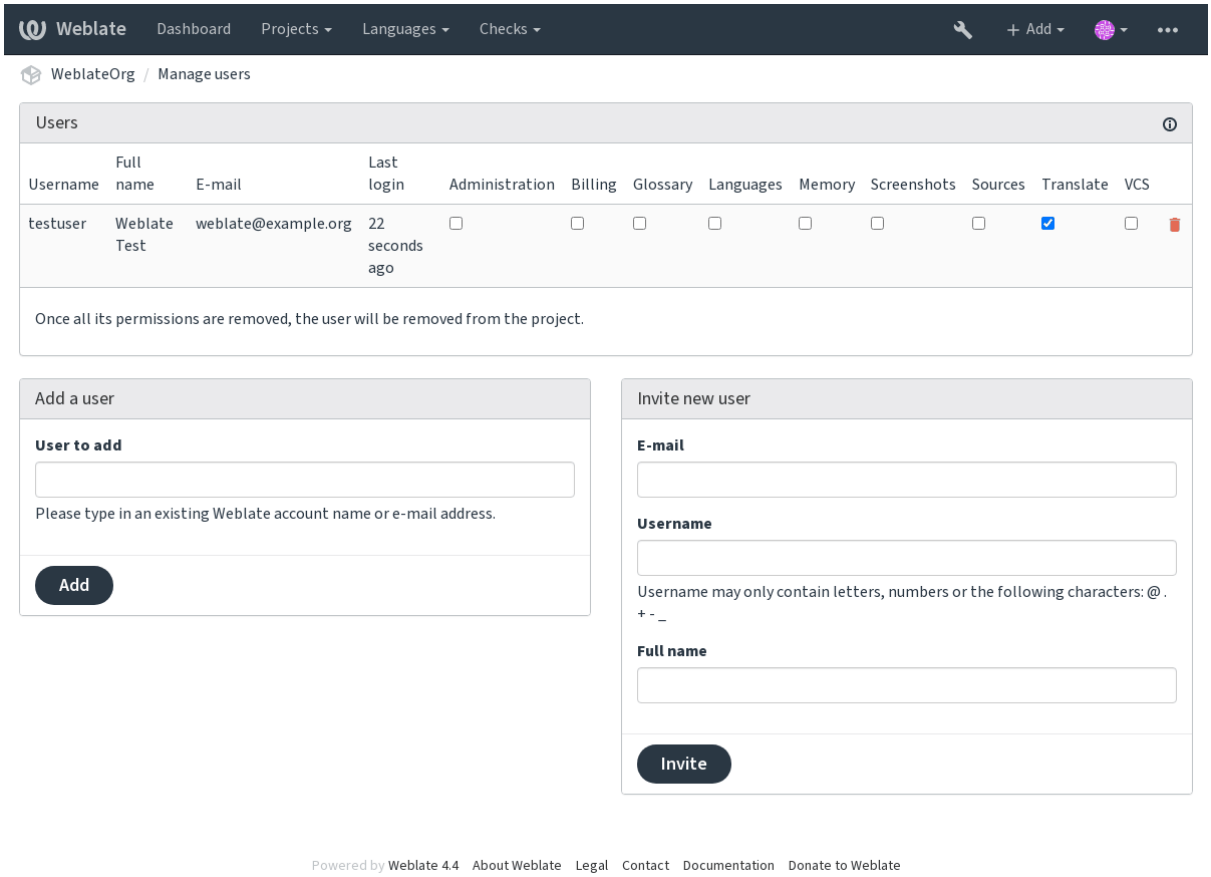
具有 *Manage project access* 特权的用户（请参见访问控制 ）还可以通过项目页面上的打开访问控制来管理用户。这个界面允许你：

- 将现有用户添加到项目中
- 邀请新用户参加到项目中
- 更改用户的权限
- 取消用户的权限

3.11 新版功能.

- 重新发送用户电子邮件邀请，使任何之前发送的要求无效

可以在项目菜单的 *Manage* 进行用户管理：



参见:

项目访问控制

预定义的群组

Weblate 的项目带有预定义的群组，可以为之指定用户。

Administration
在项目中可以有所有的权限。

Glossary
可以管理词汇表（添加或删除权限，或上传）。

Languages
可以管理翻译语言（添加或删除翻译）。

Screenshots
可以管理截屏（添加或删除截屏，并将其与源字符串关联）。

Sources
可以在:ref: ‘monollingual ‘和源字符串信息中编辑源字符串。

Translate
可以翻译项目，并将离线的翻译上传。

VCS
可以管理版本控制系统（VCS）并访问导出的仓库。

Review
可以在复查时批准翻译。

Billing
可以访问账单信息（请参见账单 ）。

2.6.6 客户访问控制

通过选择 *Custom* 作为 *Access control* ，Weblate 会对停止管理给定项目的访问权限，使用 Django 管理界面可以管理所有用户和群组。这可以用于确定更富在的访问控制，或在单一的 Weblate 界面中对所有项目新建可分享的访问策略。如果想要对所有项目默认打开这个功能，请配置`DEFAULT_ACCESS_CONTROL`。

> **警告：** 通过将其打开，Weblate 会删除所有为这个项目建立的项目访问控制 。如果没有事件的管理权限却去做的话，会立即丢失管理项目的访问权限。

2.6.7 默认群组和角色

这些角色和群组在安装时创建。内建的角色总是在升级时由数据库合并来更新，并且任何客户更改都会丢失。在想要定义自己的一组许可的情况下，请为此定义一个新的角色。

特权列表

账单（请参见账单 ） 查看账单信息 [*Administration*, *Billing*]

修改 下载更改 [*Administration*]

注释 发表注释 [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*]

删除注释 [*Administration*]

组件 编辑组件设置 [*Administration*]

锁定组件，防止被翻译 [*Administration*]

词汇表 添加词汇表入口 [*Administration*, *Manage glossary*, *Power user*]

编辑词汇表入口 [*Administration*, *Manage glossary*, *Power user*]

删除词汇表入口 [*Administration*, *Manage glossary*, *Power user*]

上传词汇表入口 [*Administration*, *Manage glossary*, *Power user*]

自动建议 使用自动建议 [*Administration*, *Power user*]

项目 编辑项目设置 [*Administration*]

更改项目访问权限 [*Administration*]

报告 下载报告 [*Administration*]

截图 添加截屏 [*Administration*, *Manage screenshots*]

编辑截屏 [*Administration*, *Manage screenshots*]

删除截屏 [*Administration*, *Manage screenshots*]

源字符串 编辑源字符串信息 [*Administration*, *Edit source*]

字符串 添加新字符串 [管理组]

忽略失败的复查 [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*]

编辑字符串 [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*]

复查字符串 [*Administration*, *Review strings*]

当建议被强制执行时需要编辑字符串 [*Administration*, *Review strings*]

编辑源字符串 [*Administration*, *Edit source*, *Power user*]

建议 接受建议 [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*]

添加建议 [*Add suggestion*, *Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*]

删除建议 [*Administration*]

为建议投票 [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*]

翻译 新建翻译 [*Administration*, *Manage languages*, *Power user*]

执行自动翻译 [*Administration*, *Manage languages*]

删除现有的翻译 [*Administration*, *Manage languages*]

开始另一门语言的翻译 [*Administration*, *Manage languages*]

上传 定义翻译上传的作者 [*Administration*]

使用上传的内容覆盖现有的字符串 [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*]

上传翻译的字符串 [*Administration*, *Edit source*, *Power user*, *Review strings*, *Translate*]

版本控制系统（VCS） 访问内部仓库 [*Access repository*, *Administration*, *Manage repository*, *Power user*]

将更改提交到内部代码库 [*Administration*, *Manage repository*]

从内部代码库推送更改 [*Administration*, *Manage repository*]

重置内部代码库的更改 [*Administration*, *Manage repository*]

查看上游仓库位置 [*Access repository*, *Administration*, *Manage repository*, *Power user*]

更新内部代码库 [*Administration*, *Manage repository*]

全网站范围的特权 使用管理界面

添加新项目

添加语言定义

管理语言定义

管理群组

管理用户

管理角色

管理公告

管理翻译记忆库

管理组件列表

注解: 全网站特权不会授予任何默认角色。这些特权特别强大，非常接近超级用户状态——多数特权会影响 Weblate 安装的所有项目。

群组列表

下面的群组在安装时建立（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 对非授权用户确定权限。

这个群组只包括匿名用户（请参参见`ANONYMOUS_USER_NAME`）。

可以从群组中去掉角色，来限制非授权用户的权限。

默认角色: *Add suggestion*, *Access repository*

Viewers 这一角色确保公开项目对所有用户可见。所有用户默认是这个群组的成员。

所有用户默认为这个群组的成员，使用自动分配组 。

默认角色：无

用户 所有用户的默认群组。

所有用户默认为这个群组的成员，使用自动分配组 。

默认角色: *Power user*

校对 复核员的群组（参见翻译工作流 ）。

默认角色: *Review strings*

管理人员 管理员的群组。

默认角色: *Administration*

> **警告:** 永远不要删除预先定义的 Weblate 群组和用户，因为这会导致意外的错误。如果不想要使用这些特性，只去掉他们的特权即可。

2.7 翻译项目

2.7.1 翻译组织

项目/组件的可翻译的版本控制系统（VCS ）内容，由 Weblate 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，:ref:‘component‘（实际上是要翻译的组件），您定义要使用的版本控制系统（VCS ）仓库以及要翻译的文件的掩码。
- 在组件配置 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（与组件配置 中定义的掩码匹配）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。

注解: 您可以使用 *Weblate internal URLs* 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 ）的组件时，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。

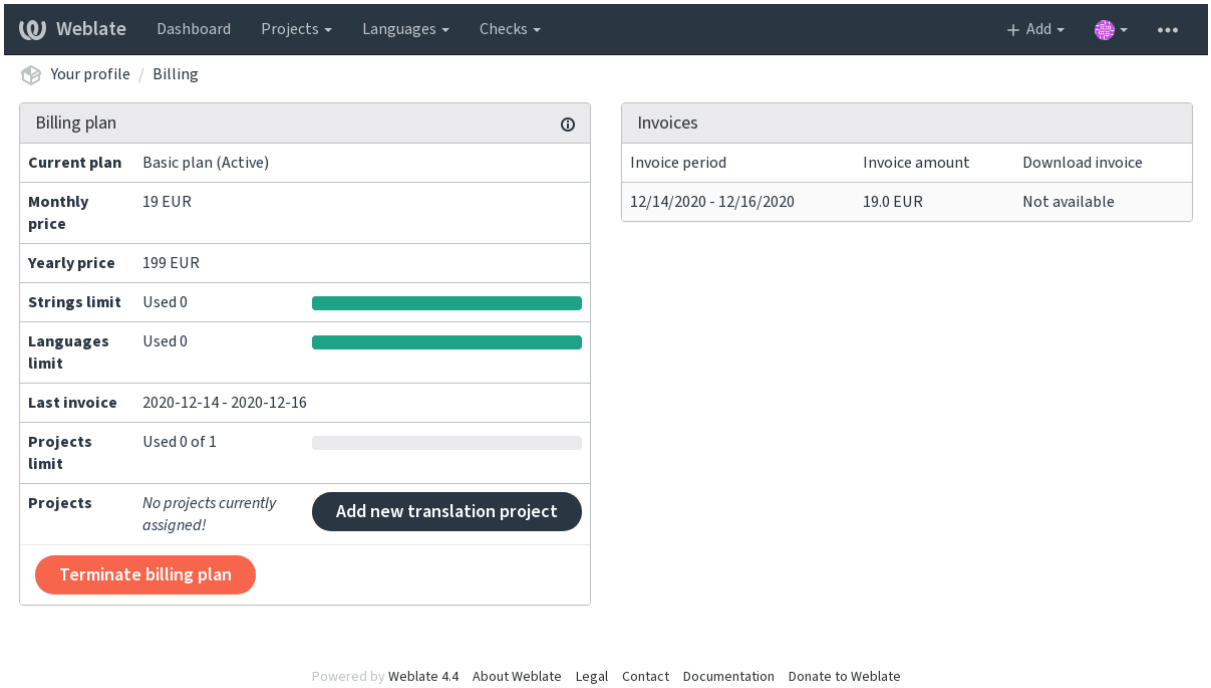
2.7.2 添加翻译项目和组件

在 3.2 版更改: 已包含用于添加项目和组件的界面，您不再需要使用*Django 管理界面*。

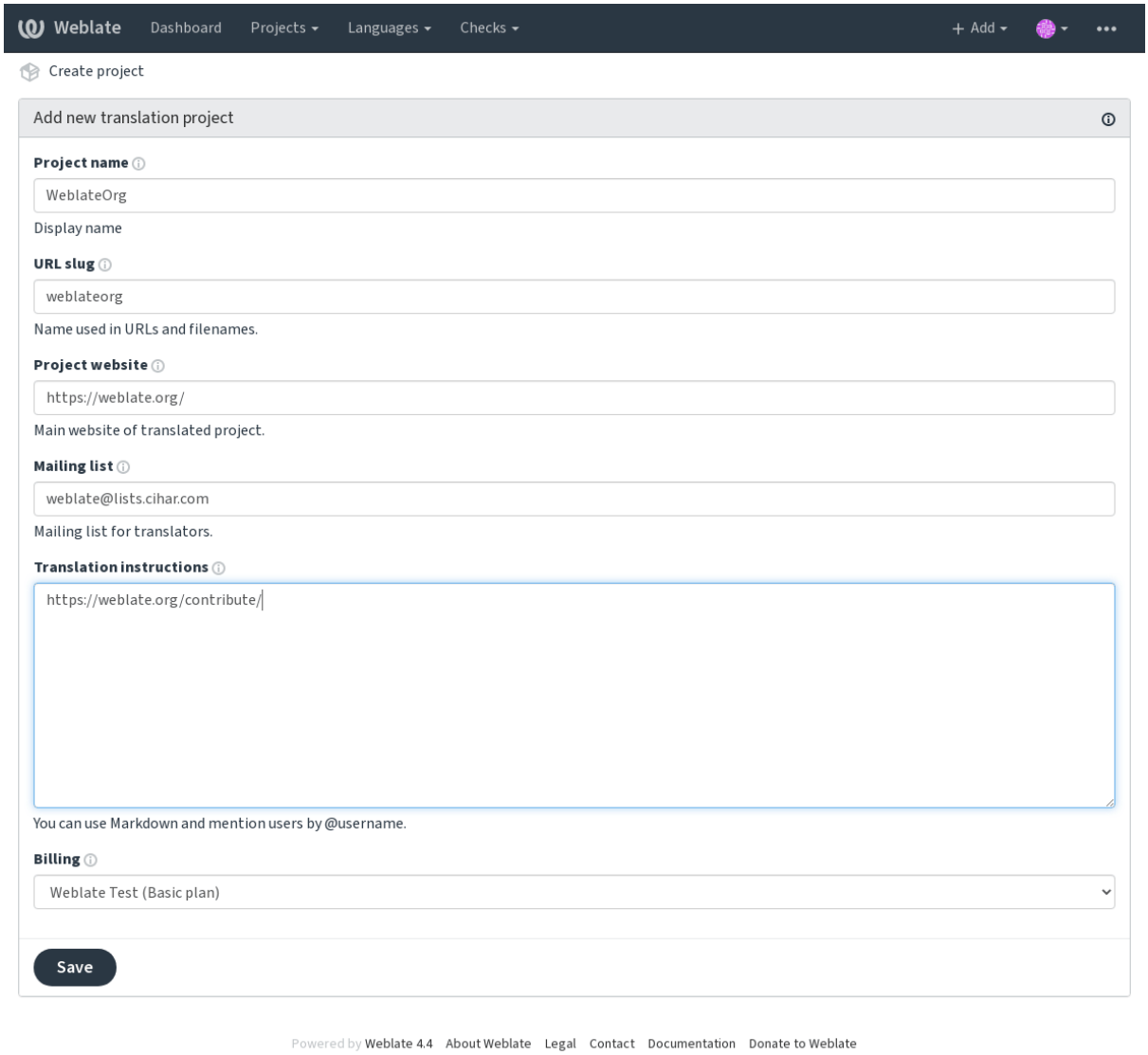
在 3.4 版更改: 现在，添加组件的过程是多阶段的，可以自动发现大多数参数。

根据你的权限，新的翻译项目和组件可以被创建。具备:guilabel:‘Add new projects‘权限的用户总是可以这么做。如果使用付费托管，你还可以从管理账单的用户账户基于套餐限额创建它们。

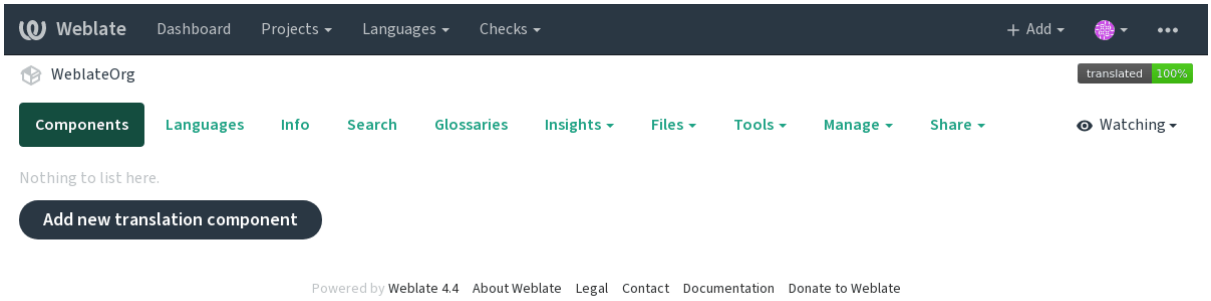
您可以在单独的页面上查看当前的结算方案：



您可以从此处开始创建项目，也可以使用导航条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组件的操作。创建组件的过程是多阶段的，并自动检测大多数翻译参数。有几种创建组件的方法：

从版本控制 从远程版本控制仓库创建组件。

从现有组件 通过选择不同的文件为现有组件创建其他组件。

其他分支 仅针对不同分支，为现有组件创建其他组件。

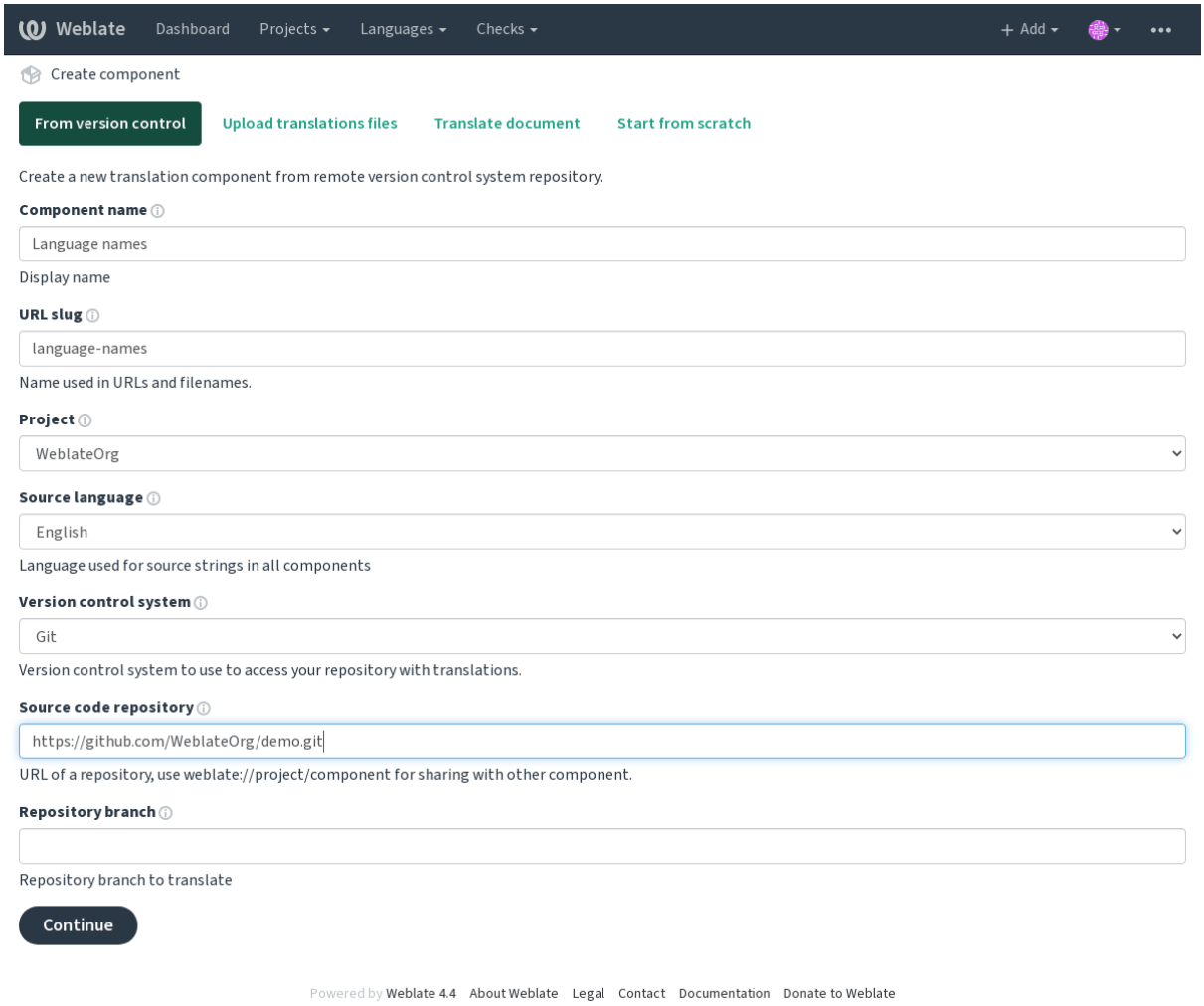
上传翻译文件 如果您没有版本控制或不想将其与 Weblate 集成，则将翻译文件上传到 Weblate。您以后可以使用网络界面或 API 更新内容。

翻译文档 上传单个文档并进行翻译。

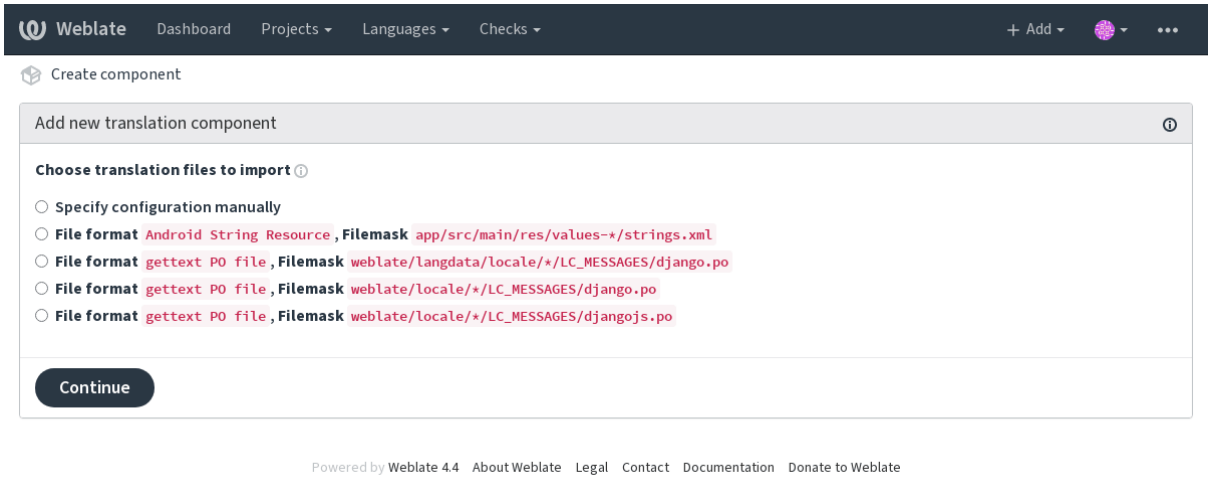
从头开始 创建空白翻译项目并手动添加字符串。

一旦有了现有的翻译组件，就可以使用同一仓库轻松地为其他文件或分支添加新的组件。

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，您检查翻译组件信息并填写可选详细信息：

Weblate Dashboard Projects Languages Checks + Add

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.

File format

gettext PO file

Filemask

weblate/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

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.

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

Adding new translation

Create new language file

How to handle requests for creating new translations.

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 filemask.

Source language

English

Language used for source strings in all components

You will be able to edit more options in the component settings after creating it.

Save

Powered by Weblate 4.4 About Weblate Legal Contact Documentation Donate to Weblate

参见:

Django 管理界面, 项目配置, 组件配置

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组件。这个项目就像一个架子，里面堆放着真正的翻译。同一项目中的所有组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组件（除非在组件配置中关闭），请参见 Memory Management。

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。

项目标识串

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。

邮件列表

译者可以在其中讨论或评注释译的邮件列表。

翻译说明

指向更多网站的 URL，为翻译人员提供了更详细的说明。

设置 Language-Team 头

Weblate 是否应管理 `Language-Team` 头（目前这是仅*GNU gettext* 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。

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。

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。

启用复查

允许复核翻译的工作流程，请参见专门的审核者。

启用来源评论

允许复核源字符串的工作流程，请参见源字符串复查。

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, *Quality gateway for the source strings*, 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。当您的存储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`

使用非标准编码：`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:

分析语言代码

2.7.4 组件配置

组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例子。

注解: 推荐将翻译文件保持在合理的大小——在您的案例中使用任何合理的工具（独立的 app 或插件、书籍的章节或网站）来分割翻译文件。

Weblate 能够轻松处理 10000 个字符串，但大的翻译组件的分割工作和翻译者之间的协调更困难。

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组件中，并且从中取出翻译：

组件名称

冗长组件名称，用于显示组件的名称。

组件标识串

适用于 URLs 的组件名称。

组件项目

组件所属的项目配置。

版本控制系统

使用的版本控制系统（VCS），细节请参见：版本控制集成。

源代码库

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*Accessing repositories* 。

提示： 这可以或者是真实的版本控制系统（VCS）的 URL，或者是 `weblate://project/component`，指示了仓库应该与其它组件分享。更多细节请参见*Weblate internal URLs*。

代码库推送 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。

参见：

关于如何指定仓库 URL 的更多细节请见*Accessing repositories*，并且关于从 Weblate 推送更改的更多细节，请参见推送 *Weblate* 的更改 。

代码库浏览器

用于显示源文件（已使用消息的位置）仓库浏览器的的 URL。当空白时将不生成这样的连接。您可以使用模板标记。

例如在 GitHub 上，使用像："https://github.com/WeblateOrg/hello/blob/{{branch}}/{{filename}}#L{{line}}"那样的一些东西

在您的路径相对于不同文件夹的时候，您会想使用 `parentdir` filter (see 模板标记): `https://github.com/WeblateOrg/hello/blob/{{branch}}/{{filename|parentdir}}#L{{line}}`，来剥除前导文件夹

已导出代码库 URL

由 Weblate 进行的更改被导出的 URL。当不使用持续本地化集成 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。

推送分支

用于推送更改的分支，留为空白来使用仓库分支。

注解： 此功能目前只支持 Git、GitLab 和 GitHub，无法在其他 VCS 集成中工作。

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组件。

例如 "po/*.po" 或 "*locale*/LC_MESSAGES/django.po"。

如果文件名包含特殊字符（例如 "[，]"），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，用于单语言组件。

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

对于单语言组件 是否允许编辑译文模板文件。

中间语言文件

对于单语言组件 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。

当设置时，源翻译基于这个文件，但所有其它的基于单语言译文模版语言文件。在字符串在源翻译中没有被翻译的情况下，会禁止翻译为其它语言。这提供了*Quality gateway for the source strings*。

参见:

Quality gateway for the source strings, 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？

新翻译的译文模版

用于生成新翻译的译文模板文件，例如 gettext 的 `.pot` 文件。

提示: 在很多单语言格式中，Weblate 默认以空白文件开始。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:

添加新的翻译, 添加新翻译, 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

用于汇报上游缺陷的电子邮箱地址。Weblate 中做出的任何字符串注释的通知，也由这个地址接收。

允许同步翻译

您可以关闭项目内从其它组件到这个组件的翻译的传播。这真正依赖于您在翻译的是什么，有时最好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。

默认值可以通过`DEFAULT_TRANSLATION_PROPAGATION` 来更改。

启用建议

对于这个组件，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。

翻译许可证

翻译的许可（不需要与源代码的许可相同）。

贡献者协议

翻译此内容前需同意的协议。

添加新翻译

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，项目维护者将收到有关该语言的通知。由他们决定是否向仓库添加（或不添加）语言。

显示翻译介绍 向用户显示的页面链接描述了开始新翻译的过程。如果需要更正式的流程（例如，在开始实际翻译之前组成人员团队），请使用此选项。

创建新语言文件 用户可以选择语言，然后 Weblate 会自动为其新建文件并开始翻译。

禁用添加新翻译 用户将无法选择开始新的翻译。

参见：

adding-translation.

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

您可以配置如何处理上游仓库的升级。对于一些版本控制系统（VCS），这会不需要支持。更多细节请参见结合或变基。

默认值可以由:setting:‘DEFAULT_MERGE_STYLE‘更改。

提交、添加、删除、合并以及插件消息

当提交翻译时使用的消息，请参见模板标记。

默认值可由*DEFAULT_ADD_MESSAGE*, *DEFAULT_ADDON_MESSAGE*, *DEFAULT_COMMIT_MESSAGE*, *DEFAULT_DELETE_MESSAGE*, :setting:‘DEFAULT_MERGE_MESSAGE‘来更改。

提交者姓名

用于 Weblate 提交的提交者名称，作者总是真实的翻译者。在一些版本控制系统（VCS）中这可能不被支持。

默认值可以由*DEFAULT_COMMITER_NAME* 来更改。

提交者邮箱

用于 Weblate 提交的提交者电子邮箱，作者总是真实的翻译者。在一些版本控制系统（VCS）中这可能不被支持。默认的值可以在:setting:‘DEFAULT_COMMITER_EMAIL‘中更改。

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 将更改提交给内部仓库，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*。

对变更进行提交的延时时间

在它们由后台任务或由*commit_pending* 管理命令提交前，设置如何得到旧的更改（几个小时内）。一旦有至少一个比这个时间段更旧，组件中所有的更改就会提交。

默认值可以由*COMMIT_PENDING_HOURS* 更改。

出错时锁定

允许在仓库故障（拉取、推动或合并失败）时锁定组件。这种情况下的锁定，避免了添加另一个必须由手动解决的冲突。

一旦仓库没有故障留下来了，组件将会自动解锁。

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，你也许会想要选择 *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。这可以用于限制 Weblate 管理的语言列表。

注解： 单出现在文件名中时，您需要列出语言代码。

过滤的一些例子：

过滤器的描述	正则表达式
只有选择的语言	`^(cs|de|es)$`
排除的语言	`^(?!(it|fr)$).+$`
只筛选两个字母的代码	`^..$`
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。

注解： 多数字段可以由项目所有者或管理者在 Weblate 界面上编辑。

参见：

Weblate 支持 Git 和 Mercurial 意外的 VCS 吗？, alerts

优先权

高优先级的组件将最先提供给翻译者。

受限制的访问

组件默认对访问项目的任何人都可见，即使不能在组件中进行任何更改。这会容易地使翻译在项目内保持一致。

在您想要明确地授权访问这个组件的情况下来允许——项目层次的权限将不会应用，并且您必须指定组件或组件列表级别的权限，来授权访问。

默认只可以由*DEFAULT_RESTRICTED_COMPONENT* 更改。

提示： 这也应用于项目管理——请确认切换状态后，您不会丢失对组件的访问。

2.7.5 模板标记

Weblate 在需要提供文本的几个地方使用简单的标记语言。它基于 The Django template language，因此能够非常强大。

当前它用在：

- 提交消息格式，请参见*组件配置*
- **几个插件**
  - *组件发现*
  - *统计数据生成器*
  - *从附加组件执行脚本*

可以在组件模板中得到后面的变量：

{{ language_code }} 语言代码

{{ language_name }} 语言名称

{{ component_name }} 组件名称

{{ component_slug }} 组件标识串

{{ 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 }} 翻译的字符串计数

{{ stats.translated_percent }} 翻译的字符串百分比

{{ stats.allchecks }} 检查失败的字符串数量

{{ stats.allchecks_percent }} 检查失败的字符串百分比

{{ author }} 当前提交的作者，只在提交范围可用。

{{ addon_name }} 当前执行的插件名称，旨在插件提交信息中可用。

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

取回版本控制系统（VCS）仓库，并将翻译导入 Weblate，依赖于您的翻译的大小，这会是漫长的过程。这里是一些提示：

优化配置

对于测试并调试 Weblate，默认的配置是有用的，当用于生产设置时，您应该进行一些调整。它们中的很多都对形成具有巨大的冲击。特别是，更多细节请查看*生产设置*：

- 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

一些质量检查可以使非常昂贵的，而如果不需要，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 自动新建组件

在您的项目有十多个翻译文件的情况下（例如不同的 gettext 域，或安卓 App 的几），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*组件发现* 插件，通过命令行来实现。

为了使用插件，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）新建组件，并且在这个组件上安装插件。

对于管理命令，您需要新建包含所有组件的项目，然后运行*import_project* 或*import_json*。

参见:

管理命令, *组件发现*

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 分析语言代码

当分析翻译时，Weblate 试图将语言代码（通常为 ISO 639-1 ）映射到现有的任何语言对象。

您可以通过*语言别名* 在项目层次来进一步调整这种映射。

如果不能找到完全匹配，将尝试将其最好地适配到现有语言（例如对给定的语言忽略默认国家代码——选择 `cs` 而不是 `cs_CZ` ）。

如果这也失败了，将使用默认值创建一个新的语言定义 (从左到右的文本方向，一个复数)。自动创建的带有代码 `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 做贡献*），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*

2.8.2 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。

2.8.3 内置语言定义

Weblate 中包括了超过 550 种语言的定义，并且每次发布时列表都在扩大。无论何时更新 Weblate 时（更特别地是无论何时执行 **weblate migrate**`时，请参见:ref:`generic-upgrade-instructions**）。语言数据库都被更新，来包括 Weblate 上市时的所有语言定义。

这个特性可以使用 UPDATE_LANGUAGES`来禁止。还可以使用:djadmin:`setuplang 来强制更新数据库，从而匹配 Weblate 内建数据。

2.8.4 歧义语言代码和宏语言

在很多情况下，为翻译使用宏语言代码不是个好主意。典型出问题的情况会是库尔德语，会根据实际的变体以阿拉伯语或拉丁语脚本撰写。为了在 Weblate 中得到正确的行为，推荐只使用单独的语言代码，并且避免宏语言。

参见:

宏语言定义 <https://iso639-3.sil.org/about/scope#Macrolanguages>，宏语言列表 <https://iso639-3.sil.org/code_tables/macrolanguage_mappings/data>

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。对于多数语言这个属性都能够正确地自动监测。

复数数量

语言中使用的复数的数量。

复数式

Gettxt 兼容的复数公式，用于确定根据给定的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

2.8.6 添加新的翻译

在 2.18 版更改: 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。

一些格式希望以空文件开始，并只包含已翻译的字符串（*Android string resources*），而其它希望使所有的 key 出现（例如*GNU gettext*）。在一些情形下，这真正比依赖于格式，而更依赖于用来处理翻译的框架（例如通过*JSON files*）。

当在组件配置 中指定新翻译的译文模版 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。

当新翻译的译文模版 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 依赖于文件格式，对于其中的多数使用 POSIX。

POSIX 风格使用下划线作为分隔 典型地由 gettext 和相关工具使用，生成像 `pt_BR` 那样的语言代码。

POSIX 风格使用下划线作为分隔，包括国家/地区代码 POSIX 风格的语言代码即使不必要时也包括国家代码；例如 `cs_CZ`）。

GCP 风格使用连字符作为分隔 典型地在 web 平台上使用，生成像 `pt-BR` 那样的语言编码。

GCP 风格使用连字符作为分隔，包括国家/地区代码 BCP 风格的语言代码即使不必要时也包括国家代码；例如 `cs-CZ`）。

Android 风格 只在安卓 apps 中使用，生成像 `pt-rBR` 那样的语言代码。

Java 风格 由 Java 使用——多数 BCP 带有用于中文的遗产代码。

此外，语言别名 中定义的任何映射都反向应用。

注解: Weblate 当解析翻译文件是识别所有这些，上面的设置只影响如何新建新的文件。

参见:

语言代码, 分析语言代码

2.9 持续本地化集成

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。

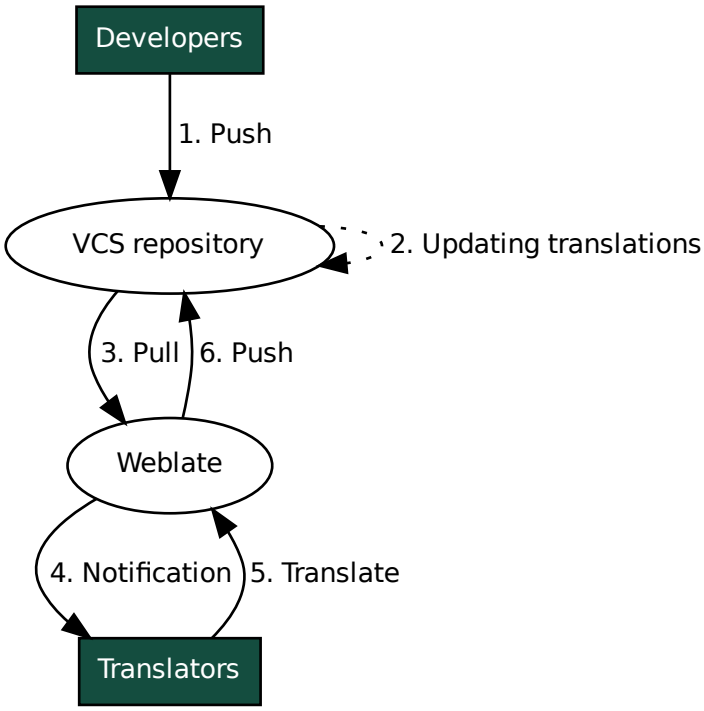
参见:

/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，便会根据翻译者的订阅设置通知他们。
5. 翻译者使用 Weblate Web 界面提交翻译，或上传离线更改。
6. 翻译者完成后，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

应该新建一些从他们的源来更新后端仓库的方式。

- 使用通知钩子 来与多数常见的代码托管服务集成
- 在仓库管理中或使用 API 或 *Weblate* 客户端 来手动触发更新
- 允许`AUTO_UPDATE` 在你的 Weblate 事例上自动更新所有组件
- 执行`updategit` （选择项目，或 *–all* 来更新全部）

每当 Weblate 更新存储库时，更新后插件都将被触发，请参见：附加组件 。

避免合并冲突

当相同的文件在 Weblate 之内与之外都更改时导致 Weblate 的合并冲突。有两种方法来处理——避免在 Weblate 之外编辑，或者将 Weblate 集成到你的更新过程中，从而在更新 weblate 之外的文件之前刷新更改。

第一种方法容易用于单语言文件——可以添加 Weblate 之内的新字符串，并将文件的整个编辑留在那里。对于双语言文件，通常存在某种消息提取过程而从源代码产生翻译文件。在一些情况下，这可以分成两部分——一部分用于提取过程产生模板（例如使用 **xgettext** 产生 gettex POT ），然后下一步过程将它合并到真正的翻译中（例如使用 **msgmerge** 更新 gettext PO 文件）。可以在 Weblate 中执行第二步，它将确认在这个操作前所有待定的更改都包括进去了。

第二种方法可以这样实现，使用 API ，当你在自己一侧进行更改时，强制 Weblate 推送所有待定的更改，并锁定翻译。

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
```

如果多个组件分享相同的仓库，需要分别将他们全部锁定：

```
wlc lock foo/bar
wlc lock foo/baz
wlc lock foo/baj
```

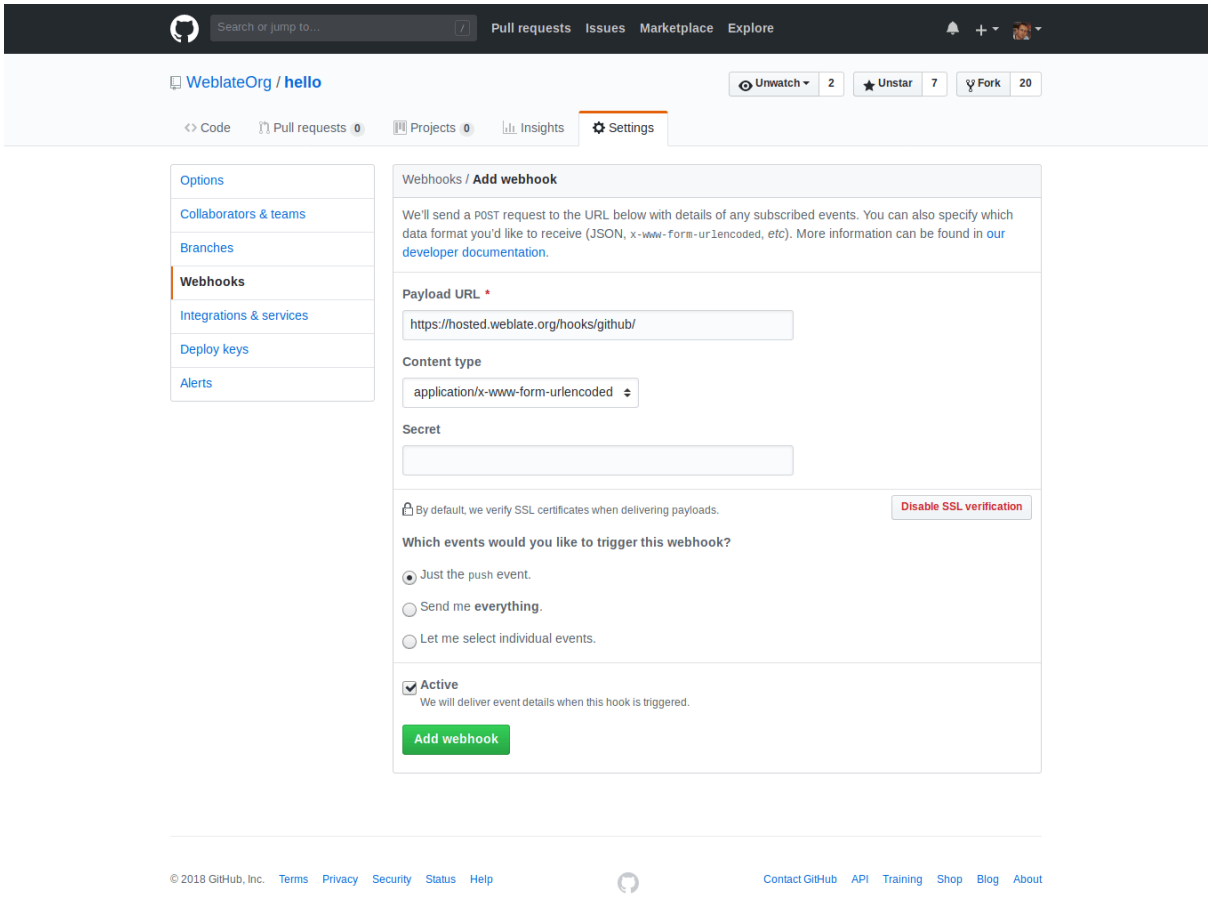
注解： 例子使用了 *Weblate 客户端* ，这需要配置（API 密钥）来远程控制 Weblate 。可以通过使用 HTTP 客户端代替 wlc ，例如 curl 来实现，请参见 API 。

从 GitHub 自动接收更改

Weblate 伴随 GitHub 本地支持。

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/` 。

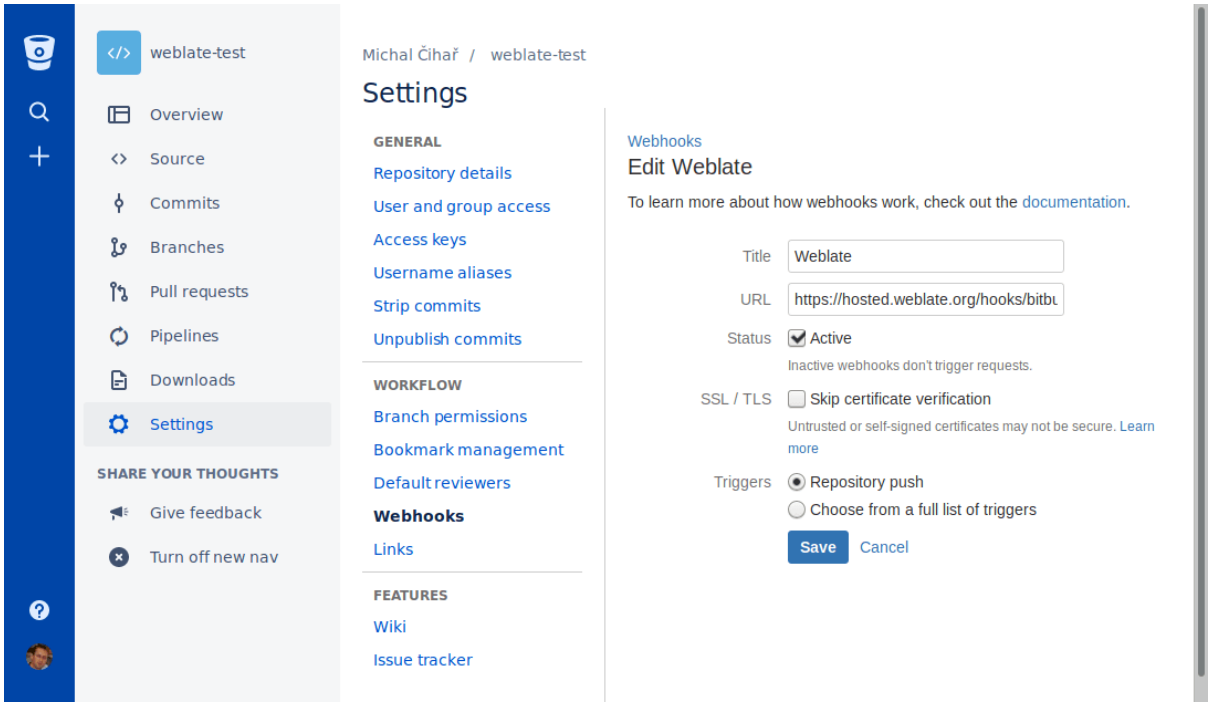
可以将其他设置保留为默认值（Weblate 可以处理内容类型，并只消费 *push* 事件）。

参见:

`POST /hooks/github/`, *Accessing repositories from Hosted Weblate*

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/`, *Accessing repositories from Hosted Weblate*

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/`）。

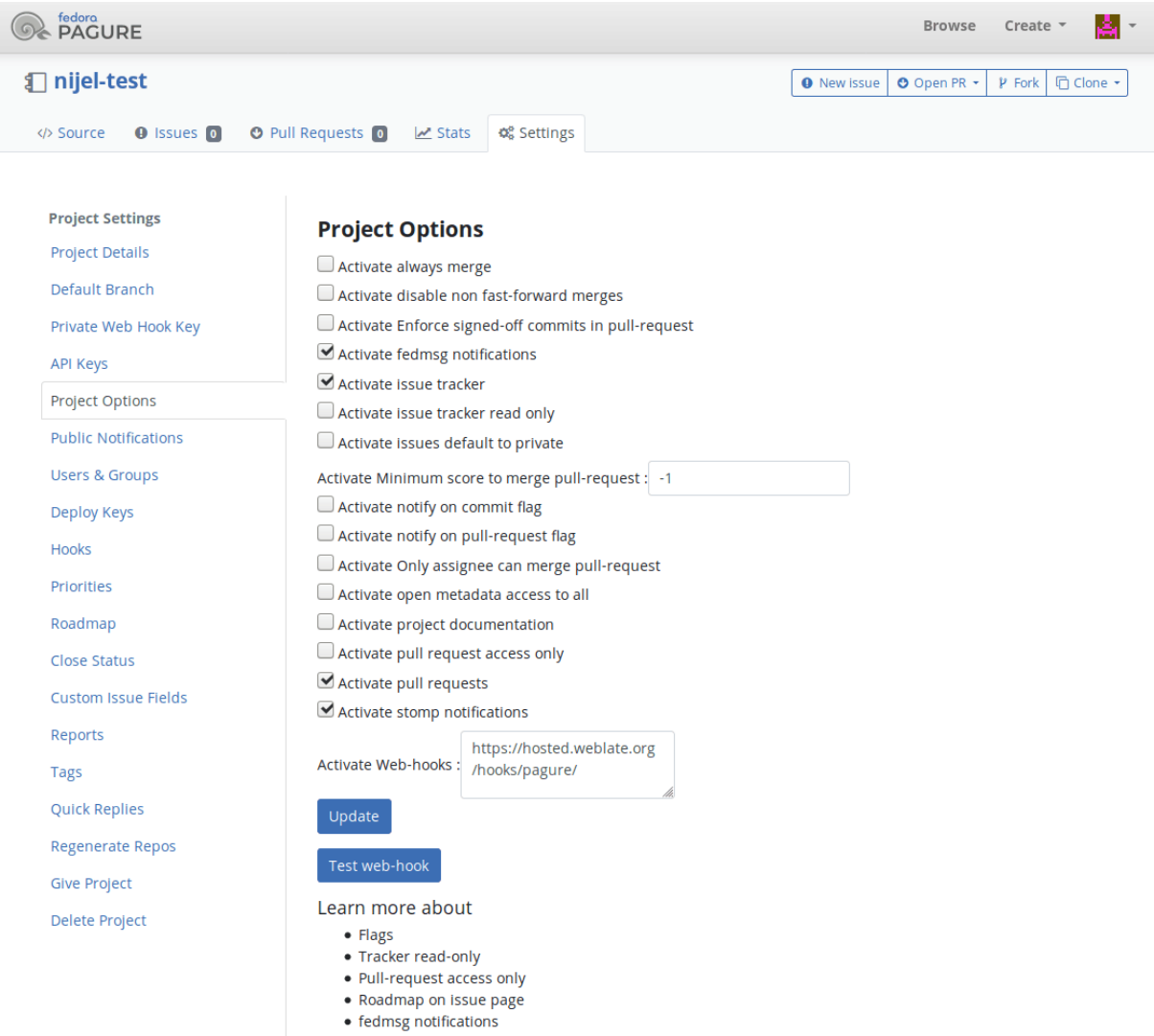
参见:

`POST /hooks/gitlab/`, *Accessing repositories from Hosted Weblate*

从 Pagure 自动接受更改

3.3 新版功能.

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/, *Accessing repositories from Hosted Weblate*

从 Azure Repos 自动接收更改

3.8 新版功能.

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/*, *Accessing repositories from Hosted Weblate*

从 Gitea Repos 自动接收更改

3.9 新版功能.

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 <https://docs.gitea.io/en-us/webhooks/>'_, *POST /hooks/gitea/*, *Accessing repositories from Hosted Weblate*

从 Gitee Repos 自动接收更改

3.9 新版功能.

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 <https://gitee.com/help/categories/40>'_, *POST /hooks/gitee/*, *Accessing repositories from Hosted Weblate*

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

每个翻译组件可以新建推送 URL（请参见*代码库推送 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 *Repository maintenance* 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* 、*GitLab*、vcs-pagure'的拉取请求或:ref:'vcs-gerrit 审核。你可以通过在*组件配置* 中选择 *GitHub* 、*GitLab* 、*Gerrit* 或 *Pagure* 作为*版本控制系统* 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	代码库推送 *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
来自叉子的 GitHub 拉取请求	*GitHub*	空	空
来自分支的 GitHub 拉取请求	*GitHub*	SSH URL[1]	分支名称
来自叉子的 GitLab 结合请求	*GitLab*	空	空
来自分支的 GitLab 结合请求	*GitLab*	SSH URL[1]	分支名称
来自分叉的 Pargue 合并请求	*Pagure*	空	空
来自分支的 Pagure 合并请求	*Pagure*	SSH URL[1]	分支名称

注解: 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。

[1] 在*源代码库* 支持推送的情况下可以为空。

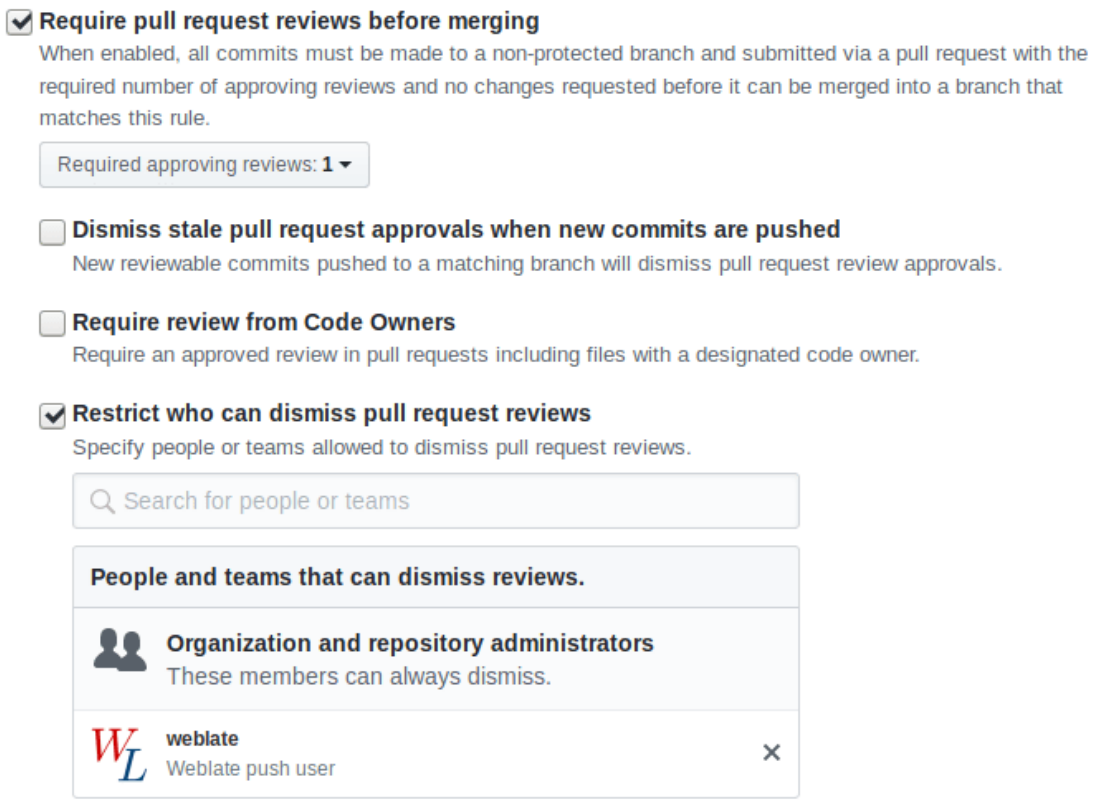
参见:

请参见*Accessing repositories* 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 结合或变基

Weblate 默认将上游仓库结合到自身。在你还通过其他方式访问下层仓库的情况下，这是最安全的方式。在不需要这个的情况下，可以允许上游更改的变基，这会产生较少的融合提交的历史。

注解: 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。

2.9.4 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:

API

2.9.5 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 *Managers*'组是默认允许的，参见*:ref:'privileges*）。

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。
- 更改比组件配置 上定义为 *Age of changes to commit* 的时间段更陈旧。

提示: 每个组件都建立提交。所以在具有很多组件的情况下，仍然会看到很多提交。在这种情况下你会使用压缩 *Git* 提交 插件。

如果想要更频繁地提交更改，并且不检查新旧，可以安排周期定时性任务来执行提交：

```
CELERY_BEAT_SCHEDULE = {
    # Unconditionally commit all changes every 2 minutes
    "commit": {
        "task": "weblate.trans.tasks.commit_pending",
        # Ommiting hours will honor per component settings,
        # otherwise components with no changes older than this
        # won't be committed
        "kwargs": {"hours": 0},
        # How frequently to execute the job in seconds
        "schedule": 120,
    }
}
```

2.9.6 用脚本处理仓库

定制 Weblate 与仓库交互的方式是附加组件 。关于如何通过插件执行外部脚本的信息，请咨询从附加组件执行脚本 。

2.9.7 跨组件保持翻译一致

一旦具有多个翻译组件，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

允许翻译宣传时（这是默认的，请参见组件配置），在所有的组件中字符串匹配时，所有新的翻译自动进行。在所有的组件中这样的翻译都适当地归功于当前翻译的用户。

注解： 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

任何时候字符串不同时不一致的 都会检查启动。可以使用这个来手动复查这样的差异，并选择正确的翻译。

自动化翻译

基于不同组件的自动翻译，可以是跨组件同步翻译的方式。可以或者手动触发（请参见自动化翻译），或者使用插件（自动化翻译）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。

您应该指定组件配置 版权信息。您应该避免那些需要获得贡献者版权协议的情况，尽管这是可能的。

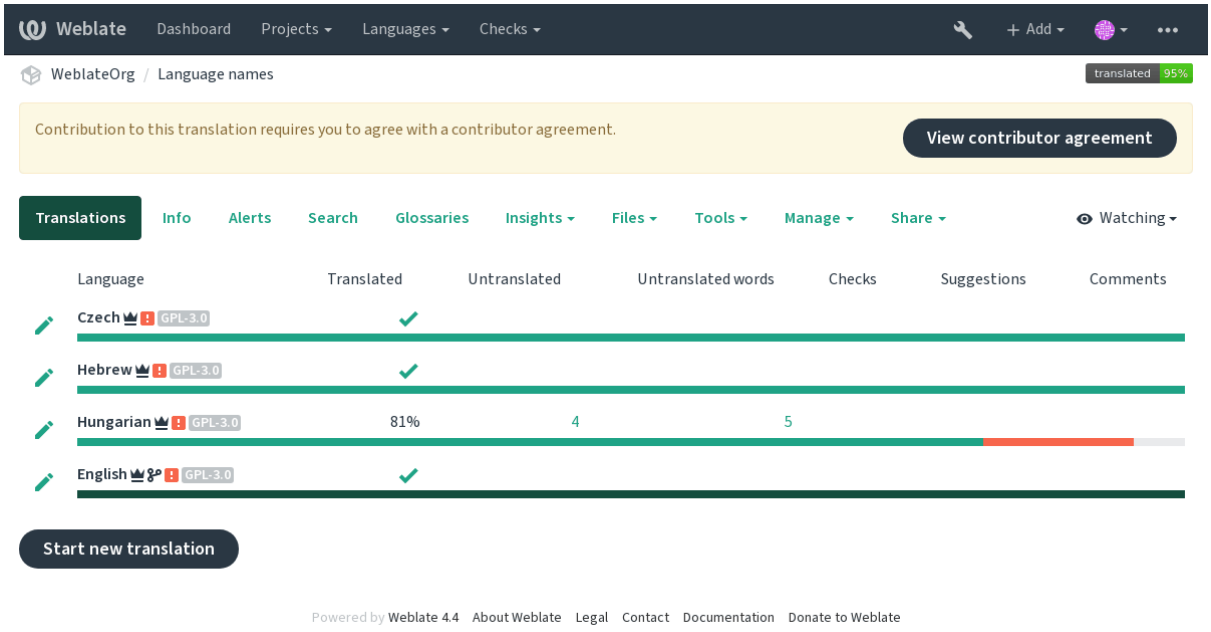
2.10.1 版权信息

在指定版权信息的时候（版权名称和 URL），这个信息显示在各个组件配置 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开源项目，那么您最可能需要事先同意。

2.10.2 贡献者协议

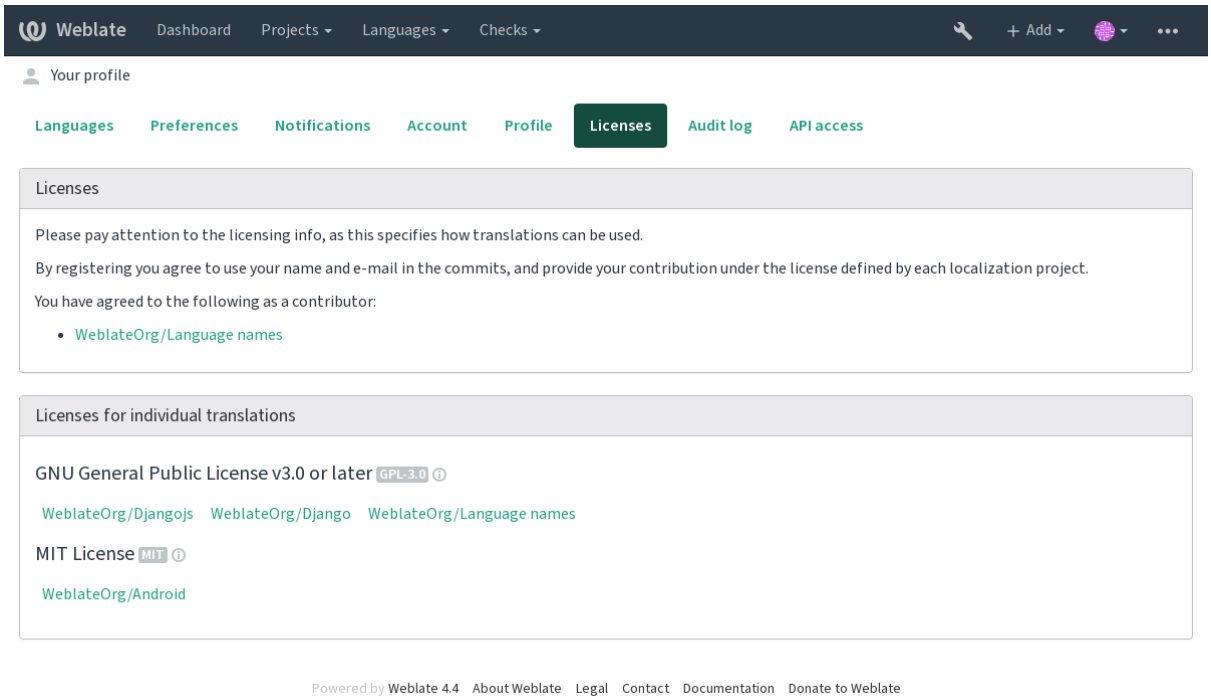
如果您指定了贡献者版权协议，那么只有同意协议的用户能够做出贡献。当进入到翻译时，这是清晰可见的步骤：



输入的文本被格式化为段落，并非且可以包括外部连接。不能使用 HTML 标记。

2.10.3 用户版权

任何用户可以在其简介的实例中看到所有公开项目的所有翻译版权：



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，通过用 *Suggestion voting* 来设置组件配置 配置，来打开投票，并且通过 *Autoaccept suggestions* 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）。

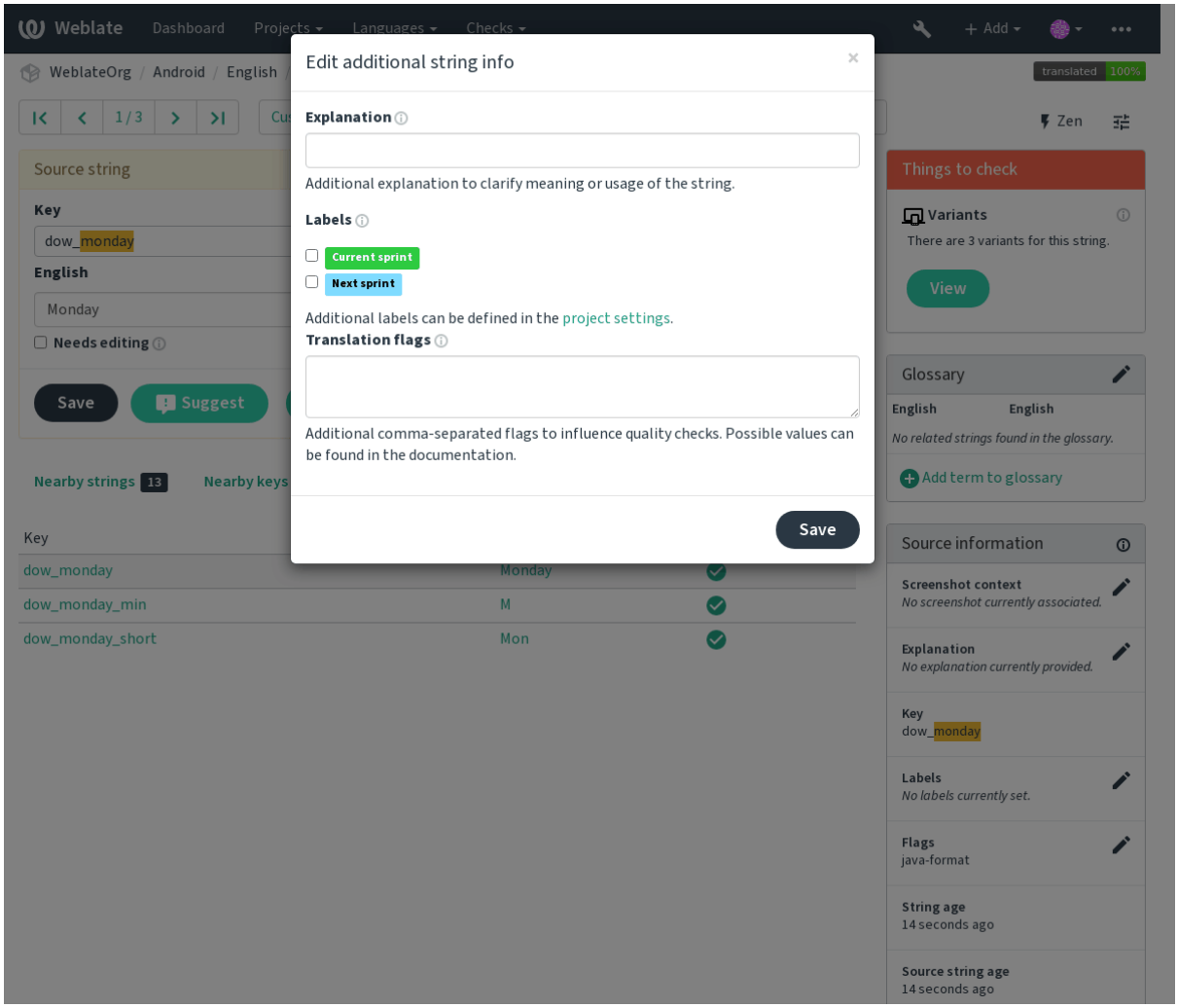
注解: 一旦新建了自动接受，那么普通用户会失去直接存储翻译或接受建议的特权。这可以由 *Edit string when suggestions are enforced* 特权来否决（请参见访问控制）。

您可以将其与访问控制 一起结合到后面的一个设置中：

- 用户建议并对建议进行投票，并且有限的组控制接受什么。——打开投票。——关闭自动接受。——不要让用户存储翻译。
- 用户建议并对建议进行投票，一旦他们之中确定的数量同意则自动接受。——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:



可以通过点击紧邻 *Screenshot context* 或 *Flags* 的“Edit”标签而从翻译界面直接访问。

Weblate Dashboard Projects Languages Checks + Add

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 Clone source NBS

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

Save Suggest Skip

Nearby strings 26 Comments Automatic suggestions Other languages History

Context	English	Czech	State
	Files	Soubory	
	Automatic translation	Automatický překlad	
	Add new translation string	Add new translation string	
	Translation status	Stav překladu	
	%(count)s word	%(count)s slovo	
	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 správně, 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

Source information

Screenshot context
No screenshot currently associated.

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
3 seconds ago

Translation file
weblate/locale/cs/LC_MESSAGES/django.po, string 11

Powered by Weblate 4.4 About Weblate Legal Contact Documentation Donate to Weblate

字符串优先级

2.0 新版功能.

使用 `priority` 标识可以更改字符串优先级来为字符串提供更高优先级，以便更早地进行翻译。

提示: 这可以用于以逻辑的方式将翻译流程排序。

参见:

质量检查

翻译标记

2.4 新版功能.

在 3.3 版更改: 之前被称为 *Quality checks flags*，它不再只配置检查了。

翻译标记的默认设置由翻译组件配置 和翻译文件来决定。然而，你会想要用它来定制每个源字符串。

参见:

质量检查

解释

在 4.1 版更改: 在以前的版本中这被称为 *Extra context*。

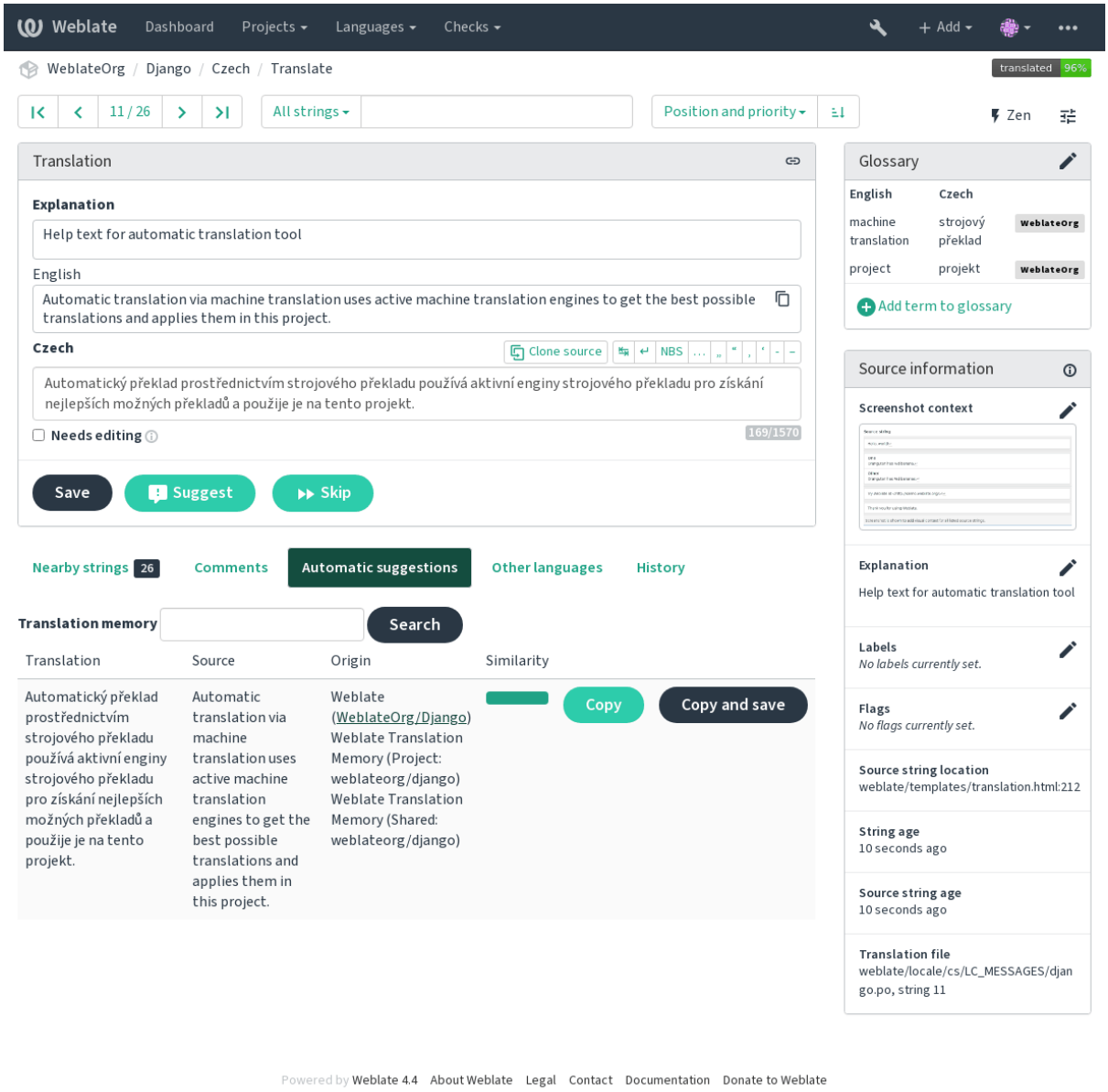
使用解释来明确翻译的范围或翻译的使用。您可以使用 Markdown 来包括连接和其它标志。

字符串的可见语境

2.9 新版功能.

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。

上传的截屏在翻译语境的侧边条中显示：



additional 除了*源字符串另外的信息*，截屏在 *Tools* 菜单下有个单独的管理界面。上传截屏，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。

一旦上传了截屏，那么这个界面处理管理以及源字符串的联系：

Weblate Dashboard Projects Languages Checks + Add

WeblateOrg / Django / Screenshots / Automatic translation

Screenshot has been uploaded, you can now assign it to source strings.

Assigned source strings

Source string	Context	Location	Assigned screenshots	Actions
No source strings are currently assigned!				

Screenshot is shown to add visual context for all listed source strings.

Assign source strings

Source string	Context	Location	Assigned screenshots	Actions
No new matching source 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 4.4 About Weblate Legal Contact Documentation Donate to Weblate

2.12 检查并修正

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 © 2012 - 2021 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/>.
#

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、插件和自动修复 。

2.12.2 定制行为

可以在每个源字符串（在源字符串复核中，请参见源字符串另外的信息 ）或在组件配置 (翻译标记) 中精细调整 Weblate 的行为。一些文件格式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白字符或特定字符。例如：

```
placeholders:"special:value":"other value", regex:.*
```

这里是现在能接受的标记的列表：

rst-text 将文本视为 reStructuredText 文档，影响未更改的翻译 。

md-text 将文本看作 Markdown 文件。

dos-eol 使用 DOS 的行末标记，而不是 Unix 的（`\r\n` instead of `\n`）。

url 字符串应该只包括 URL 。

safe-html 字符串应该是 HTML 安全的，请参见不安全的 *HTML* 网站 。

read-only 字符串应该只读，并且不能在 Weblate 中编辑。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 文件，影响*XML* 语法 and *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 font-spacing 来提供检查，请参见管理字型 。

placeholders:NAME 翻译中需要的占位字符串，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（例如在最大翻译大小或译文最大长度 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名称的文本，例如 `replacements:%s:"John Doe"` 。

regex:REGEX 用于匹配翻译文件的正则表达式，详见正则表达式 。

python-format, c-format, php-format, python-brace-format, javascript-format, c-sharp-format, ja
将所有字符串视为格式字符串，影响格式化字符串, 格式化字符串, 格式化字符串, 格式化字符串, 格式化字符串, 格式化字符串, 格式化字符串, 格式化字符串, 格式化字符串, 格式化字符串, 格式化字符串, 未更改的翻译.

strict-same 使用内建的单词黑名单，来避免“没有翻译”的检查提示。请参见未更改的翻译 。

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-double-space 跳过“双空格”质量检查。

ignore-angularjs-format 跳过“AngularJS 插补字符串”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 模板文本”质量检查。

ignore-i18next-interpolation 跳过“i18next 插补”质量检查。

ignore-java-format 跳过“Java 格式”质量检查。

ignore-java-messageformat 跳过“Java MesseageFormat ”质量检查。

ignore-javascript-format 提高过“JavaScript 格式”质量检查。

ignore-percent-placeholders 跳过“百分号占位符”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 plural 格式”质量检查。

ignore-ruby-format 跳过“Ruby 格式”质量检查。

ignore-vue-format 跳过“Vue I18n 格式”质量检查。

ignore-translated 跳过“已被翻译”质量检查。

ignore-inconsistent 跳过“不一致的”质量检查。

ignore-kashida 跳过“已使用 Kashida 字符”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“译文最大尺寸”质量检查。

ignore-escaped-newline 跳过“换行符 n 不匹配”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“不匹配的断行”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”指令检查。

ignore-same-plurals 跳过“相同复数”质量检查。

ignore-begin-newline 跳过“换行开头”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“未更改的翻译”质量检查。

ignore-safe-html 跳过“不安全的 HTML 网站”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-long-untranslated 跳过“长期未翻译”质量检查。

ignore-multiple-failures 跳过“多项检查失败”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。

注解： 通常规则是，任何检查都使用识别文字来命名 `ignore-*` ，所以能够将规则应用在定制检查中。

每个源字符串的设置，在组件配置 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

3.11 新版功能.

你可以通过设置组件配置 中的:ref:*component-enforced_checks* ，来配置不能省略的检查列表。列出的每个检查在用户界面中都不能省略，并且检查失败的任何字符串都被标记为 *Needs editing* （请参见翻译状态）。

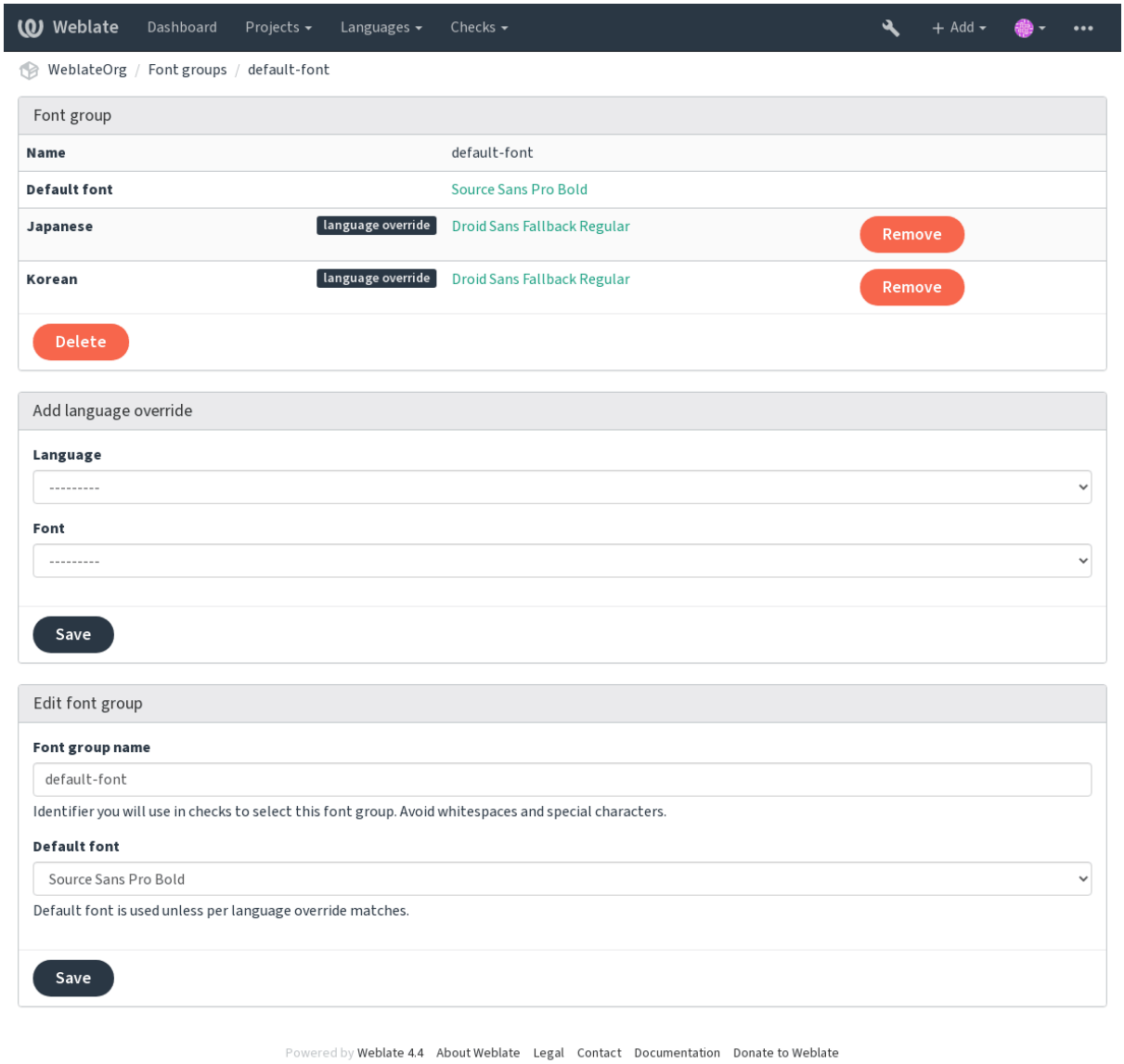
2.12.4 管理字型

3.7 新版功能.

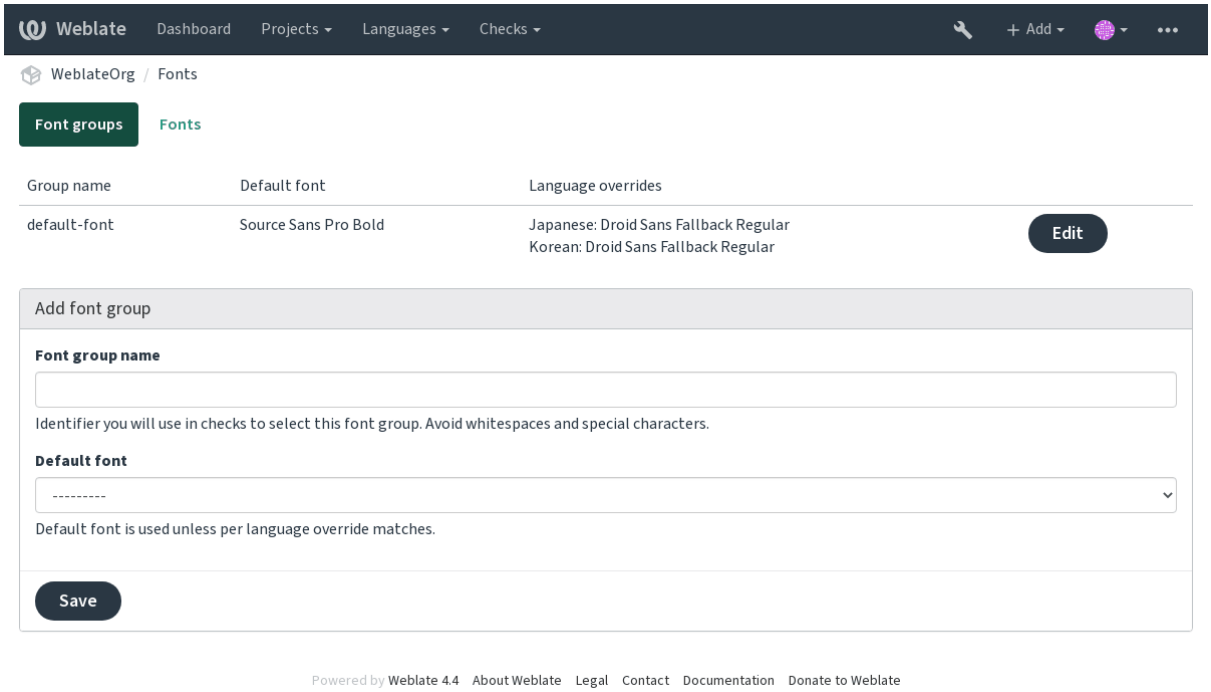
用于计算呈现文本需要的尺寸的:ref: ‘check-max-size ‘检查需要字体被加载进 Weblate 并被一个翻译标识选中 (见定制行为)。

在您的翻译项目 *Manage* 菜单下 *Fonts* 中的 Weblate 字体管理工具提供了接口来上传并管理字体。可以上传 TrueType 或 OpenType 字体，设置 font-groups 并在检查中使用它们。

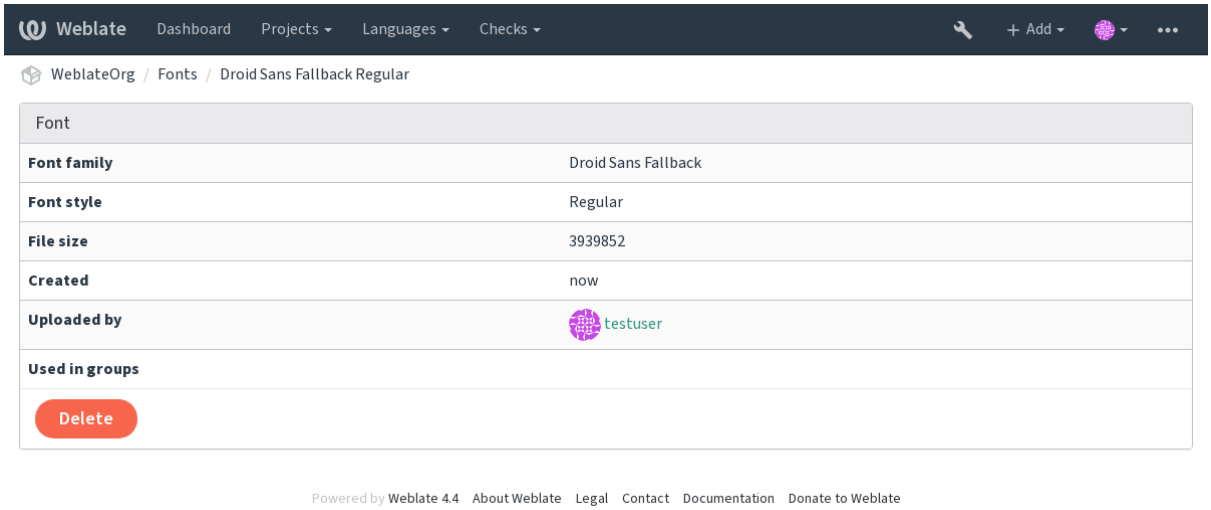
字型组允许为不同语言确定不同字型，这是非拉丁语言中典型需要的：



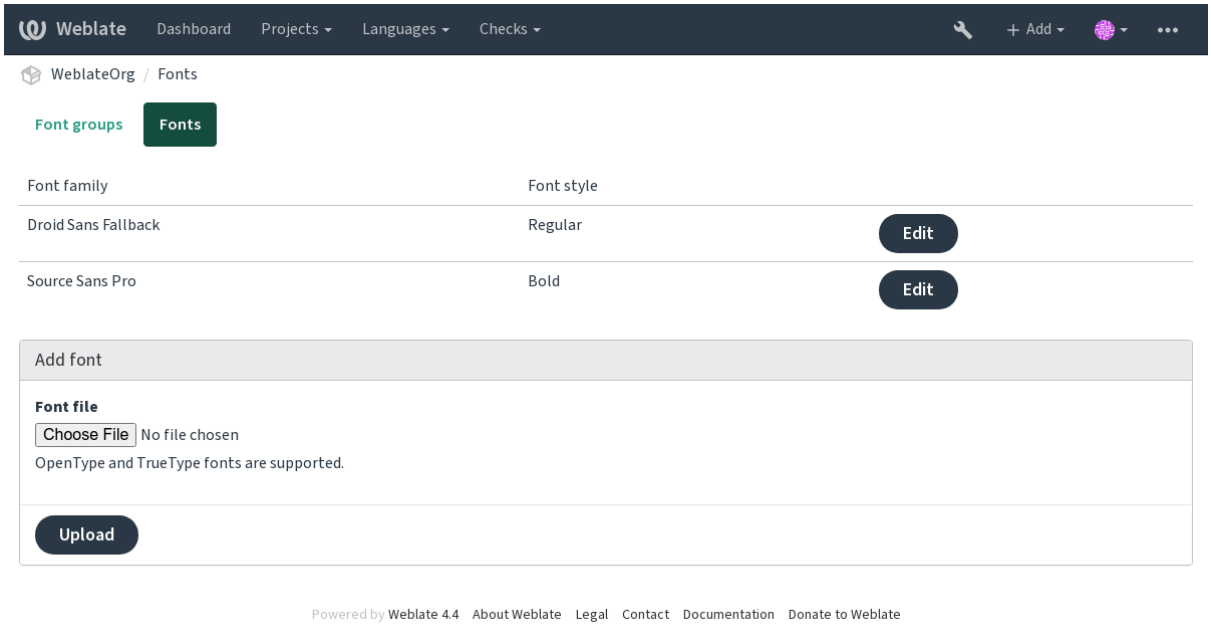
字型组通过名称识别，名称不能包含空白字符或特殊字符，这使它能够容易地用在检查定义中：



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（请参见定制行为）。可能会需要后面这些：

max-size:500 确定最大宽度。

font-family:ubuntu 确定字型组，通过指定其识别文字来使用。

font-size:22 确定字号。

2.12.5 编写自己的检查

Weblate 内建了很大范围的质量检查，（请参见质量检查），尽管可能没有覆盖想要检查的所有内容。可以使用*CHECK_LIST* 来调整执行检查的列表，也可以添加定制的检查。

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）。

一些例子：

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、插件和自动修复 。

检查翻译文本不含有 “foo ”

这是非常简单的检查，只检查翻译中是否丢失了字符串 “foo ”。

```
#
# Copyright © 2012 - 2021 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
```

（下页继续）

(续上页)

```
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/>.
#
"""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
```

检查捷克语翻译文本的复数差异

使用语言信息来检查，验证捷克语中的两种复数形式不同。

```
#
# Copyright © 2012 - 2021 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/>.
#
"""Quality check example for Czech plurals."""

from django.utils.translation import gettext_lazy as _

from weblate.checks.base import TargetCheck


class PluralCzechCheck(TargetCheck):
```

(下页继续)

(续上页)

```
    # 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

    def check_single(self, source, target, unit):
        """We don't check target strings here."""
        return False
```

2.13 机器翻译

内建了几种及其翻译服务的支持，并且每一个可以由管理员使用*MT_SERVICES* 来打开。它们受到自己的使用条款约束，因此请确认允许您以所需要的方式来使用它。

源语言可以在*项目配置* 配置。

2.13.1 amaGama

由 Virtaal 作者运行的*tmserver* 的特殊安装。

通过将 `weblate.machinery.tmserver.AmagamaTranslation` 添加到*MT_SERVICES* 来打开这个服务。

参见:

Installing amaGama, Amagama, **'amaGama 翻译记忆 <https://amagama.translatehouse.org/>'_**

2.13.2 Apertium

开源原件机器翻译平台提供一组有限语言的翻译。

使用 Apertium 的推荐方式是运行您自己的 Apertium-APy 服务器。

通过将 "weblate.machinery.apertium.ApertiumAPYTranslation" 添加到*MT_SERVICES* 中，并设置*MT_APERTIUM_APY* 来打开这项服务。

参见:

MT_APERTIUM_APY, Apertium 网站, Apertium APy 文档

2.13.3 AWS

3.1 新版功能.

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。

1. Turn on this service by adding `weblate.machinery.aws.AWSTranslation` to *MT_SERVICES*.
   2. 安装 *boto3* 模块。
   3. 配置 Weblate 。

参见:

MT_AWS_REGION, *MT_AWS_ACCESS_KEY_ID*, *MT_AWS_SECRET_ACCESS_KEY* 亚马逊翻译文档

2.13.4 Baidu API 机器翻译

3.2 新版功能.

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。

通过将 `weblate.machinery.baidu.BaiduTranslation` 添加到*MT_SERVICES* 并设置*MT_BAIDU_ID* 和*MT_BAIDU_SECRET* 来打开这项服务。

参见:

MT_BAIDU_ID, *MT_BAIDU_SECRET* 百度翻译 API

2.13.5 DeepL

2.20 新版功能.

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。

通过将 `weblate.machinery.deepl.DeepLTranslation` 添加到*MT_SERVICES* ，并设置*MT_DEEPL_KEY* 来打开这项服务。

提示: 在您已经订阅 CAT 工具的情况下，您可能会使用 Weblate 使用的 "v1 API" instead of default "v2"（在这种情况下不是真正的 API 版本）。您可以通过*MT_DEEPL_API_VERSION* 来切换。

参见:

MT_DEEPL_KEY, *MT_DEEPL_API_VERSION*, DeepL 网站, DeepL 定价, DeepL API 文档

2.13.6 Glosbe

几乎每一种活语言的免费字典与翻译服务。

API 免费使用，但受到使用数据源许可的约束。一段时期有少量来自一个 IP 的呼叫，防止侵权。

通过将 `weblate.machinery.glosbe.GlosbeTranslation` 添加到*MT_SERVICES* 来打开这项服务。

参见:

Glosbe website

2.13.7 谷歌翻译

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。

将 `weblate.machinery.google.GoogleTranslation` 添加到*MT_SERVICES* ，并设置*MT_GOOGLE_KEY* 来打开这项服务。

参见:

MT_GOOGLE_KEY, 谷歌翻译文档

2.13.8 Google Translate API V3 （高级版）

Google 云服务提供的机器翻译服务。

这项服务在如何进行身份验证方面不同于前面的服务。将 `weblate.machinery.googlev3.GoogleV3Translation` 添加到*MT_SERVICES* ，并通过后面的设置来允许服务

- *MT_GOOGLE_CREDENTIALS*
- *MT_GOOGLE_PROJECT*

如果 *location* 失败，您还需要指定*MT_GOOGLE_LOCATION* 。

参见:

MT_GOOGLE_CREDENTIALS, *MT_GOOGLE_PROJECT*, *MT_GOOGLE_LOCATION* 谷歌翻译文档

2.13.9 微软认知服务翻译工具

2.10 新版功能.

由 Microsoft 在 Azure 门户提供的机器翻译服务，作为 Cognitive Services 的一种。

Weblate 实施了 Translator API V3.

为了允许这项服务，将 `weblate.machinery.microsoft.MicrosoftCognitiveTranslation` 添加到*MT_SERVICES* ，并设置*MT_MICROSOFT_COGNITIVE_KEY* 。

Translator Text API 第二版

您用于 Translator API V2 的密钥可以用于 API 3 。

Translator Text API 第三版

您需要在 Azure 门户注册，并使用得到的密钥。关于新的 Azure 密钥，您还需要设置*MT_MICROSOFT_REGION* 为您的服务设置地区。

参见:

MT_MICROSOFT_COGNITIVE_KEY, *MT_MICROSOFT_REGION*, 认知服务 - 文本翻译 API, Microsoft Azure 门户

2.13.10 微软术语服务

2.19 新版功能.

Microsoft Terminology Service API 允许您通过 Web 服务，可编程地访问 Langurage Portal 上可用的术语、定义和用户界面（UI ）字符串。

通过将 `weblate.machinery.microsoftterminology.MicrosoftTerminologyService` 添加到*MT_SERVICES* 来打开这项服务。

参见:

微软术语服务 API

2.13.11 ModernMT

4.2 新版功能.

通过将 `weblate.machinery.modernmt.ModernMTTranslation` 添加到*MT_SERVICES* ，并配置*MT_MODERNMT_KEY* 来打开这项服务。

参见:

ModernMT API, *MT_MODERNMT_KEY*, *MT_MODERNMT_URL*

2.13.12 MyMemory

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*MT_MYMEMORY_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。

通过将 `weblate.machinery.mymemory.MyMemoryTranslation` 添加到*MT_SERVICES* ，并设置*MT_MYMEMORY_EMAIL* 来打开这项服务。

参见:

MT_MYMEMORY_EMAIL, *MT_MYMEMORY_USER*, *MT_MYMEMORY_KEY*, MyMemory 网站

2.13.13 网易见外 API 机器翻译

3.3 新版功能.

网易提供的机器翻译服务。

这项服务使用 API ，兵器额您需要从网易得到密钥和密码。

通过将 `weblate.machinery.youdao.NeteaseSightTranslation` 添加到*MT_SERVICES* 并设置*MT_NETEASE_KEY* 和*MT_NETEASE_SECRET* 来打开这项服务。

参见:

MT_NETEASE_KEY, *MT_NETEASE_SECRET* 网易见外翻译平台

2.13.14 tmserver

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：

2. Turn on this service by adding `weblate.machinery.tmserver.TMServerTranslation` to *MT_SERVICES*.

```
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
```

3. 启动 tmserver 来收听您的请求：

```
tmserver -d /var/lib/tm/db
```

4. 配置 Weblate 来与之对话：

```
MT_TMSERVER = "http://localhost:8888/tmserver/"
```

参见:

MT_TMSERVER, tmserver Installing amaGama, Amagama, Amagama 翻译记忆

2.13.15 Yandex 翻译

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。

通过将 `weblate.machinery.yandex.YandexTranslation` 添加到*MT_SERVICES*，并设置*MT_YANDEX_KEY* 来打开这项服务。

参见:

MT_YANDEX_KEY, Yandex 翻译 API, 由 Yandex 翻译驱动

2.13.16 有道智云 API 机器翻译

3.2 新版功能.

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。

通过将 `weblate.machinery.youdao.YoudaoTranslation` 添加到*MT_SERVICES*，并设置*MT_YOUDAO_ID* 和*MT_YOUDAO_SECRET* 来打开这项服务。

参见:

MT_YOUDAO_ID, *MT_YOUDAO_SECRET* 有道智云自然语言翻译服务

2.13.17 Weblate

Weblate 也可以作为机器翻译的来源。它基于 Woosh 全文引擎，并提供精确和和不精确的匹配。

通过将 `weblate.machinery.weblatetm.WeblateTranslation` 添加到*MT_SERVICES* 来打开这项服务。

2.13.18 Weblate 翻译记忆库

2.20 新版功能.

翻译记忆库 也可以用作机器翻译建议的来源。

通过将 `weblate.memory.machine.WeblateMemory` 添加到*MT_SERVICES* ，来打开这项服务。这项服务默认打开。

2.13.19 SAP 翻译中心

SAP 提供的机器翻一下服务。

您需要具有 SAP 账户（并在 SAP Cloud Platform 上允许 SAP Translation Hub ）来使用这项服务。

通过将 `weblate.machinery.saptranslationhub.SAPTranslationHub` 添加到*MT_SERVICES* ，并设置对沙箱或生产 API 的适当访问，来打开这项服务。

注解: 为了访问 Sandbox API ，您需要设置*MT_SAP_BASE_URL* 和*MT_SAP_SANDBOX_APIKEY* 。

为了访问生产 API ，您需要设置*MT_SAP_BASE_URL* 、*MT_SAP_USERNAME* 和*MT_SAP_PASSWORD* 。

参见:

MT_SAP_BASE_URL, *MT_SAP_SANDBOX_APIKEY*, *MT_SAP_USERNAME*, *MT_SAP_PASSWORD*, *MT_SAP_USE_MT* SAP 翻译中心 API

2.13.20 定制化的机器翻译

您还可以通过使用一些 Python 代码来实施您自己的机器翻译服务。这个例子使用 `dictionary` Python 模块来实施一组固定语言的机器翻译：

```
#
# Copyright © 2012 - 2021 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/>.
#
"""Machine translation example."""
```

(下页继续)

(续上页)

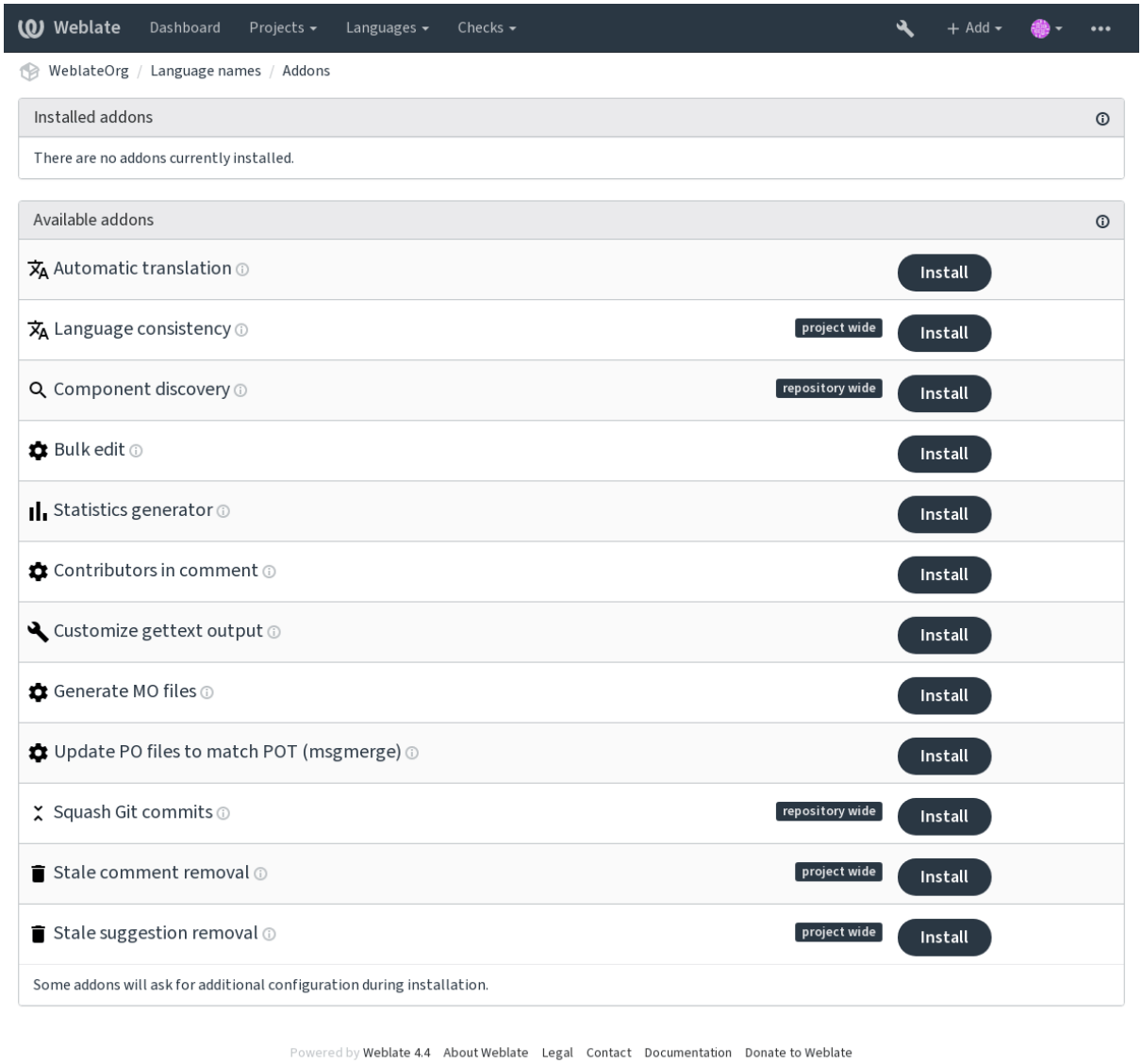
```
import dictionary

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}
```

您可以在*MT_SERVICES* 中列出自己的类，Weblate 就会开始使用它了。

2.14 附加组件

2.19 新版功能.

附加组件提供了自定义翻译工作流的方法。它们可以被安装在翻译组件视图中，并在幕后工作。管理员可从各翻译组件的: 图形用户界面标签: ‘管理‘↓’附加组件’菜单中管理它们。



2.14.1 内置插件

自动化翻译

3.9 新版功能.

使用机器翻译或其他组件自动翻译字符串。

当组件中出现新字符串时，会自动触发此附加组件。

参见:

自动化翻译, 跨组件保持翻译一致

JavaScript 本地化 CDN

4.2 新版功能.

将译文发布到内容交付网络，以便在 JavaScript 或 HTML 本地化中使用。

它可以用于将静态 HTML 网页本地化，或者用于在 JavaScript 代码中加载本地化文件。

在安装时，插件为你的组件产生唯一的 URL ，可以将其包括在 HTML 文件中，使它们本地化。细节请参见 *weblate-cdn* 。

参见:

cdn-addon-config, weblate-cdn, cdn-addon-extract, cdn-addon-html

移除空白字符串

4.4 新版功能.

从翻译文件中删除没有翻译的字符串.

如果不想空字符串出现在翻译文件中的话（例如当本地化库将其显示为空字符串而不是填回源字符串时），要使用这个。

参见:

Weblate 除了更新翻译，还更新翻译文件吗？

清理翻译文件

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？

语言一致性

确保一个项目中的所有组件都为每种要添加的语言进行翻译。

它对未添加到组件中的语言建立空的翻译。

每 24 小时，和在新的语言加入 Weblate 时，检查一次丢失的语言。

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*自动化翻译* 自动翻译新添加的字符串。

组件发现

根据版本控制系统中文件更改的情况来自动添加或删除项目的组件.

每次版本控制系统（VCS ）升级时触发，否则与`import_project` 管理命令相似。这种方式可以在一个版本控制系统（VCS ）中跟踪多个翻译组件。

建立一个至少未来不大可能会消失的主要组件，其他会采用其*Weblate internal URLs* 作为版本控制系统（VCS ）的配置，并且配置它来找到其中的所有组件。

匹配是使用正则表达式完成的，在正则表达式中，强大的功能是配置复杂性的一种折衷。一些常见用例的例子可以在附加组件帮助部分找到。

一旦点击了 *Save* ，将显示匹配组件的预览，可以检查配置是否匹配于自己的需要：

Weblate Dashboard Projects Languages Checks + Add

WeblateOrg / Language names / Addons / Component discovery

Configure addon

- Please review and confirm the matched components.

Component	Matched files
Following components would be created	
Djangojs	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	weblate/locale/hu/LC_MESSAGES/django.po (hu) weblate/locale/he/LC_MESSAGES/django.po (he) weblate/locale/cs/LC_MESSAGES/django.po (cs)

☐ **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.

Language filter

^(cs|he|hu)$

Regular expression to filter translation files against when scanning for filemask.

☑ **Clone addons from the main component to the newly created ones**

☐ **Remove components for inexistant 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.

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 4.4 About Weblate Legal Contact Documentation Donate to Weblate

参见:

模板标记

批量编辑

3.11 新版功能.

批量编辑字符串标记、标签或状态。

新字符串自动贴标签会有用（通过搜索查询 `NOT has:label` 开始，并且添加所需要的标签，直到所有的字符串都适当地被贴上标签）。还以对 Weblate 元数据执行其他任何自动化操作。

例子:

表 1: 自动添加标签到新的字符串

搜索条目	`NOT has:label`
要添加的标签	近期

表 2: 将所有应用商店元数据文件更改日志条目标记为只读

搜索条目	`language:en AND key:changelogs/`
要添加的翻译标记	`read-only`

参见:

批量编辑

将未更改的译文标记为“需要编辑”

3.1 新版功能.

每当新的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。这对于包含全部字符串的文件格式十分有用，即使它们没有被翻译。

将新的源字符串标记为“需要编辑”

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。

将导入的新译文标记为“需要编辑”

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。

统计数据生成器

生成一个包含关于翻译状态详细信息的文件。

可以在文件名和内容中使用 Django 模板，参见模板标记 的具体标记表述。

例如对每个翻译产生摘要文件：

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

在注释中添加贡献信息

在 PO 文件的头部以注释的形式添加贡献者的名字与贡献年份。

PO 文件头包含贡献者与贡献年份的列表:

```
# Michal Čihař <michal@cihar.com>, 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

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

允许自定义 gettext 的输出行为，例如是否启用自动换行。

提供了后面的选项:

- 在 77 个字符处和新一行时来换行
- 仅在换行符处折行
- 不换行

注解: 默认 gettext 在 77 个字符处和新一行时换行。使用 `--no-wrap` 参数时只在新一行时换行。

更新 LINGUAS 文件

添加新的翻译时更新 LINGUAS 文件。

生成 MO 文件

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)

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（如新翻译的译文模版 所配置）匹配。

每当从上游存储库拉取新更改时，这个插件被触发。您可以通过附加配置设定大多数 msgmerge 命令行选项。

参见:

Weblate 除了更新翻译，还更新翻译文件吗？

压缩 Git 提交

推送变更之前压缩 Git 提交。

可以选择后面的模式之一：

3.4 新版功能.

- 所有提交成一个
- 每种语言
- 每个文件

3.5 新版功能.

- 每位作者

原始提交信息保留，但其作者信息丢失，除非选择 *Per author* ，或者定制提交信息来包括它。

4.1 新版功能.

原始提交信息可选地由定制的提交信息覆盖。

预告（提交行像 `Co-authored-by: ...` ）可选地从原始提交信息中去掉，并且添加在去掉的提交信息后面。这还可以为每一位翻译者产生适当的 `Co-authored-by:` 信誉。

自定义 JSON 输出

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

排序 Java 属性文件。

陈旧注释删除

3.7 新版功能.

设置删除注释的时间。

这可以用于删除可能变得过时的陈旧注释。小心使用，因为陈旧的注释不意味着失去了重要性。

陈旧建议删除

3.7 新版功能.

设置删除建议的时间。

可以用于连接建议投票（请参见同行评审），将给定时间内没有得到足够积极票数的建议删除。

更新 RESX 文件

3.9 新版功能.

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 YAML 输出

3.10.2 新版功能.

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类绝对名称即可。

2.14.3 编写附加组件

你也可以编写自己的附加组件，所需要做的是把:class:‘weblate.addons.base.BaseAddon‘归入子类，定义附加组件的元数据，并实现一个会执行处理的回调函数。

参见:

开发附加组件

2.14.4 从附加组件执行脚本

附加逐渐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 © 2012 - 2021 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/>.
#
"""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 descrption
    verbose = _("Execute script before commit")
    description = _("This addon 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、插件和自动修复 。

对于任何给定的组件，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，可以访问后面的环境参数：

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
2.11 新版功能.

当前组件配置的仓库分支。

WL_FILEMASK
当前组件的 Filemask。

WL_TEMPLATE
单语言翻译模板的文件名（可以为空）。

WL_NEW_BASE
2.14 新版功能.

建立新的翻译所使用文件的文件名（可以为空）。

WL_FILE_FORMAT
当前组件使用的文件格式。

WL_LANGUAGE
当前处理的翻译语言（对于组件水平的钩子不可获得）。

WL_PREVIOUS_HEAD
之前更新上的 HEAD （只有当运行过去的更新钩子时可获得）。

WL_COMPONENT_SLUG
3.9 新版功能.

组件标识串用于构建 URL 。

WL_PROJECT_SLUG
3.9 新版功能.

项目标识串用于构建 URL 。

WL_COMPONENT_NAME
3.9 新版功能.

组件名称。

WL_PROJECT_NAME
3.9 新版功能.

项目名称。

WL_COMPONENT_URL
3.9 新版功能.

组件 URL 。

WL_ENGAGE_URL
3.9 新版功能.

项目参与 URL 。

参见:

组件配置

更新后存储库处理

以往更新仓库处理可以用于在版本控制系统（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 只看到提交给版本控制系统（VCS ）的文件，所以需要同意更改作为脚本的一部分。

例如 Gulp ，可以使用后面的代码来执行：

```
#! /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

使用提交脚本在提交给仓库前自动地对翻译做出更改。

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

2.20 新版功能.

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（依赖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，通过使用自动化翻译 或自动化翻译 插件来翻译字符串。

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

3.2 新版功能: 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翻译者共享，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

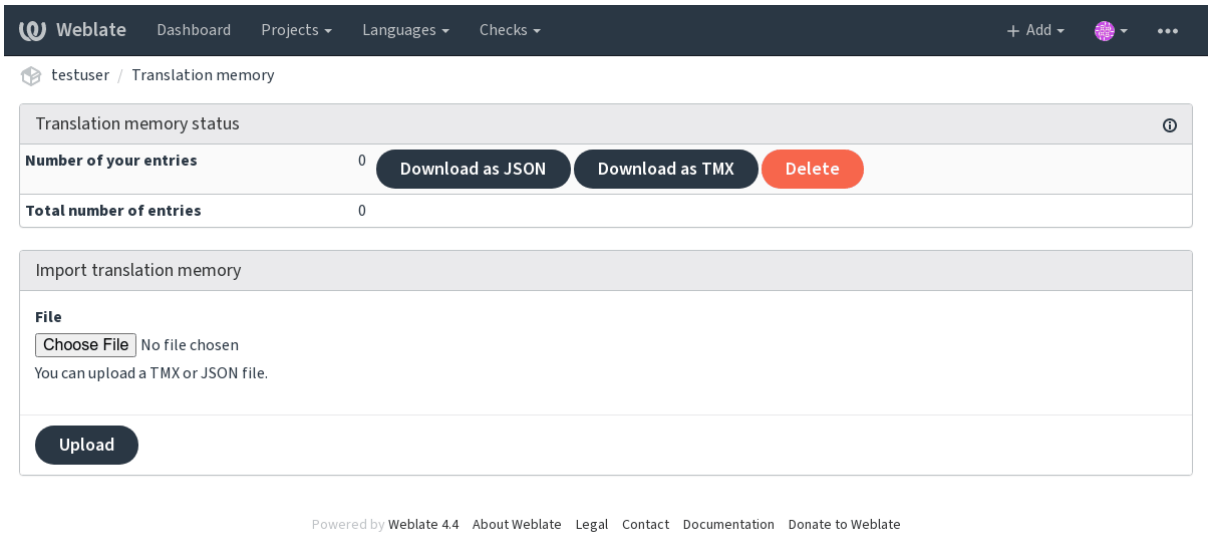
3.2 新版功能.

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

所有的设置存储在 `settings.py` 中，（如 Django 通常那样）。

注解： 在更改这些设置的任何一部分后，需要重新启动 Weblate ——WSGI 和 Celery 两个过程。

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：

还要请查看 Django’s documentation 中关于配置 Django 自身的参数。

2.16.1 AKISMET_API_KEY

Weblate 可以使用 Akismet 检查到来的对垃圾邮件的匿名建议。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

3.6 新版功能.

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

2.14 新版功能.

在 rate 限制应用前，授权尝试失败的最多次数。

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邮件。

默认为 10 。

参见:

频次限制 ,

2.16.5 AUTO_UPDATE

3.2 新版功能.

在 3.11 版更改: 更改原来的开关选项，来区别接受哪个字符串。

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。

注解: 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。

注解： 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

2.14 新版功能.

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

2.15 新版功能.

应该允许几天来使用相同的密码。

注解： 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。

注解： 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` 将字符串开始与结尾的空白字符与元字符串匹配。

`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` 去掉零宽度字符，如果源字符串不包含的话。

`weblate.trans.autofixes.chars.RemoveControlChars` 去掉控制字符，如果源字符串不包含的话。

`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* 网站 ）。

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 BASE_DIR

Weblate 源所在的基本目录。用于默认得到几个其他路径：

- *DATA_DIR*

默认值：Weblate 源的顶层目录。

2.16.11 BASIC_LANGUAGES

4.4 新版功能.

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。

注解： 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 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，Content Security Policy (CSP)

2.16.13 CHECK_LIST

翻译时执行的质量检查列表。

注解: 为实施检查界面的 Python 类提供完全合规的路径。

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。它们在配置的例子 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",
)
```

注解: 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, 定制行为

2.16.14 COMMENT_CLEANUP_DAYS

3.6 新版功能.

在一定天数后删除注释。默认为 `None` ，意味着不删除。

2.16.15 COMMIT_PENDING_HOURS

2.10 新版功能.

通过后台任务方式提交待定的更改之间的小时数。

参见:

组件配置, 对变更进行提交的延时时间, 运行维护任务, *commit_pending*

2.16.16 DATA_DIR

Weblate 文件夹将所有数据存储其中。它包含到版本控制系统（VCS ）仓库的链接，全文索引和外部工具的各种文件。

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 `STATIC_ROOT` 指定。

media Django 媒体文件的默认位置，由 `MEDIA_ROOT` 指定。

vcs 版本控制仓库。

backups 每日备份数据，细节请参考下载的数据用于备份 。

注解: 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
```

默认到 `$BASE_DIR/data` 。

参见:

BASE_DIR, *备份和移动 Weblate*

2.16.17 DATABASE_BACKUP

3.1 新版功能.

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

2.16.18 DEFAULT_ACCESS_CONTROL

3.3 新版功能.

新项目的默认访问控制设置：

0 *Public*

1 *Protected*

100 *Private*

200 *Custom*

如果手动管理 ACL 则使用 *Custom* ，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*, *访问控制*

2.16.19 DEFAULT_RESTRICTED_COMPONENT

4.1 新版功能.

组件限制的默认值。

参见:

项目访问控制, *受限制的访问*, *访问控制*

2.16.20 DEFAULT_ADD_MESSAGE, DEFAULT_ADDON_MESSAGE, DEFAULT_COMMIT_MESSAGE, DEFAULT_DELETE_MESSAGE, DEFAULT_MERGE_MESSAGE

不同操作的默认执行信息，细节请查阅组件配置 。

参见:

模板标记, 组件配置, 提交、添加、删除、合并以及插件消息

2.16.21 DEFAULT_ADDONS

安装在每个创建的组件上的默认附加组件。

注解: 此设置只影响新创建的组件。

例:

```
DEFAULT_ADDONS = {
    # Addon with no parameters
    "weblate.flags.target_edit": {},
    # Addon with parameters
    "weblate.autotranslate.autotranslate": {
        "mode": "suggest",
        "filter_type": "todo",
        "auto_source": "mt",
        "component": "",
        "engines": ["weblate-translation-memory"],
        "threshold": "80",
    },
}
```

参见:

install_addon, *WEBLATE_ADDONS*

2.16.22 DEFAULT_COMMITER_EMAIL

2.4 新版功能.

已创建的翻译组件的提交者电子邮件地址，默认为 `noreply@weblate.org` 。

参见:

DEFAULT_COMMITER_NAME, 组件配置, 提交者邮箱

2.16.23 DEFAULT_COMMITER_NAME

2.4 新版功能.

建立翻译组件的执行者姓名默认为 `Weblate` 。

参见:

DEFAULT_COMMITER_EMAIL, 组件配置, 提交者姓名

2.16.24 DEFAULT_LANGUAGE

4.3.2 新版功能.

例如在:ref:‘component-source_language‘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

2.16.25 DEFAULT_MERGE_STYLE

3.4 新版功能.

任何新组件的合并风格。

- *rebase* - default
- *merge*

参见:

组件配置, 合并方式

2.16.26 DEFAULT_TRANSLATION_PROPAGATION

2.5 新版功能.

翻译传播的默认设置，默认为 `True` 。

参见:

组件配置, 允许同步翻译

2.16.27 DEFAULT_PULL_MESSAGE

用于新的拉取请求的标题，默认为 `'Update from Weblate'` 。

2.16.28 ENABLE_AVATARS

是否为用户打开基于 Gravatar 的头像。默认打开。

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

2.16.29 ENABLE_HOOKS

是否允许匿名远程钩子。

参见:

通知钩子

2.16.30 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*

2.16.31 ENABLE_SHARING

打开/关闭:guilabel: *Share* 菜单，使用户能够将翻译过程分享到社交网络上。

2.16.32 GITLAB_CREDENTIALS

4.3 新版功能.

用于 GitLab 服务器的证明列表。

提示: 如果你想要 Weblate 与它们中的更多进行交互，请使用此功能，对于单一的 GitLab 端点，继续使用*GITLAB_USERNAME* 和*GITLAB_TOKEN*。

```
GITLAB_CREDENTIALS = {
    "gitlab.com": {
        "username": "weblate",
        "token": "your-api-token",
    },
    "gitlab.example.com": {
        "username": "weblate",
        "token": "another-api-token",
    },
}
```

2.16.33 GITLAB_USERNAME

GitLab 用户名，用于发送翻译更新的和并请求。

参见:

GITLAB_CREDENTIALS, *GitLab*

2.16.34 GITLAB_TOKEN

4.3 新版功能.

使用 GitLab 个人访问令牌，用于翻译更新的 API 进行调用。

参见:

GITLAB_CREDENTIALS, *GitLab*, GitLab: Personal access token

2.16.35 GITHUB_CREDENTIALS

4.3 新版功能.

用于 GitHub 服务器的证明列表。

提示: 在想要 Weblate 与它们中的更多进行交互的情况下，使用这个，对于单一的 GitHub 终点，紧跟*GITHUB_USERNAME* and *GITHUB_TOKEN*。

```
GITHUB_CREDENTIALS = {
    "api.github.com": {
        "username": "weblate",
        "token": "your-api-token",
    },
    "github.example.com": {
        "username": "weblate",
        "token": "another-api-token",
    },
}
```

2.16.36 GITHUB_USERNAME

GitHub 用户名，用于发送翻译更新的拉取请求。

参见:

GITHUB_CREDENTIALS, *GitHub*

2.16.37 GITHUB_TOKEN

4.3 新版功能.

GitHub 个人访问令牌，用于进行 API 调用，来发送翻译更新的拉取请求。

参见:

GITHUB_CREDENTIALS, *GitHub*, 创建个人访问令牌

2.16.38 GOOGLE_ANALYTICS_ID

谷歌分析 ID，开启使用谷歌分析对 Weblate 的监控。

2.16.39 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。

例如除了 `https://user:password@git.example.com/repo.git` 会只显示 `https://git.example.com/repo.git` 。它也试图以相似的方式清除版本控制系统（VCS）错误信息。

注解： 默认这是打开的。

2.16.40 HIDE_VERSION

4.3.1 新版功能.

从未进行身份验证的用户那里的隐藏版本信息。这样将所有文档的连接点连接到最新的版本，而不是与当前安装的版本匹配文档的版本。

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。

注解： 默认这是关闭的。

2.16.41 IP_BEHIND_REVERSE_PROXY

2.14 新版功能.

指示 Weblate 是否在反向代理后面运行。

如果设置为 `True` ，Weblate 会从:setting:‘IP_PROXY_HEADER‘定义的标头中得到 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。

注解： 默认这不是打开的。

参见：

在反向代理后面运行, *频次限制*, *IP_PROXY_HEADER*, *IP_PROXY_OFFSET*

2.16.42 IP_PROXY_HEADER

2.14 新版功能.

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:

在反向代理后面运行, *频次限制*, `SECURE_PROXY_SSL_HEADER`, *IP_BEHIND_REVERSE_PROXY*, *IP_PROXY_OFFSET*

2.16.43 IP_PROXY_OFFSET

2.14 新版功能.

指示*IP_PROXY_HEADER* 的哪部分用作客户端 IP 地址。

依赖于你的设置，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:

在反向代理后面运行, *频次限制*, `SECURE_PROXY_SSL_HEADER`, *IP_BEHIND_REVERSE_PROXY*, *IP_PROXY_HEADER*

2.16.44 LEGAL_URL

3.5 新版功能.

你的 Weblate 事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* 。

例:

```
LEGAL_URL = "https://weblate.org/terms/"
```

2.16.45 LICENSE_EXTRA

包括在许可选择中的其他许可。

> **注解:** 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",
    ),
]
```

2.16.46 LICENSE_FILTER

在 4.3 版更改: 将其设置为空值现在关闭了许可提醒。

要显示的被筛选的许可列表。在设置为空时还关闭了许可提醒。

注解: 这个过滤器使用了短许可名称。

例如:

```
LICENSE_FILTER = {"AGPL-3.0", "GPL-3.0-or-later"}
```

后面关闭了许可提醒:

```
LICENSE_FILTER = set()
```

参见:

alerts

2.16.47 LICENSE_REQUIRED

定义了是否需要组件配置 中的许可属性。

注解: 默认这是关闭的。

2.16.48 LIMIT_TRANSLATION_LENGTH_BY_SOURCE_LENGTH

给定翻译的长度是否应被限制。限制为源字符串的长度 * 10 个字符。

提示: 将其设置为 `False` 来允许更长的翻译，而不管源字符串的长度。

注解: 默认为 `True` 。

2.16.49 LOCALIZE_CDN_URL 和 LOCALIZE_CDN_PATH

这些设置配置了*JavaScript 本地化 CDN* 插件。*LOCALIZE_CDN_URL* 定义了可获得本地化 CDN 的根 URL，而*LOCALIZE_CDN_PATH* 定义了 Weblate 应该存储生成文件的路径，生成的文件在*LOCALIZE_CDN_URL* 使用。

提示： 在 hosted Weblate 中，这使用 `https://weblate-cdn.com/` 。

参见：

JavaScript 本地化 CDN

2.16.50 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，如上面的例子所示。

参见：

REQUIRE_LOGIN

2.16.51 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
)
```

2.16.52 MATOMO_SITE_ID

你想要跟踪的 Matomo （以前的 Piwik ）中的网站 ID。

注解: 这个集成不支持 Matomo Tag Manager 。

参见:

MATOMO_URL

2.16.53 MATOMO_URL

你想要跟踪的 Matomo（以前的 Piwik）安装的全 URL（包括反斜杠）。更多细节请查阅 <https://matomo.org/> 。

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

2.16.54 MT_SERVICES

在 3.0 版更改: 设置从 `MACHINE_TRANSLATION_SERVICES` 重命名为 `MT_SERVICES` ，而与其它机器翻译设置一致。

允许使用的机器翻译服务的列表。

注解: 很多服务需要像类似 API 密钥的额外配置，更多细节请查阅:ref:*machine-translation-setup*。

```
MT_SERVICES = (
    "weblate.machinery.apertium.ApertiumAPYTranslation",
    "weblate.machinery.deepl.DeepLTranslation",
    "weblate.machinery.glosbe.GlosbeTranslation",
    "weblate.machinery.google.GoogleTranslation",
    "weblate.machinery.microsoft.MicrosoftCognitiveTranslation",
    "weblate.machinery.microsoftterminology.MicrosoftTerminologyService",
    "weblate.machinery.mymemory.MyMemoryTranslation",
    "weblate.machinery.tmserver.AmagamaTranslation",
    "weblate.machinery.tmserver.TMServerTranslation",
    "weblate.machinery.yandex.YandexTranslation",
    "weblate.machinery.weblatetm.WeblateTranslation",
    "weblate.machinery.saptranslationhub.SAPTranslationHub",
    "weblate.memory.machine.WeblateMemory",
)
```

参见:

机器翻译, *自动建议*

2.16.55 MT_APERTIUM_APY

Apertium-APy 服务器的 URL，https://wiki.apertium.org/wiki/Apertium-apy

参见:

Apertium, 机器翻译, 自动建议

2.16.56 MT_AWS_ACCESS_KEY_ID

Amazon Translate 的访问密钥 ID 。

参见:

AWS, 机器翻译, 自动建议

2.16.57 MT_AWS_SECRET_ACCESS_KEY

Amazon Translate 的 API 密钥。

参见:

AWS, 机器翻译, 自动建议

2.16.58 MT_AWS_REGION

Amazon Translate 使用的区域名称。

参见:

AWS, 机器翻译, 自动建议

2.16.59 MT_BAIDU_ID

百度智云 API 的客户端 ID ，可以在 https://api.fanyi.baidu.com/api/trans/product/index 注册

参见:

Baidu API 机器翻译, 机器翻译, 自动建议

2.16.60 MT_BAIDU_SECRET

百度智云 API 的客户端密钥，可以在 https://api.fanyi.baidu.com/api/trans/product/index 注册

参见:

Baidu API 机器翻译, 机器翻译, 自动建议

2.16.61 MT_DEEPL_API_VERSION

4.1.1 新版功能.

DeepL 服务使用的 API 版本。版本限制了使用范围：

v1 意味着计算机辅助翻译工具，并且在基于用户的订阅时使用。

v2 意味着 API 应用，并且订阅是基于使用的。

此前 Weblate 被 DeepL 分类为计算机辅助翻译工具，因此应该使用 v1 API ，但现在应该使用 v2 API 。这样默认为 v2 ，在你有现有的计算机辅助翻译工具订阅，并想要 Weblate 使用它的情况下，可以将其更改为 v1 。

参见:

DeepL, 机器翻译, 自动建议

2.16.62 MT_DEEPL_KEY

DeepL API 的 API 密钥，可以在 https://www.deepl.com/pro.html 注册使用

参见:

DeepL, 机器翻译, 自动建议

2.16.63 MT_GOOGLE_KEY

谷歌翻译 API v2 的 API 密钥，可以在 https://cloud.google.com/translate/docs 上注册使用

参见:

谷歌翻译, 机器翻译, 自动建议

2.16.64 MT_GOOGLE_CREDENTIALS

Google 云控制台中得到 API v3 JSON 证明文件。请提供 OS 全路径。证明是某个项目附属的每个服务账户。更多细节请查看 https://cloud.google.com/docs/authentication/getting-started 。

2.16.65 MT_GOOGLE_PROJECT

已激活翻译服务和账单处于激活状态的 Google Cloud API v3 项目 ID，更多细节请看 https://cloud.google.com/appengine/docs/standard/nodejs/building-app/creating-project

2.16.66 MT_GOOGLE_LOCATION

API v3 Google 云应用引擎可能特定于某个位置。如果默认的 “global“ 回退无法正常工作，则需要相应地进行更改。

细节请查看 https://cloud.google.com/appengine/docs/locations

参见:

Google Translate API V3 （高级版）

2.16.67 MT_MICROSOFT_BASE_URL

在 "Base URLs" 部分 定义的基于 URL 域名的区域。

Azure Global 的默认值为 `api.cognitive.microsofttranslator.com` 。

对于 Azure China ，请使用 `api.translator.azure.cn` 。

2.16.68 MT_MICROSOFT_COGNITIVE_KEY

Microsoft Cognitive Services Translator API 客户端密匙。

参见:

微软认知服务翻译工具, *机器翻译*, *自动建议*, *认知服务 -文本翻译 API*, **'Microsoft Azure 门户 <https://portal.azure.com/>'_**

2.16.69 MT_MICROSOFT_REGION

"通过一个多服务资源进行验证" 部分中定义的地区前缀。

2.16.70 MT_MICROSOFT_ENDPOINT_URL

在 "Authenticating with an access token" 部分 定义的用于访问令牌的区域端点 URL 域名。

Azure Global 的默认值为 `api.cognitive.microsoft.com`。

对于 Azure 中国，请使用你的 Azure Portal 的端点。

2.16.71 MT_MODERNMT_KEY

ModernMT 机器翻译引擎的 API 密钥。

参见:

ModernMT `MT_MODERNMT_URL`

2.16.72 MT_MODERNMT_URL

ModernMT URL 默认值为 `https://api.modernmt.com/` 。

参见:

ModernMT `MT_MODERNMT_KEY`

2.16.73 MT_MYMEMORY_EMAIL

MyMemory 身份电子邮件地址。每天允许 1000 个请求。

参见:

MyMemory, *机器翻译*, *自动建议*, MyMemory: API technical specifications

2.16.74 MT_MYMEMORY_KEY

MyMemory 访问密钥，用于私有翻译记忆库，与`MT_MYMEMORY_USER`一起使用。

参见:

MyMemory, 机器翻译, 自动建议, MyMemory: API key generator

2.16.75 MT_MYMEMORY_USER

MyMemory 用户 ID ，用于私有翻译记忆库，与:setting:‘MT_MYMEMORY_KEY‘一起使用。

参见:

MyMemory, 机器翻译, 自动建议, MyMemory: API key generator

2.16.76 MT_NETEASE_KEY

NetEase Sight API 的 App 密匙，可以在 https://sight.youdao.com/ 注册

参见:

网易见外 API 机器翻译, 机器翻译, 自动建议

2.16.77 MT_NETEASE_SECRET

NetEase Sight API 的 App secret，可以在 https://sight.youdao.com/ 上注册

参见:

网易见外 API 机器翻译, 机器翻译, 自动建议

2.16.78 MT_TMSERVER

正在运行 TMServer 的 URL 。

参见:

tmserver, 机器翻译, 自动建议, tmserver

2.16.79 MT_YANDEX_KEY

Yandex 翻译 API 的 API 密钥，可以在 https://yandex.com/dev/translate/进行注册

参见:

Yandex 翻译, 机器翻译, 自动建议

2.16.80 MT_YOUDAO_ID

有道志云 API ID，可以注册在 https://ai.youdao.com/product-fanyi-text.s 。

参见:

有道智云 API 机器翻译, *机器翻译*, *自动建议*

2.16.81 MT_YOUDAO_SECRET

有道智云 API 密匙，可以在 https://ai.youdao.com/product-fanyi-text.s 注册。

参见:

有道智云 API 机器翻译, *机器翻译*, *自动建议*

2.16.82 MT_SAP_BASE_URL

SAP Translation Hub 服务的 API URL 。

参见:

SAP 翻译中心, *机器翻译*, *自动建议*

2.16.83 MT_SAP_SANDBOX_APIKEY

sandbox API 使用的 API 密钥

参见:

SAP 翻译中心, *机器翻译*, *自动建议*

2.16.84 MT_SAP_USERNAME

SAP 的用户名

参见:

SAP 翻译中心, *机器翻译*, *自动建议*

2.16.85 MT_SAP_PASSWORD

你的 SAP 密码

参见:

SAP 翻译中心, *机器翻译*, *自动建议*

2.16.86 MT_SAP_USE_MT

除了术语数据库，是否还使用机器翻译服务。可能值：`True` 或 `False`

参见:

SAP 翻译中心, *机器翻译*, *自动建议*

2.16.87 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。

2.16.88 PAGURE_CREDENTIALS

4.3.2 新版功能.

Pagure 服务器凭据列表。

提示: 如果您希望 Weblate 与更多端点互动，请使用此功能，对于单个 Pagure 端点，保持设置项:setting:‘PAGURE_USERNAME‘和:setting:‘PAGURE_TOKEN‘不变即可。

```
PAGURE_CREDENTIALS = {
    "pagure.io": {
        "username": "weblate",
        "token": "your-api-token",
    },
    "pagure.example.com": {
        "username": "weblate",
        "token": "another-api-token",
    },
}
```

2.16.89 PAGURE_USERNAME

4.3.2 新版功能.

Pagure 用户名，用于发送翻译更新的合并请求。

参见:

PAGURE_CREDENTIALS, *Pagure*

2.16.90 PAGURE_TOKEN

4.3.2 新版功能.

Pagure 个人访问令牌，用于进行翻译更新的 API 调用。

参见:

PAGURE_CREDENTIALS, *Pagure*, Pagure API

2.16.91 RATELIMIT_ATTEMPTS

3.2 新版功能.

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*

2.16.92 RATELIMIT_WINDOW

3.2 新版功能.

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*

2.16.93 RATELIMIT_LOCKOUT

3.2 新版功能.

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*

2.16.94 REGISTRATION_ALLOW_BACKENDS

4.1 新版功能.

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。

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

2.16.95 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格。

2.16.96 REGISTRATION_EMAIL_MATCH

2.17 新版功能.

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$"
```

2.16.97 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 Social Auth 的身份验证（可以使用*REGISTRATION_ALLOW_BACKENDS* 为适当的后端建立白名单）。

注解： 如果使用第三方身份验证方法，如*LDAP 身份验证* ，只是隐藏注册表格，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*

2.16.98 REPOSITORY_ALERT_THRESHOLD

4.0.2 新版功能.

触发仓库过期警告的阈值，或者包含了太多更改的阈值。默认为 25 。

参见：

alerts

2.16.99 REQUIRE_LOGIN

4.1 新版功能.

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。

注解: 这实现在 sample-configuration‘中。对于 Docker，使用:envvar:‘WEBLATE_REQUIRE_LOGIN。

2.16.100 SENTRY_DSN

3.9 新版功能.

Sentry DSN 用于收集错误报告 。

参见:

Sentry 的 Django 集成

2.16.101 SESSION_COOKIE_AGE_AUTHENTICATED

4.3 新版功能.

对身份验证的用户设置会话过期。这补充了用于没有身份验证用户的 `SESSION_COOKIE_AGE` 。

参见:

`SESSION_COOKIE_AGE`

2.16.102 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。

2.16.103 SITE_DOMAIN

配置网站域名。这在很多领域产生正确的绝对链接是必要的（例如激活电子邮箱、通知或 RSS 推送）。

在 Weblate 运行在非标准端口时，这里同样要包括它。

例子:

```
# Production site with domain name
SITE_DOMAIN = "weblate.example.com"

# Local development with IP address and port
SITE_DOMAIN = "127.0.0.1:8000"
```

注解: 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* and for changing URL, use *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*

2.16.104 SITE_TITLE

用于网站和发送电子邮件的网站名称。

2.16.105 SPECIAL_CHARS

屏幕键盘中包括的另外的字符，*Visual keyboard* 。

默认值为:

```
SPECIAL_CHARS = ("\t", "\n", "…")
```

2.16.106 SINGLE_PROJECT

3.8 新版功能.

将用户直接重定向到项目或组件，而不是显示控制面板。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 版更改: 设置只接受项目标识串，来强制显示那个单一项目。

例:

```
SINGLE_PROJECT = "test"
```

2.16.107 STATUS_URL

Weblate 事例报告其状态的 URL 。

2.16.108 SUGGESTION_CLEANUP_DAYS

3.2.1 新版功能.

给定天数后自动删除建议。默认为 `None` ，意味着不删除。

2.16.109 UPDATE_LANGUAGES

4.3.2 新版功能.

控制在运行数据库迁移时是否应该更新语言数据库并在默认情况下启用。这个设置对:djadmin: *setuplang* 的调用没有影响。

参见:

内置语言定义

2.16.110 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）。

注解: 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。

例:

```
URL_PREFIX = "/translations"
```

注解: 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。

2.16.111 VCS_BACKENDS

可用的版本控制系统（VCS ）后端的配置。

注解: 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

2.16.112 VCS_CLONE_DEPTH

3.10.2 新版功能.

配置 Weblate 应该对仓库进行深度为多少的克隆。

注解: 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
```

2.16.113 WEBLATE_ADDONS

附加组件可供使用的附加组件列表。为了使用，必须对给定的翻译组件允许它们。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 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.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.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",
    # Addon you want to include
    "weblate.addons.example.ExampleAddon",
)
```

注解： 从列表中删除插件并不将它从组件中卸载。Weblate 在这种情况下会崩溃。请在从这个列表中删除前从所有组件中卸载插件。

参见：

附加组件, *DEFAULT_ADDONS*

2.16.114 WEBLATE_EXPORTERS

4.2 新版功能.

可用的导出程序列表，提供下载各种文件格式的翻译或词汇表。

参见：

支持的文件格式

2.16.115 WEBLATE_FORMATS

3.0 新版功能.

可供使用的文件格式列表。

注解: 默认列表已经具有了常见格式。

参见:

支持的文件格式

2.16.116 WEBLATE_GPG_IDENTITY

3.1 新版功能.

Weblate 使用的身份，用于登录 Git Commits ，例如：

```
WEBLATE_GPG_IDENTITY = "Weblate <weblate@example.com>"
```

搜索 Weblate GPG 钥匙链（`home/.gnupg` ，在*DATA_DIR* 之下）。如果没有找到，则产生密钥，更多细节请查询*签署 GnuPG 的 Git 承诺* 。

参见:

签署 GnuPG 的 Git 承诺

2.17 配置的例子

后面的例子作为 `weblate/settings_example.py` 与 Weblate 一起上市：

```
#
# Copyright © 2012 - 2021 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/>.
#


import os
import platform
from logging.handlers import SysLogHandler

#
# Django settings for Weblate project.
#
```

(下页继续)

(续上页)

```
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:
            # "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,
        },
    }
}

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Data directory
DATA_DIR = os.path.join(BASE_DIR, "data")

# 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", "العربية"),
    ("az", "Azərbaycan"),
    ("be", "Беларуская"),
    ("be@latin", "Biełaruskaja"),
    ("bg", "Български"),
```

(下页继续)

(续上页)

```
    ("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"),
    ("ru", "Русский"),
    ("sk", "Slovenčina"),
    ("sl", "Slovenščina"),
    ("sq", "Shqip"),
    ("sr", "Српски"),
    ("sv", "Svenska"),
    ("tr", "Türkçe"),
    ("uk", "Українська"),
    ("zh-hans", "简体字"),
    ("zh-hant", "正體字"),
)

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

# 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/"
```

(下页继续)

(续上页)

```
# 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(DATA_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",
    "compressor.finders.CompressorFinder",
)

# Make this unique, and don't share it with anybody.
# You can generate it using weblate/examples/generate-secret-key
SECRET_KEY = ""

_TEMPLATE_LOADERS = [
    "django.template.loaders.filesystem.Loader",
    "django.template.loaders.app_directories.Loader",
]
if not DEBUG:
    _TEMPLATE_LOADERS = [("django.template.loaders.cached.Loader", _TEMPLATE_
↪LOADERS)]
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",
            ],
            "loaders": _TEMPLATE_LOADERS,
        },
    }
]


# GitHub username for sending pull requests.
# Please see the documentation for more details.
GITHUB_USERNAME = None
GITHUB_TOKEN = None

# GitLab username for sending merge requests.
# Please see the documentation for more details.
```

(下页继续)

(续上页)

```
GITLAB_USERNAME = None
GITLAB_TOKEN = None

# Authentication configuration
AUTHENTICATION_BACKENDS = (
    "social_core.backends.email.EmailAuth",
    # "social_core.backends.google.GoogleOAuth2",
    # "social_core.backends.github.GithubOAuth2",
    # "social_core.backends.bitbucket.BitbucketOAuth",
    # "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_BITBUCKET_KEY = ""
SOCIAL_AUTH_BITBUCKET_SECRET = ""
SOCIAL_AUTH_BITBUCKET_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",
```

(下页继续)

(续上页)

```
)
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

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")
    #     }
    # },
]

# Allow new user registrations
REGISTRATION_OPEN = True

# Shortcut for login required setting
REQUIRE_LOGIN = False

# Middleware
MIDDLEWARE = [
    "weblate.middleware.RedirectMiddleware",
```

(下页继续)

(续上页)

```
    "weblate.middleware.ProxyMiddleware",
    "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",
    "weblate.api.middleware.ThrottlingMiddleware",
    "weblate.middleware.SecurityMiddleware",
]

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",
    # 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",
    "compressor",
    "rest_framework",
    "rest_framework.authtoken",
    "django_filters",
]

# Custom exception reporter to include some details
DEFAULT_EXCEPTION_REPORTER_FILTER = "weblate.trans.debug.
↪WeblateExceptionReporterFilter"
```

(下页继续)

(续上页)

```
# Default logging of Weblate messages
# - to syslog in production (if available)
# - otherwise to console
# - you can also choose "logfile" to log into separate file
#   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

if DEBUG or not HAVE_SYSLOG:
    DEFAULT_LOG = "console"
else:
    DEFAULT_LOG = "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": {
```

(下页继续)

(续上页)

```
            "level": "DEBUG",
            "class": "logging.handlers.SysLogHandler",
            "formatter": "syslog",
            "address": "/dev/log",
            "facility": SysLogHandler.LOG_LOCAL2,
        },
        # Logging to a file
        # "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.microsoft.MicrosoftCognitiveTranslation",
    #     "weblate.machinery.microsoftterminology.MicrosoftTerminologyService",
    #     "weblate.machinery.modernmt.ModernMTTranslation",
    #     "weblate.machinery.mymemory.MyMemoryTranslation",
    #     "weblate.machinery.netease.NeteaseSightTranslation",
```

(下页继续)

(续上页)

```
    #     "weblate.machinery.tmserver.AmagamaTranslation",
    #     "weblate.machinery.tmserver.TMServerTranslation",
    #     "weblate.machinery.yandex.YandexTranslation",
    #     "weblate.machinery.saptranslationhub.SAPTranslationHub",
    #     "weblate.machinery.youdao.YoudaoTranslation",
    "weblate.machinery.weblatetm.WeblateTranslation",
    "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

# 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
```

(下页继续)

(续上页)

```
MT_SAP_USERNAME = None
MT_SAP_PASSWORD = None
MT_SAP_USE_MT = True

# Title of site to use
SITE_TITLE = "Weblate"

# Site domain
SITE_DOMAIN = ""

# Whether site uses https
ENABLE_HTTPS = False

# 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
# Sent referrer 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
# Increase allowed upload size
DATA_UPLOAD_MAX_MEMORY_SIZE = 50000000

# 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

# 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/"
```

(下页继续)

(续上页)

```
# URL of logout
LOGOUT_URL = f"{URL_PREFIX}/accounts/logout/"

# Default location for login
LOGIN_REDIRECT_URL = f"{URL_PREFIX}/"

# 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_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.CSharpFormatCheck",
#     "weblate.checks.format.JavaFormatCheck",
#     "weblate.checks.format.JavaMessageFormatCheck",
```

(下页继续)

(续上页)

```
#     "weblate.checks.format.PercentPlaceholdersCheck",
#     "weblate.checks.format.VueFormattingCheck",
#     "weblate.checks.format.I18NextInterpolationCheck",
#     "weblate.checks.format.ESTemplateLiteralsCheck",
#     "weblate.checks.angularjs.AngularJSInterpolationCheck",
#     "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",
# )

# 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.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.properties.PropertiesSortAddon",
#     "weblate.addons.git.GitSquashAddon",
```

(下页继续)

(续上页)

```
#     "weblate.addons.removal.RemoveComments",
#     "weblate.addons.removal.RemoveSuggestions",
#     "weblate.addons.resx.ResxUpdateAddon",
#     "weblate.addons.yaml.YAMLCustomizeAddon",
#     "weblate.addons.cdn.CDNJSAddon",
#     "weblate.addons.autotranslate.AutoTranslateAddon",
# )

# 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",
    },
    "avatar": {
        "BACKEND": "django.core.cache.backends.filebased.FileBasedCache",
        "LOCATION": os.path.join(DATA_DIR, "avatar-cache"),
        "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"
        if REQUIRE_LOGIN
        else "rest_framework.permissions.IsAuthenticatedOrReadOnly"
    ],
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.TokenAuthentication",
        "weblate.api.authentication.BearerAuthentication",
```

(下页继续)

(续上页)

```
        "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": ("rest_framework.pagination.PageNumberPagination"),
    "PAGE_SIZE": 20,
    "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
# )

# 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://"
# 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
```

(下页继续)

(续上页)

```
CELERY_WORKER_MAX_MEMORY_PER_CHILD = 200000
CELERY_BEAT_SCHEDULE_FILENAME = os.path.join(DATA_DIR, "celery", "beat-schedule")
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"},
}

# 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
AKISMET_API_KEY = None
```

2.18 管理命令

注解: 不同用户下运行管理命令而不是一人运行您的 web 服务器，可以导致文件得到错误的权限，更多细节请查看文件系统权限 。

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，以用依赖于您如何安装 Weblate 。

如果使用 Virtualenv 来运行 Weblate ，那么您可以或者为 **weblate** 指定全路径，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 或 pip3 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 映像，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*, *从源文件安装*

2.18.2 add_suggestions

weblate add_suggestions <project> <component> <language> <file>

2.5 新版功能.

从文件导入翻译，来作为给定翻译的建议来使用。它跳过了复制翻译的步骤；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）。

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>

2.5 新版功能.

根据其他组件翻译进行自动翻译。

--source PROJECT/COMPONENT
指定组件，用作可获得翻译的来源。如果不指定，将使用项目中的所有组件。

--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组件作为机器翻译。

--threshold THRESHOLD
用于机器翻译的相似性阈值，默认为 80 。

例：

```
weblate auto_translate --user nijel --inconsistent --source weblate/application␣
↪weblate website cs
```

参见：

自动化翻译

2.18.4 celery_queues

weblate celery_queues

3.7 新版功能.

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组件要更新（例如 `weblate/application` ），或者使用 `--all` 来更新所有现有组件。

2.18.6 commitgit

weblate commitgit <project|project/component>

将任何可能待定的更改提交给后端 Git 仓库。

您可以确定或者哪个项目或组件要更新（例如 `weblate/application` ），或者使用 `--all` 来更新所有现有组件。

2.18.7 commit_pending

weblate commit_pending <project|project/component>

提交早于给定时间段的待定更改。

您可以确定或者哪个项目或组件要更新（例如 `weblate/application` ），或者使用 `--all` 来更新所有现有组件。

--age HOURS
时间段以小时为单位。如果不指定，则使用在*组件配置* 中配置的值。

注解： 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*组件配置* 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 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 版更改: 添加参数 `--username` 、`--email` 、`--name` 和 `--update` 。

2.18.10 dump_memory

weblate dump_memory

2.20 新版功能.

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*

2.18.11 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 事例是会很方便。

2.18.12 import_demo

weblate import_demo

4.1 新版功能.

根据 <https://github.com/WeblateOrg/demo> 使用组件来新建演示项目。

这在开发 Weblate 时会有用。

2.18.13 import_json

weblate import_json <json-file>

2.7 新版功能.

根据 JSON 数据批量导入组件。

导入的 JSON 文件结构非常符合组件对象（请参见*GET /api/components/(string:project)/(string:component)/*）。您必须包括 `name` 和 `filemask` 字段。

--project PROJECT
指定从哪里导入组件。

--main-component COMPONENT
对所有的使用来自这个组件的给定版本控制系统（VCS）仓库。

--ignore
跳过（已经）导入的组件。

--update
更新（已经）导入的组件。

在 2.9 版更改: 那里的参数 `--ignore` 和 `--update` 用于处理已经导入的组件。

JSON 文件的例子:

```
[
  {
    "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

2.18.14 import_memory

weblate import_memory <file>

2.20 新版功能.

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。

在您的 TMX 文件地区恰好与您在 Weblate 使用地区不同的情况下，这会有用。

参见:

翻译记忆库, *Weblate 翻译记忆库概要*

2.18.15 import_project

weblate import_project <project> <gitrepo> <branch> <filemask>

在 3.0 版更改: import_project 命令现在基于组件发现 插件，导致一些行为的更改，并接受一些参数。

根据 filemask ，将组件批量导入项目。

<project> 将已存在的项目命名，组件将导入其中。

<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 组件导入另外的翻译组件，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。

简单的匹配对组件名称使用 `**` ，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，导入与现有的组件匹配，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 模板语法来定制组件的名称。

例如: `Documentation: {{ component }}`

--base-file-template TEMPLATE
为单语言翻译定制译文模板文件。

例如: `{{ component }}/res/values/string.xml`

--new-base-template TEMPLATE
为另外新的翻译定制译文模板文件。

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: 组件配置 ）。它必须是合法的正则表达式。

--main-component
您可以指定选择哪个组件作为主要的一个——即真正包含版本控制系统（VCS ）仓库的那个。

--license NAME
指定整体、项目或组件翻译的许可。

--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。

为了给出一些例子，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译文模板文件，并且指定所有组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
```

更复杂的例子是关于解析文件名而从文件名中得到正确的组件和语言，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:

更多具体的例子可以在 starting 章节找到，另外您会想要使用*import_json*。

2.18.16 importuserdata

weblate importuserdata <file.json>

从*dumpuserdata* 建立的文件中导入用户数据

2.18.17 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邮箱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
```

2.18.18 install_addon

3.2 新版功能.

weblate install_addon --addon ADDON <project|project/component>

将插件安装到一组组件中。

--addon ADDON
要安装的插件名称。例如 `weblate.gettext.customize`。

--configuration CONFIG
以 JSON 编码的插件配置。

--update
更新现有的插件配置。

可以或者定义将插件安装到哪个项目或组件中（例如 `weblate/application`），或者使用 `--all` 来包括所有现有的组件。

为所有的组件安装*自定义 gettext 输出*：

```
weblate install_addon --addon weblate.gettext.customize --config '{"width": -1}' --
↪update --all
```

参见:

附加组件

2.18.19 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>.

2.18.20 list_translators

weblate list_translators <project|project/component>

对给定的项目列出为这种语言做出贡献的译者:

```
[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组件（例如 `weblate/application`），或者使用 `--all` 从所有现存的组件中列出翻译者。

2.18.21 list_versions

weblate list_versions

列出所有 Weblate 依赖及其版本。

2.18.22 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 ）仓库完成一些更新的情况下）。

--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组件要更新（例如 `weblate/application` ），或者使用 `--all` 来更新所有现有组件。

注解: 极少调用这个，Weblate 将对每一次版本控制系统（VCS ）更新自动加载更改的文件。在您手动更改下层 Weblate 版本控制系统（VCS ）仓库的情况下，或在更新后的一些特殊情况下才需要这个。

2.18.23 lock_translation

weblate lock_translation <project|project/component>

防止进一步翻译组件。

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组件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有组件。

参见：

unlock_translation

2.18.24 move_language

weblate move_language source target

3.0 新版功能.

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。

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。

2.18.25 pushgit

weblate pushgit <project|project/component>

将执行的更改推送到上游版本控制系统（VCS）仓库。

--force-commit
在推送前，强制执行任何待定的更改。

您可以确定或者哪个项目或组件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有组件。

注解： 如果开启了组件配置 中的:ref:*component-push_on_commit*，Weblate 会自动推送更改，这是默认的。

2.18.26 unlock_translation

weblate unlock_translation <project|project/component>

将给定的组件解锁，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组件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有组件。

参见：

lock_translation

2.18.27 setupgroups

weblate setupgroups

配置默认的组，并可选地将所有用户指定到那个默认组中。

--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:

访问控制

2.18.28 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）。

2.18.29 updatechecks

weblate updatechecks <project|project/component>

对所有字符串更新所有检查。

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组件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有组件。

2.18.30 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组件要更新（例如 `weblate/application`），或者使用 `--all` 来更新所有现有组件。

注解: 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 版更改: 在此前的发布版本中，这个特性被称为白板消息。

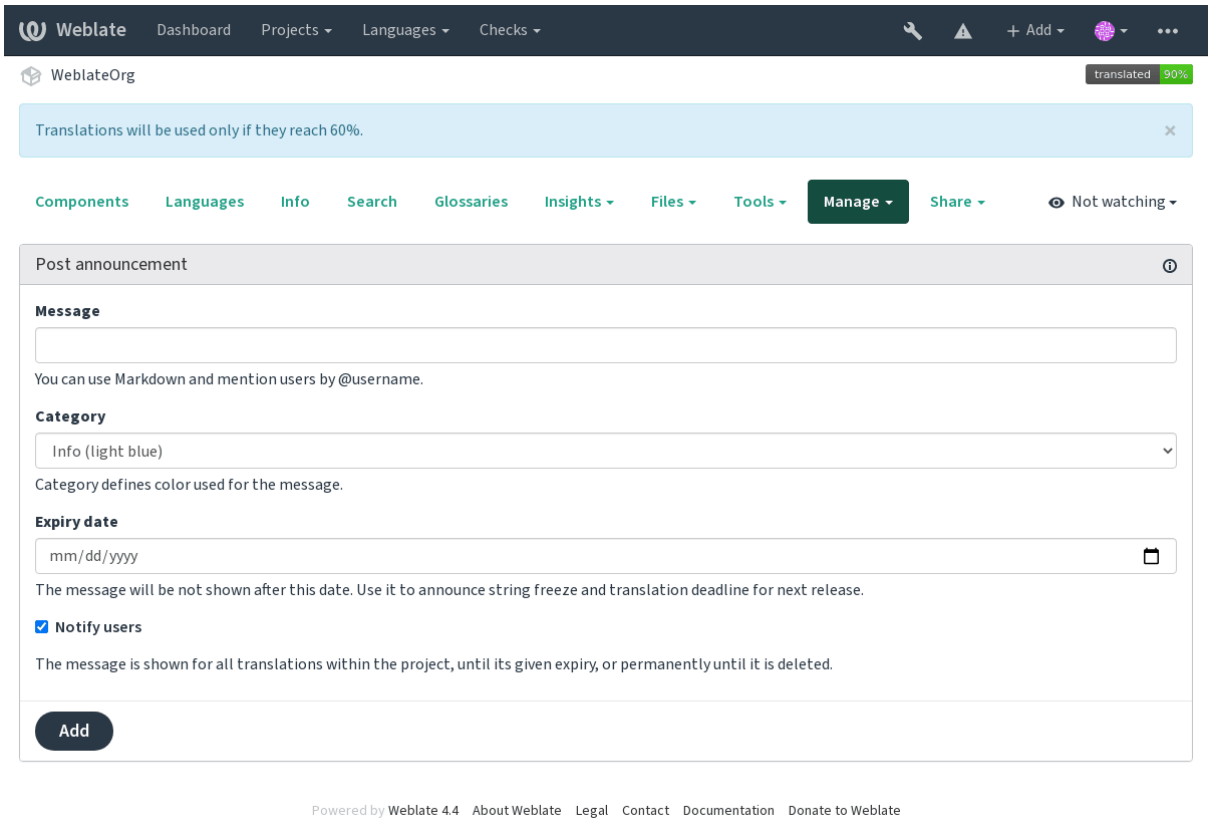
通过张贴网站范围的，每个项目、组件或语言的公告，向翻译者提供信息。

宣布翻译的目的、截止期限、状态或特定目标。

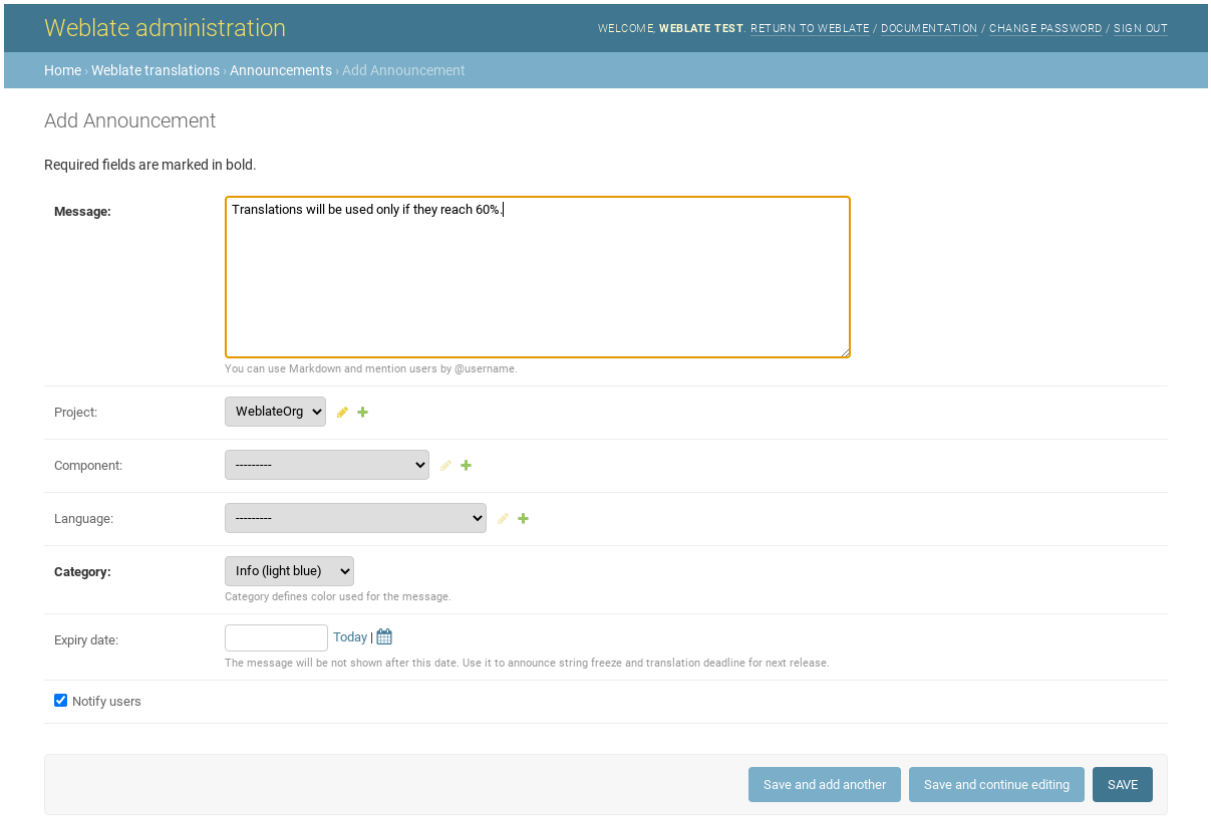
用户可以接收到关注项目公告的通知（除非用户选择关闭）。

这会用于从发布网站的目的到指定翻译的目标的各种事情。

可以使用 *Post announcement* ，在 *Manage* 菜单的每一届张贴公告：



还可以使用管理界面添加：



然后根据特定的语境显示公告：

没有特定的语境

显示在控制面版上（登录页面）。

特定项目

项目内显示，包括其所有的组件和翻译。

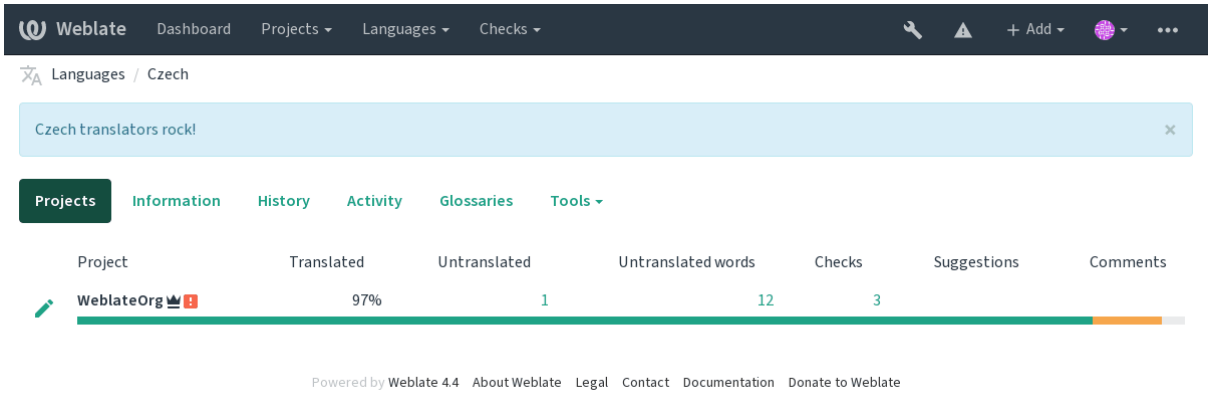
特定组件

对于给定的组件机器翻译来显示。

特定语言

显示语言的全景和该语言的全部翻译。

这就是语言全景页面上的样子：



2.20 组件列表

指定多个列表的组件，出现在用户控制面板上作为选项，从用用户可以选择一个作为默认视图。请参见控制面板 来了解更多信息。

在 2.20 版更改: 控制面板上出现的每个组件列表都会显示状态。

可以在管理界面的 *Component lists* 部分指定组件列表的名称和内容。每种组件列表必须名称来显示给用户，并具有标识串将其显示在 URL 中。

在 2.13 版更改: 从管理界面检查匿名用户的控制面板设置，修改掉控制面本显示给未授权用户的内容。

2.20.1 自动组件列表

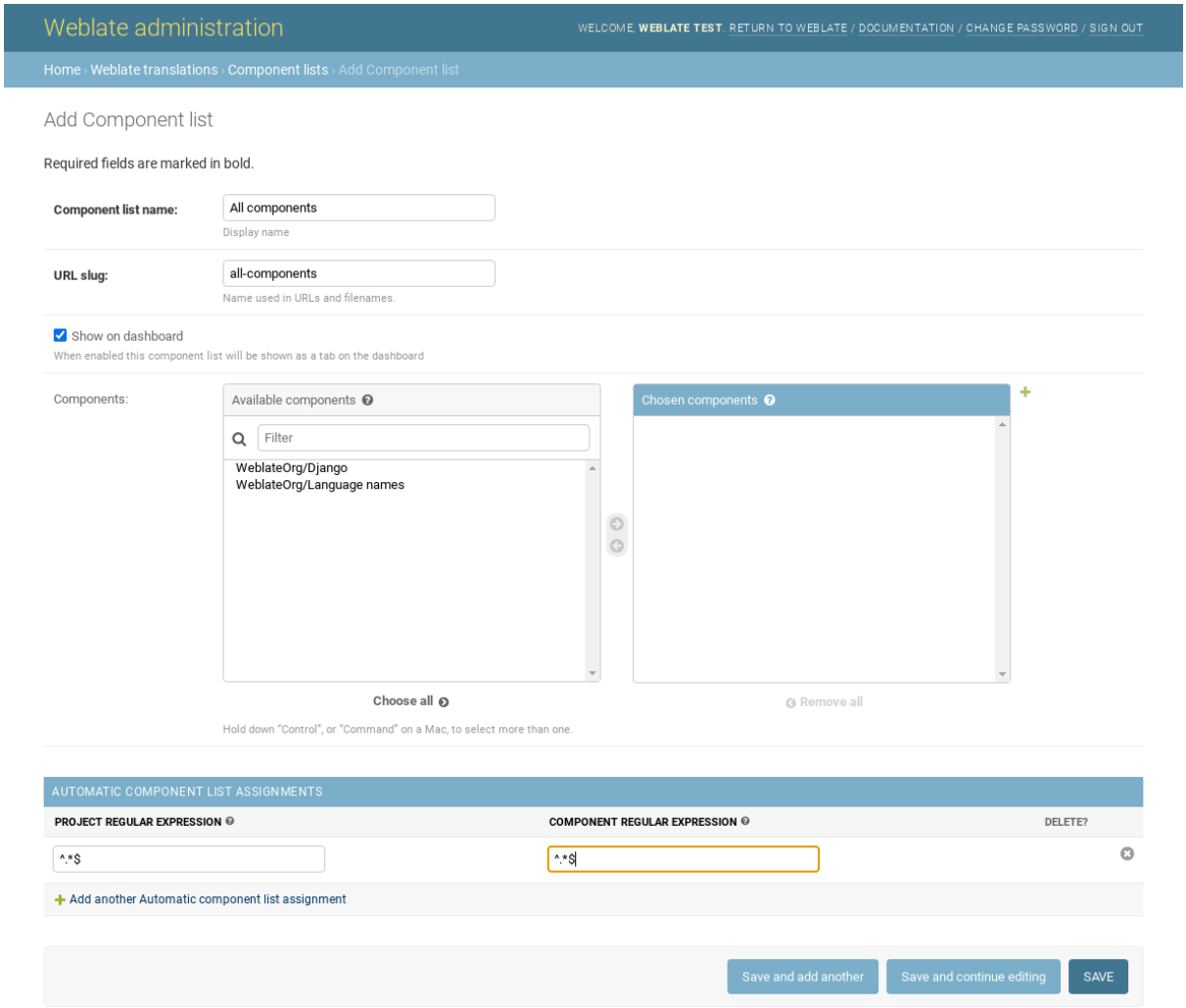
2.13 新版功能.

通过建立 *Automatic component list assignment* 规则，根据其标识串自动将组件添加到列表中。

- 对于维护大型安装的逐渐列表有用，或者如果你希望在 Weblate 安装中有一个包含所有组件的组件列表。

提示: 制作组件列表，包含自己的 Weblate 安装时的所有组件。

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

2.10 新版功能.

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*组件配置* 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/master/`。

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/master/'
```

对存储库的受限制访问（使用*项目访问控制* 或在*REQUIRE_LOGIN* 处于启用状态时）需要一个 API 令牌，获取地点在你的*用户个人资料*:

```
git clone 'https://user:KEY@example.org/git/weblate/master/'
```

2.21.2 账单

2.4 新版功能.

这用在 Hosted Weblate 上来确定付费套餐，跟踪收据和使用限制。

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。允许了账单的用户将在他们的*用户个人资料* 中得到新的 *Billing* 标签。

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组件（请参见*添加翻译项目和组件*）。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/组件），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

2.15 新版功能.

这用在 Hosted Weblate 上，来提供所需的法律文档。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

注解: 可以在这个 Git 仓库 <https://github.com/WeblateOrg/wllegal/tree/master/wllegal/templates/legal/documents> 中获取 Hosted Weblate 的法律文档。

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档，与您的服务匹配。

用法

安装并编辑后，法律文档显示在 Weblate UI 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件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

参见:

头像缓存, *AVATAR_URL_PREFIX*, *ENABLE_AVATARS*

2.21.5 针对垃圾邮件的保护

您可以通过使用 akismet.com service 来取消对用户的授权，针对建议的垃圾邮件进行保护。

1. 安装 *akismet* Python 模块
2. 配置 Akismet API 密钥。

注解： 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：*在反向代理后面运行* 。

参见：

在反向代理后面运行, *AKISMET_API_KEY*

2.21.6 签署 GnuPG 的 Git 承诺

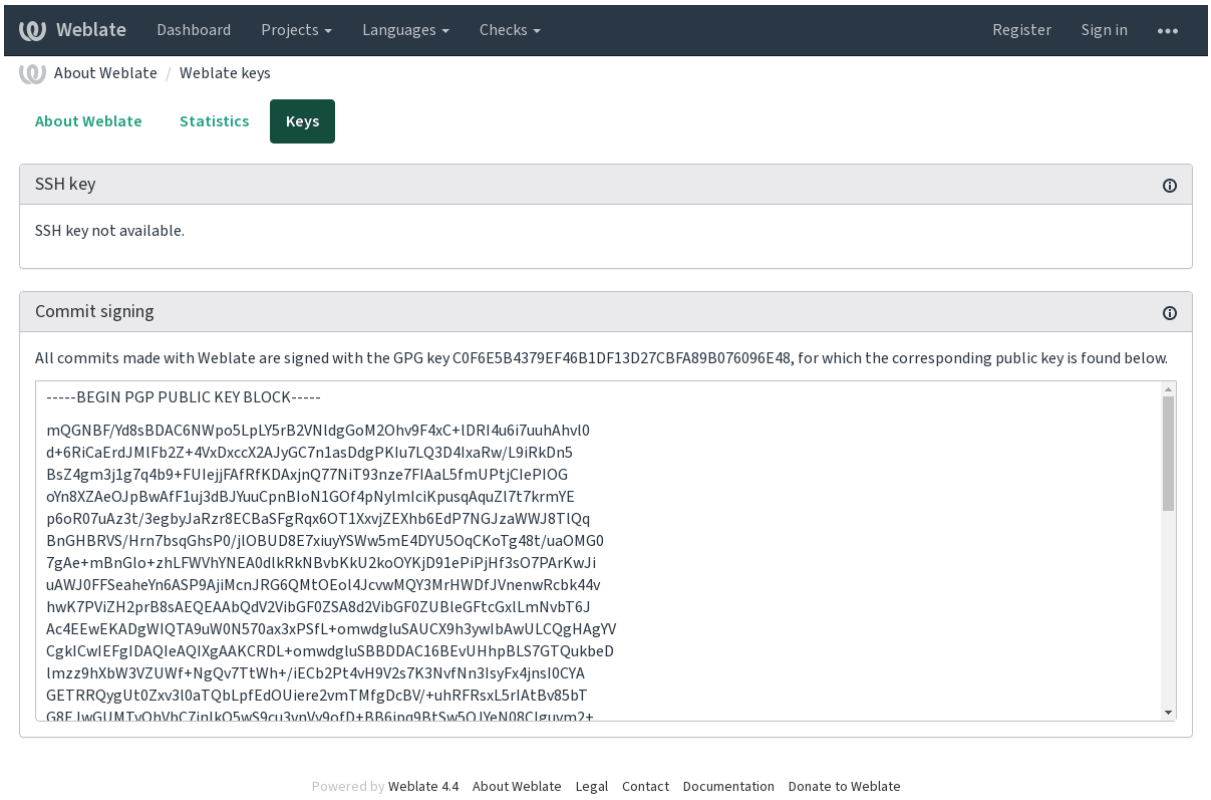
3.1 新版功能.

所有的承诺可以由 Weblate 时间的 GnuPG 密钥签署。

1. Turn on *WEBLATE_GPG_IDENTITY*. (Weblate will generate a GnuPG key when needed and will use it to sign all translation commits.)

这个特性需要安装 GnuPG 2.1 或更新版。

您可以在*DATA_DIR* 中找到密钥，而公钥显示在“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 版更改: 频次限制先择接受更细粒度的配置。

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	5	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
添加到词汇表	GLOSSARY	30	60	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。

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, *在反向代理后面运行*, *API 频次限制*

2.22 定制 Weblate

使用 Django 和 Python 扩展与定制。将你的更改贡献给上游，使每人都能够受益。这降低了你的维护成本；Weblate 中的代码对更改内部界面或重构编码时的情况。

> **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 做贡献

2.22.1 建立 Python 模块

如果不熟悉 Python ，你可以查看 Python For Beginners ，它解释了其基本内容并指向了高级教程。

为了写出一些定制 Python 代码（被称为模块），需要一个地方进行存储，或者在系统路径（通常像 /usr/lib/python3.7/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。

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",
```

（下页继续）

(续上页)

```
    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` ）。

你的模块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 的例子，它涵盖了下面描述的所有题目。

2.22.2 更改 Logo

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` 导航条中显示的 Logo 。

   `logo-*.png` 根据屏幕分辨率和 web 浏览器的 Web 图标。

   `favicon.ico` 传统浏览器使用的 Web 图标。

   `weblate-*.png` 机器人或匿名用户使用的头像。一些 Web 浏览器使用这些作为快捷图标。

   `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:

Managing 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、插件和自动修复

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：

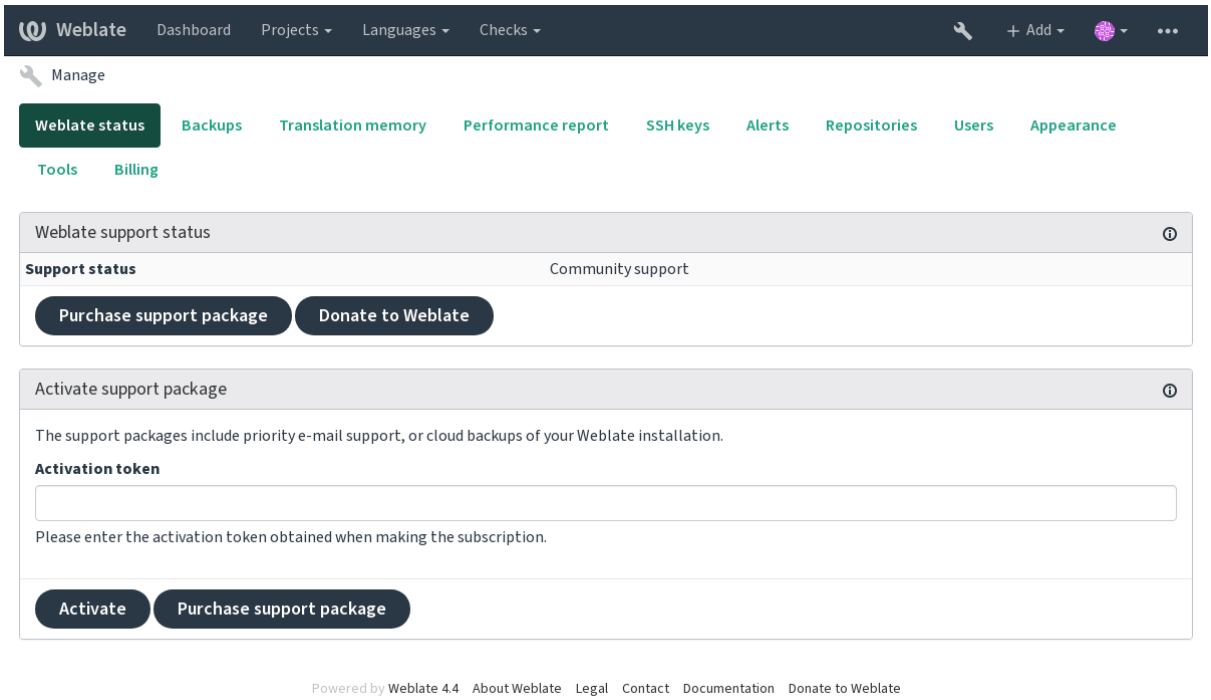
```
# Checks
CHECK_LIST += ("weblate_customization.checks.FooCheck",)

# Autofixes
AUTOFIX_LIST += ("weblate_customization.autofix.FooFixer",)

# Addons
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:file:'/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 Weblate 的基本视图：

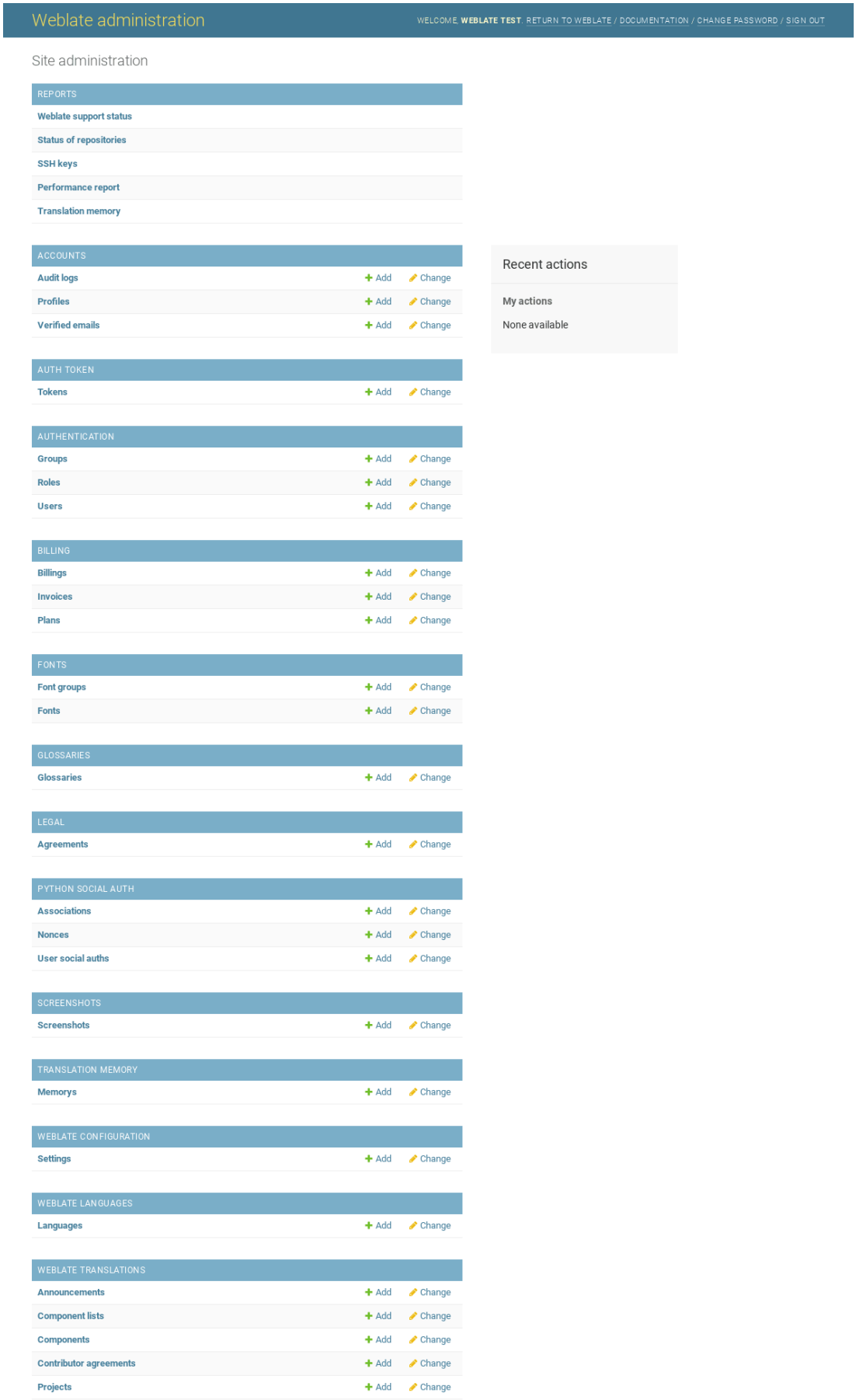
- 支持状态，请参见从 *Weblate* 获得支持
- 备份，请参见备份和移动 *Weblate*
- 共享的翻译记忆库，见翻译记忆库
- 性能报告，来复查 Weblate 的健康状况和 Celery 队列的长度
- SSH 密钥管理，见*SSH* 仓库

- 所有组件的警报概述，见 alerts

2.23.1 Django 管理界面

警告： 将来会删除，因为其应用困难——多数特性可以直接在 Weblate 中管理。

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 logs + Add Change
Profiles + Add Change
Verified emails + Add Change
AUTH TOKEN
Tokens + Add Change
AUTHENTICATION
Groups + Add Change
Roles + Add Change
Users + Add Change
BILLING
Billings + Add Change
Invoices + Add Change
Plans + Add Change
FONTS
Font groups + Add Change
Fonts + Add Change
GLOSSARIES
Glossaries + Add Change
LEGAL
Agreements + Add Change
PYTHON SOCIAL AUTH
Associations + Add Change
Nonces + Add Change
User social auths + Add Change
SCREENSHOTS
Screenshots + Add Change
TRANSLATION MEMORY
Memorys + 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

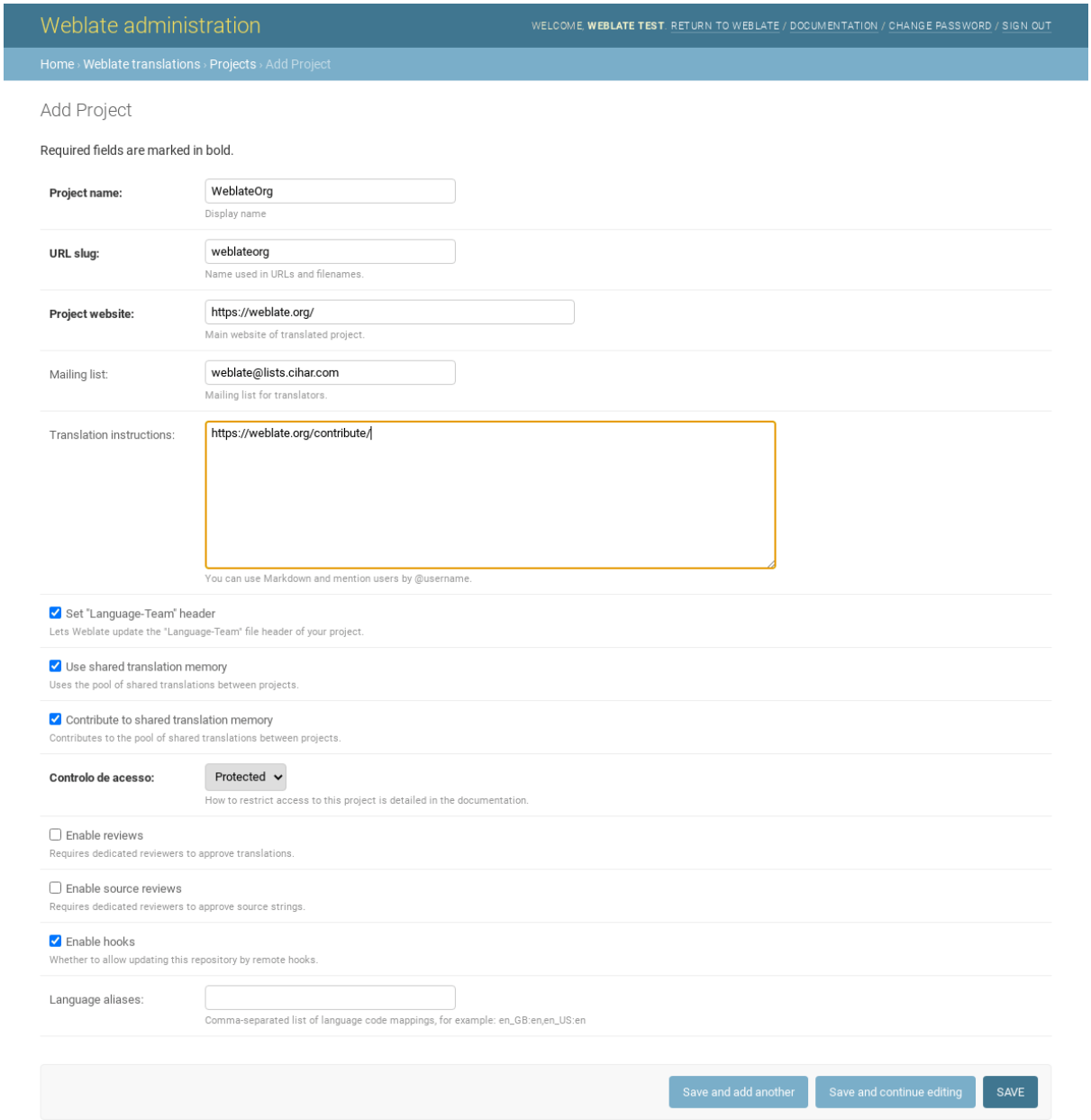
在 *Reports* 部分，可以检查网站的状态，为生产设置 进行调整，或者管理用于访问的 SSH 密钥*Accessing repositories* 。

管理任意部分下的数据库对象。最有趣的也许是 *Weblate translations* ，你可以在这里管理可翻译的项目，请参见项目配置 和组件配置 。

Weblate languages 保持语言定义，在语言定义 中进一步解释。

添加项目

添加项目作为所有组件的容器。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：



参见:

项目配置

双语言组件

一旦添加了一个项目，就可以添加翻译组件了。（关于各自参数的信息，请参见组件配置）：

Weblate administration

WELCOME, **WEBLATE TEST**. RETURN TO WEBLATE / DOCUMENTATION / CHANGE PASSWORD / SIGN OUT

Home › Weblate translations › Components › Add Component

Add Component

IMPORT SPEED DOCUMENTATION

Required fields are marked in bold.

Component name: Language names
Display name

URL slug: language-names
Name used in URLs and filenames.

Project: WeblateOrg

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 push URL:
URL of a push repository, pushing is turned off if empty.

Repository browser: https://github.com/WeblateOrg/demo/blob/{{branch}}/{{filename}}#L{
Link to repository browser, use {{branch}} for branch, {{filename}} and {{line}} as filename and line placeholders.

Exported repository URL:
URL of repository where users can fetch changes from Weblate

Source string bug reporting address:
Email address for reports on errors in source strings. Leave empty for no e-mails.

Repository branch:
Repository branch to translate

Push branch:
Branch for pushing changes, leave empty to use repository branch

Filemask: weblate/langdata/locale/*/LC_MESSAGES/djg
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:
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.

Template for new translations: weblate/langdata/locale/django.pot
Filename of file used for creating new translations. For gettext choose .pot file.

File format: gettext PO file

☐ Locked
Locked component will not get any translation updates.

☑ Allow translation propagation
Whether translation updates in other components will cause automatic translation in this one

☑ Turn on suggestions
Whether to allow translation suggestions at all.

☐ Suggestion voting
Whether users can vote for suggestions.

Autoaccept suggestions: 0
Automatically accept suggestions with this number of votes, use 0 to turn it off.

Translation flags:
Additional comma-separated flags to influence quality checks. Possible values can be found in the documentation.

Enforced checks: []
List of checks which can not be ignored.

Translation license: GNU General Public License v3.0 or later

Contributor agreement:
User agreement which needs to be approved before a user can translate this component.

Adding new translation: Create new language file
How to handle requests for creating new translations.

Language code style: Default based on the file format
Customize language code used to generate the filename for translations created by Weblate.

Merge style: Rebase
Define whether Weblate should merge the upstream repository or rebase changes onto it.

Commit message when translating:
Translated using Weblate ({{ language_name }})

Currently translated at {{ stats.translated_percent }}% ({{ stats.translated }} of {{ stats.all }} strings)

Translation: {{ project_name }}/{{ component_name }}
Translate-URL: {{ url }}

You can use template language for various info, please consult the documentation for more details.

Commit message when adding translation:
Added translation using Weblate ({{ language_name }})

You can use template language for various info, please consult the documentation for more details.

Commit message when removing translation:
Deleted translation using Weblate ({{ language_name }})

You can use template language for various info, please consult the documentation for more details.

Commit message when merging translation:
Merge branch '{{ component_remote_branch }}' into Weblate.

You can use template language for various info, please consult the documentation for more details.

Commit message when addon makes a change:
Update translation files

Updated by "{{ addon_name }}" hook in Weblate.

Translation: {{ project_name }}/{{ component_name }}
Translate-URL: {{ url }}

You can use template language for various info, please consult the documentation for more details.

Committer name: Weblate

Committer e-mail: noreply@weblate.org

☑ Push on commit
Whether the repository should be pushed upstream on every commit.

Age of changes to commit: 24
Time in hours after which any pending changes will be committed to the VCS.

☑ Lock on error
Whether the component should be locked on repository errors.

Source language: English
Language used for source strings in all components

Language filter: ^(cs|hu|ru)$
Regular expression used to filter translation files when scanning for filemask.

Variants regular expression:
Regular expression used to determine variants of a string.

Priority: Medium
Components with higher priority are offered first to translators.

☐ Restricted component
Restrict access to the component to only those explicitly given permission.

Save and add another | Save and continue editing | SAVE

参见:

组件配置, 双语和单语格式

单语言组件

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，请参见组件配置 ）:

Weblate administration

WELCOME, **WEBLATE TEST**. RETURN TO WEBLATE / DOCUMENTATION / CHANGE PASSWORD / SIGN OUT

Home › Weblate translations › Components › Add Component

Add Component

IMPORT SPEED DOCUMENTATION

Required fields are marked in bold.

Component name: Android
Display name

URL slug: android
Name used in URLs and filenames.

Project: WeblateOrg

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: weblate://weblateorg/language-names
URL of a repository, use weblate://project/component to share it with other component.

Repository push URL:
URL of a push repository, pushing is turned off if empty.

Repository browser:
Link to repository browser, use {{branch}} for branch, {{filename}} and {{line}} as filename and line placeholders.

Exported repository URL:
URL of repository where users can fetch changes from Weblate

Source string bug reporting address:
Email address for reports on errors in source strings. Leave empty for no e-mails.

Repository branch:
Repository branch to translate

Push branch:
Branch for pushing changes, leave empty to use repository branch

Filemask: app/src/main/res/values-*/strings.xml
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.

Template for new translations:
Filename of file used for creating new translations. For gettext choose .pot file.

File format: Android String Resource

☐ Locked
Locked component will not get any translation updates.

☑ Allow translation propagation
Whether translation updates in other components will cause automatic translation in this one

☑ Turn on suggestions
Whether to allow translation suggestions at all.

☐ Suggestion voting
Whether users can vote for suggestions.

Autoaccept suggestions: 0
Automatically accept suggestions with this number of votes, use 0 to turn it off.

Translation flags:
Additional comma-separated flags to influence quality checks. Possible values can be found in the documentation.

Enforced checks: []
List of checks which can not be ignored.

Translation license: MIT License

Contributor agreement:
User agreement which needs to be approved before a user can translate this component.

Adding new translation: Create new language file
How to handle requests for creating new translations.

Language code style: Default based on the file format
Customize language code used to generate the filename for translations created by Weblate.

Merge style: Rebase
Define whether Weblate should merge the upstream repository or rebase changes onto it.

Commit message when translating:
Translated using Weblate ({{ language_name }})

Currently translated at {{ stats.translated_percent }}% ({{ stats.translated }} of {{ stats.all }} strings)

Translation: {{ project_name }}/{{ component_name }}
Translate-URL: {{ url }}

You can use template language for various info, please consult the documentation for more details.

Commit message when adding translation:
Added translation using Weblate ({{ language_name }})

You can use template language for various info, please consult the documentation for more details.

Commit message when removing translation:
Deleted translation using Weblate ({{ language_name }})

You can use template language for various info, please consult the documentation for more details.

Commit message when merging translation:
Merge branch '{{ component_remote_branch }}' into Weblate.

You can use template language for various info, please consult the documentation for more details.

Commit message when addon makes a change:
Update translation files

Updated by "{{ addon_name }}" hook in Weblate.

Translation: {{ project_name }}/{{ component_name }}
Translate-URL: {{ url }}

You can use template language for various info, please consult the documentation for more details.

Committer name: Weblate

Committer e-mail: noreply@weblate.org

☑ Push on commit
Whether the repository should be pushed upstream on every commit.

Age of changes to commit: 24
Time in hours after which any pending changes will be committed to the VCS.

☑ Lock on error
Whether the component should be locked on repository errors.

Source language: English
Language used for source strings in all components

Language filter: ^[^i]+$
Regular expression used to filter translation files when scanning for filemask.

Variants regular expression:
Regular expression used to determine variants of a string.

Priority: Medium
Components with higher priority are offered first to translators.

☐ Restricted component
Restrict access to the component to only those explicitly given permission.

Save and add another | Save and continue editing | SAVE

参见:

组件配置, 双语和单语格式

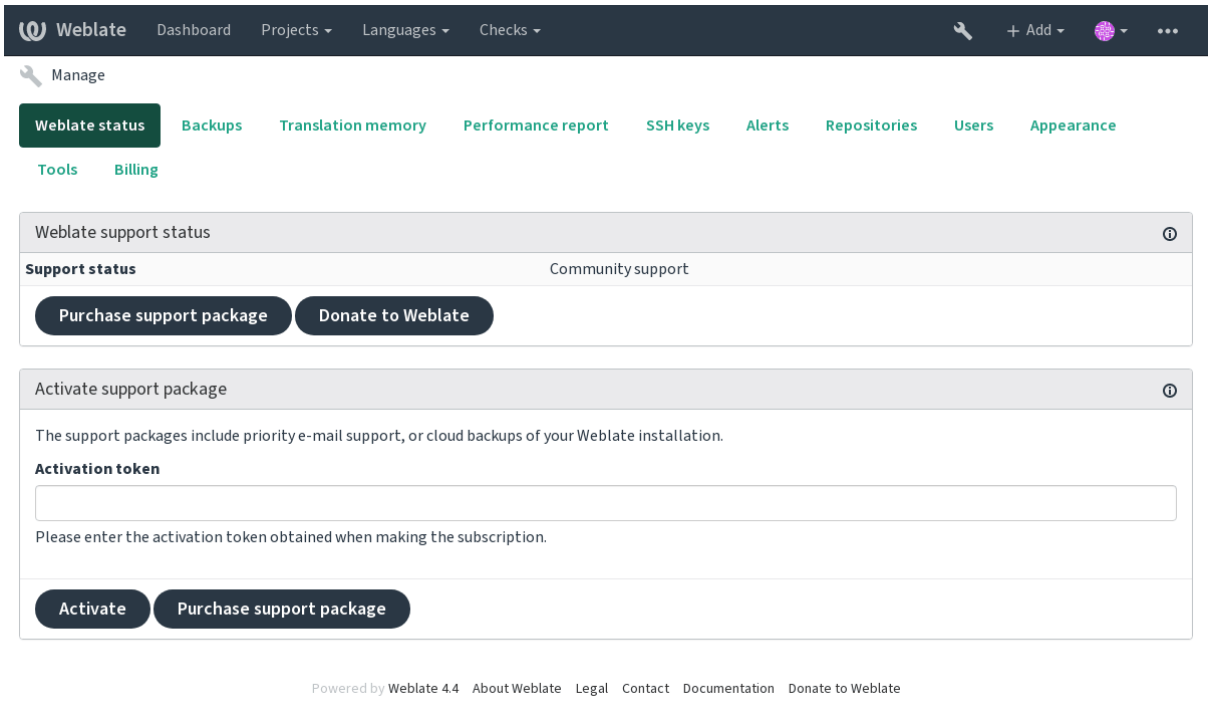
2.24 从 Weblate 获得支持

Weblate 拥有非盈利版权的开源软件，由社区支持。订购者接受优先支持而无需额外的费用。预付方式可以使软件包被每个人获得。您可以在 <https://weblate.org/support/>，找到关于当前提供的支持的更多信息。

2.24.1 集成支持

3.8 新版功能.

购买的支持包可以可选地集成在您的 Weblate 的 'subscription management <https://weblate.org/user/>'_ 界面中，您可以在那里找到它的链接。关于您的安装的基本事例的细节，也会以这种方式反馈给 Weblate。



2.24.2 提交给 Weblate 的数据

- 配置 Weblate 事件的 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、组件、语言、源字符串和用户）
- 您事件的 SSH 公钥

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*

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。

2.25 Legal documents

注解: Herein you will find various legal information you might need to operate Weblate in certain legal jurisdictions. It is provided as a means of guidance, without any warranty of accuracy or correctness. It is ultimately your responsibility to ensure that your use of Weblate complies with all applicable laws and regulations.

2.25.1 ITAR and other export controls

Weblate can be run within your own datacenter or virtual private cloud. As such, it can be used to store ITAR or other export-controlled information, however, end users are responsible for ensuring such compliance.

The Hosted Weblate service has not been audited for compliance with ITAR or other export controls, and does not currently offer the ability to restrict translations access by country.

2.25.2 US encryption controls

Weblate does not contain any cryptographic code, but might be subject export controls as it uses third party components utilizing cryptography for authentication, data-integrity and -confidentiality.

Most likely Weblate would be classified as ECCN 5D002 or 5D992 and, as publicly available libre software, it should not be subject to EAR (see Encryption items NOT Subject to the EAR).

Software components used by Weblate (listing only components related to cryptographic function):

Python See https://wiki.python.org/moin/PythonSoftwareFoundationLicenseFaq#Is_Python_subject_to_export_laws.3F

GnuPG Optionally used by Weblate

Git Optionally used by Weblate

curl Used by Git

OpenSSL Used by Python and cURL

The strength of encryption keys depends on the configuration of Weblate and the third party components it interacts with, but in any decent setup it will include all export restricted cryptographic functions:

- In excess of 56 bits for a symmetric algorithm
- Factorisation of integers in excess of 512 bits for an asymmetric algorithm
- Computation of discrete logarithms in a multiplicative group of a finite field of size greater than 512 bits for an asymmetric algorithm
- Discrete logarithms in a group different than above in excess of 112 bits for an asymmetric algorithm

Weblate doesn't have any cryptographic activation feature, but it can be configured in a way where no cryptography code would be involved. The cryptographic features include:

- Accessing remote servers using secure protocols (HTTPS)

- Generating signatures for code commits (PGP)

参见:

Export Controls (EAR) on Open Source Software

CHAPTER 3

Contributor docs

3.1 为 Weblate 做贡献

有十多种方法来为 Weblate 做出贡献。任何帮助都是受欢迎的，编写代码、图片设计、文档或赞助：

- *在 Weblate 中汇报问题*
- *开始为 Weblate 贡献代码*
- *翻译 Weblate*
- *为 Weblate 开发提供资金*

3.1.1 翻译 Weblate

Weblate 使用自身来翻译 translated，请随意加入，使 Weblate 以尽可能多的人类语言来提供使用。

3.1.2 为 Weblate 开发提供资金

可以在 *donate page*‘_ 上资助 *Weblate* 进一步开发。收集的资金用于资助免费主办自由软件项目，以及 *Weblate* 的进一步开发。具体内容如自主目标和资助者的回报，请查看 *‘donate page*。

资助 Weblate 的支持者

Weblate 支持者列表：

- Yashiro Ccs
- Cheng-Chia Tseng
- Timon Reinhard
- Cassidy James
- Loic Dachary
- Marozed
- https://freedombox.org/

- GNU Solidario (GNU Health)
- BallotReady

想要加入列表中吗？请查看 Donate to Weblate 上的选项。

3.2 开始为 Weblate 贡献代码

为了理解 Weblate 源代码，请首先查看*Weblate 源代码*、*Weblate 前端* 和*Weblate 内部*。

3.2.1 用我们的代码库开始

如果查找一些缺陷来是自己熟悉 Weblate 代码库，那么请查看标记 good first issue 的那些。

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文件安装。它将给您一个带有可编辑的 Weblate 源代码的虚拟环境。

1. 复制 Weblate 源码：

```
git clone https://github.com/WeblateOrg/weblate.git
cd weblate
```

2. 创建一个虚拟环境（virtualenv）：

```
virtualenv .venv
.venv/bin/activate
```

3. 安装 Weblate（这需要一些系统依赖性，请参见从源文件安装）：

```
pip install -e .
```

3. 安装用于开发的所有依赖性：

```
pip install -r requirements-dev.txt
```

4. 启动一台开发服务器：

```
weblate runserver
```

5. 依赖于配置，您会想要开始 Celery workers：

```
./weblate/examples/celery start
```

6. 运行测试（更多细节请参见本地测试）：

```
. scripts/test-database
./manage.py test
```

参见：

从源文件安装

3.2.3 在 Docker 中本地运行 Weblate

如果安装了 Docker 和 docker-compose，您可以启动开发环境，只通过简单地运行：

```
./rundev.sh
```

它将新建 Docker 影响并启动它。Weblate 运行在 <http://127.0.0.1:8080/> 上，并且您可以以 `admin` 用户和 `admin` 密码来登录。新的安装是空的，所以您会想要以添加翻译项目和组件 来继续。

对此，`Dockerfile` 和 `docker-compose.yml` 位于本地的 `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，然后指定任何 `test` 参数，例如：

```
./rundev.sh test --failfast weblate.trans
```

注解： 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
```

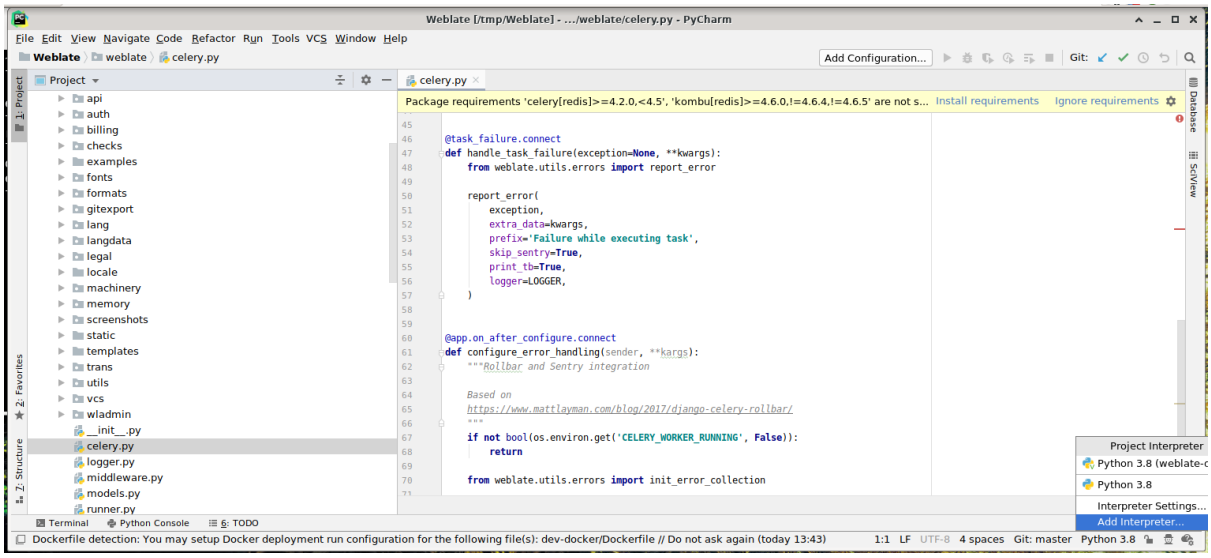
运行没有任何参数的脚本将重建 Docker 容器并重启动它。

注解： 这不是用于生产的合适设置，它包括几个不安全的破解，但使开发更容易。

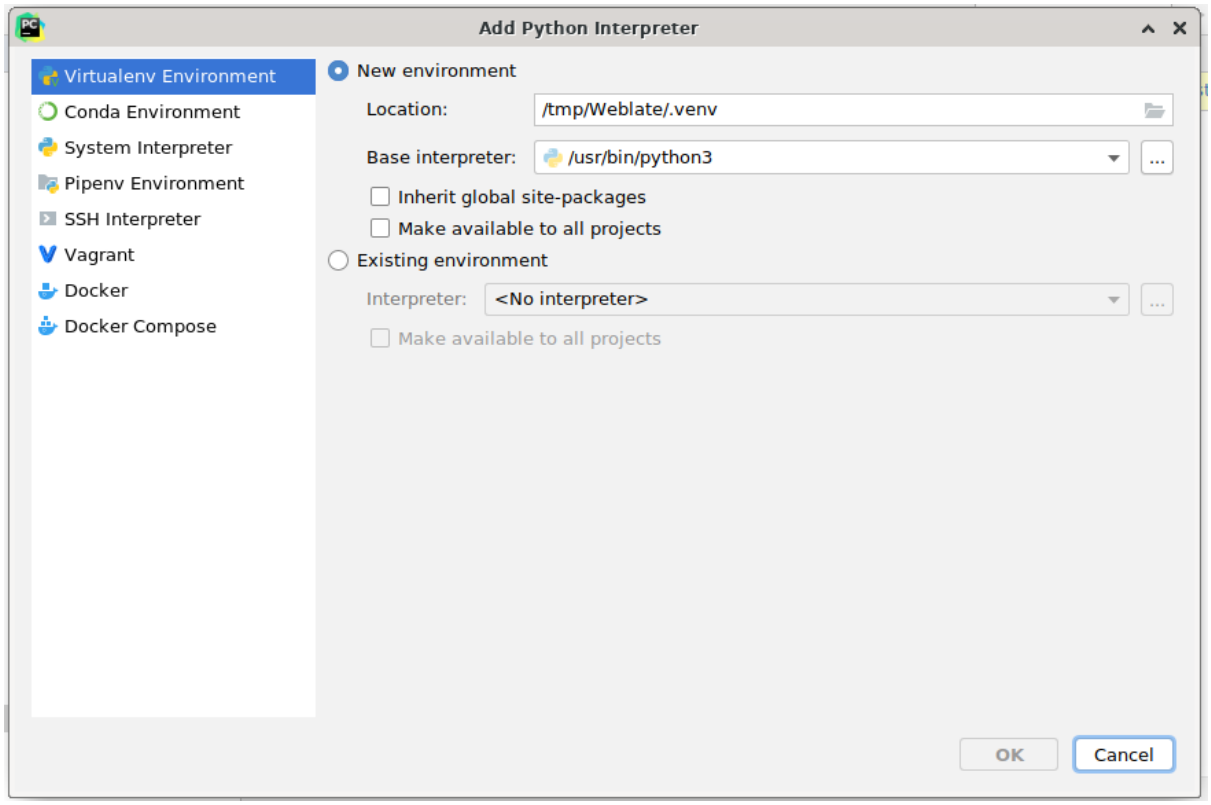
3.2.4 使用 PyCharm 为 Weblate 编写代码

PyCharm 是 Python 的著名 IDE，这里是一些指示，帮助您在其中设置 Weblate 项目。

考虑刚刚克隆了 Github 仓库，只是在 PyCharm 中打开将其克隆进去的文件夹。一旦打开 IDE，第一步是指定想要的解释程序：

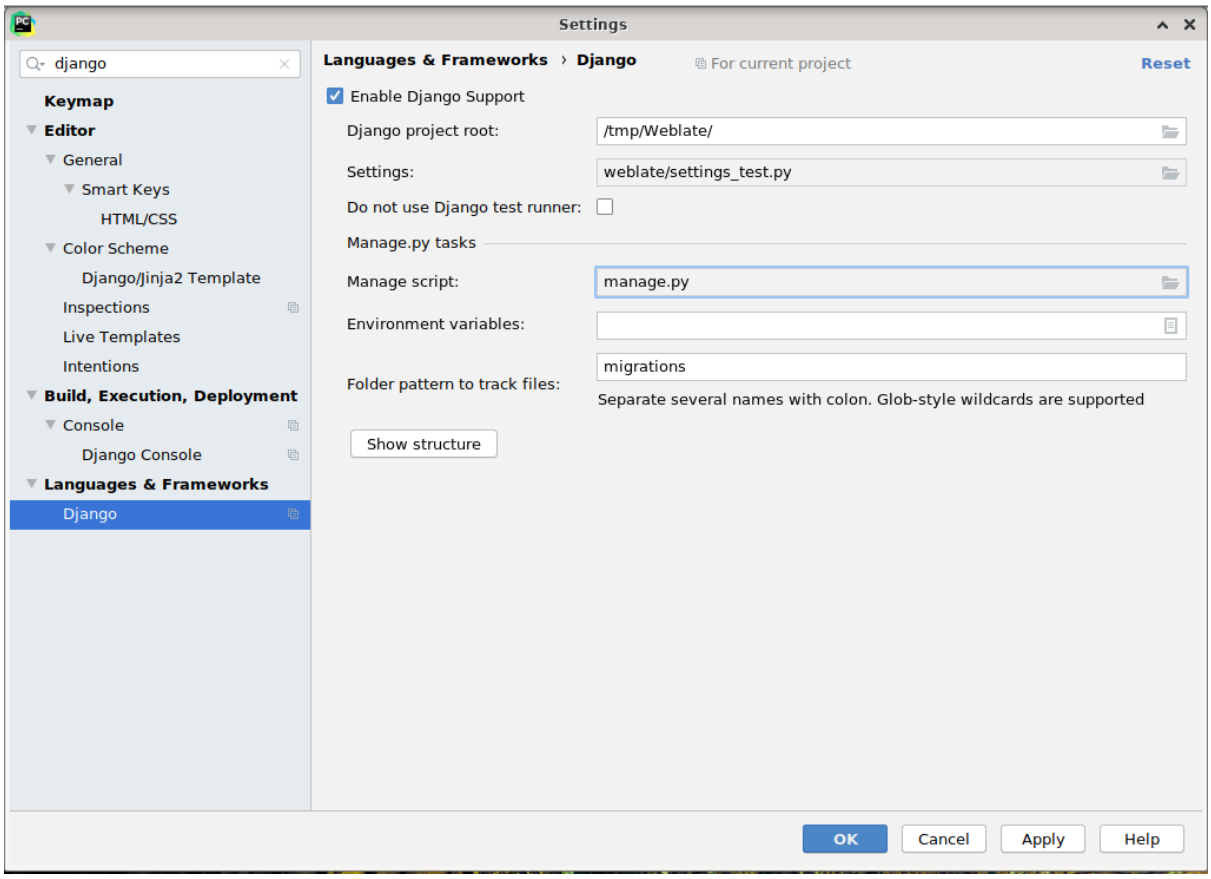


可以或者选择让 PyCharm 为您创建 virtualenv（虚拟环境），或者选择已经存在的：



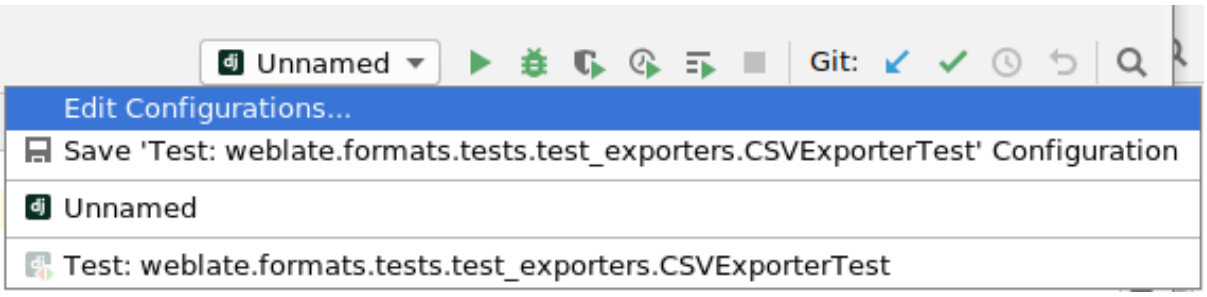
一旦设置了解释程序就不要忘记安装依赖性：可以通过控制台来进行（来自 IDE 的控制台将默认直接使用虚拟环境），或者在得到丢失依赖性的警告是通过界面来进行。

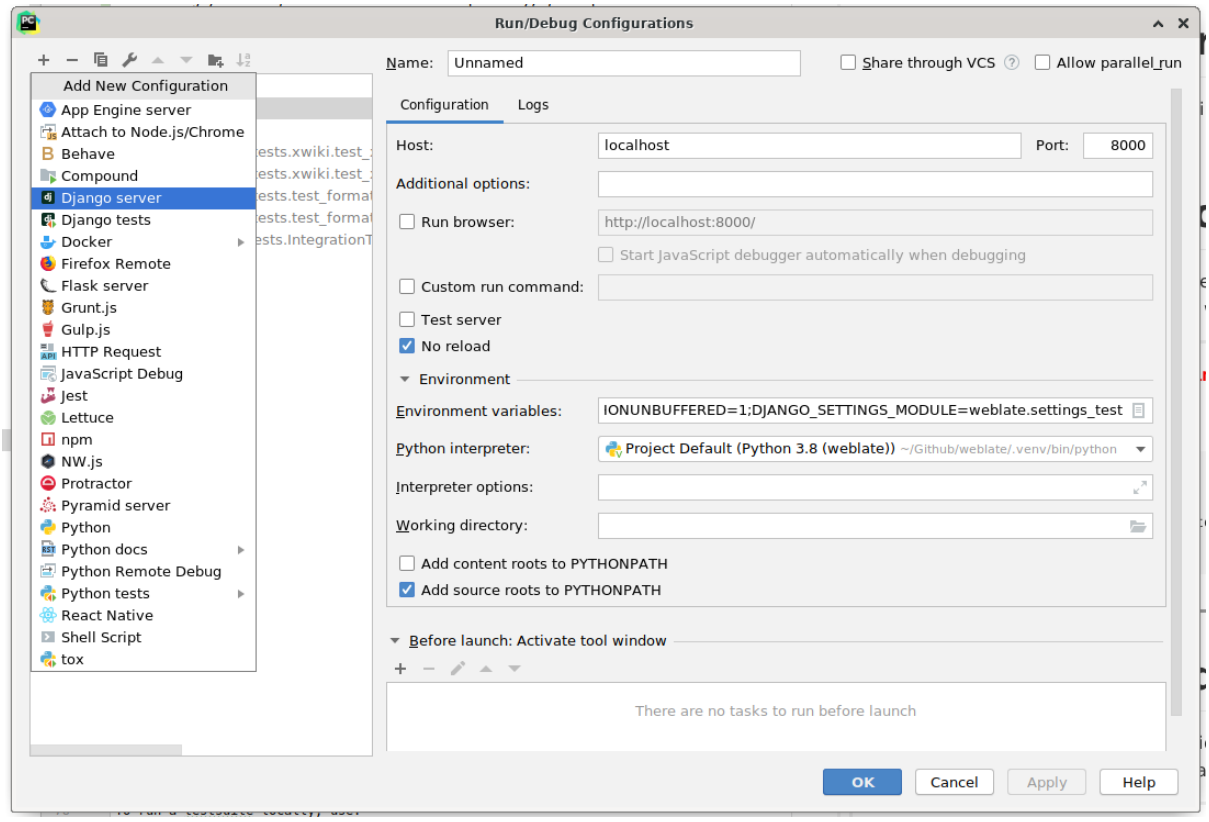
第二步是设置正确的信息，在 PyCharm 内使用本地 Django：理念是能够立即触发 IDE 中的单元测试。为此，你需要指定该 Django 项目的根路径及其设置路径：



要小心，*Django project root*‘（*Dejango* 项目的根）是仓库的根，而不是 *weblate* 子目录。关于设置，我个人使用来自仓库的 ‘*settings_test*，但您可以新建自己的设置，并在那里设置。

最后一步是能够运行服务器，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，如果勾选了，那么当修改文件时服务器的现场重载不会发生。这允许现有的调试断点被持续保留，因为它们将在重载时丢弃。

3.2.5 创立引得开发实例

您会想要使用 `import_demo` 来新建演示翻译，并且使用 `createadmin` 来创建管理用户。

3.3 Weblate 源代码

Weblate 在 GitHub 上开发。欢迎您将代码分叉并打开拉去请求。同样欢迎任何形式的补丁。

参见:

查询 *Weblate 内部* 看看 Weblate 从内部看是什么样子的。

3.3.1 由设计原理来提供安全性

编写 Weblate 的任何代码应该时刻记得‘Security by Design Principles‘_（由设计原理来提供安全性）。

3.3.2 编程标准

代码应该符合 PEP-8 变成指南，并且应该使用 **black** 代码格式化程序来格式化。

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。Weblate 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

缺陷的症状可以是应用崩溃或行为错误。欢迎您搜集任何这样的问题的信息，并将其提交给 issue tracker。

3.4.1 调试模式

打开调试模式将在浏览器中显示例外。这在 web 界面上调试问题非常有用，但不适于生产环境，因为这回导致性能问题，并泄露私有数据。

参见:

禁止调试模式

3.4.2 Weblate 日志

Weblate 可以生成关于后台什么在运行的具体日志。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（依赖于您的 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。示例的系统范围的设置被记录到 `/var/log/celery/` 下的几个日志文件中。

Docker 容器将其输入记入日志（如同 Docker 世界中通常的那样），因此可以使用 `docker-compose logs` 来查看日志。

参见:

配置的例子 包含 `LOGGING` 配置。

3.4.3 不处理后台任务

很多事情发生在后台 Celery workers 中。像发送电子邮件或删除组件这样的事情不起作用时，可能会是它有问题。

在那种情况下需要检查的事情：

- 检查 Celery 是否正在运行，见: ref:*celery*
- 或者在*管理界面* 中或者使用 using *`celery_queues`* 检查 Celery 队列状态
- 在 Celery 日志中查找错误（请参见*Weblate 日志*）

3.4.4 不接收来自 Weblate 的电子邮件

可以通过使用 `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* 标签下的*管理界面* 来验证向外发送的电子邮件是否正常工作。

这直接发送电子邮件，所以这确定了您的 SMTP 配置是正确的（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

在应用崩溃的情况下，尽可能多地收集与崩溃有关的信息是非常有用的。最容易的方式是使用第三方服务来实现，这些服务会自动收集这样的信息。可以在:ref:‘collecting-errors‘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

Weblate 在这样的情形性能很差的情况下，请收集显示问题的相关日志，以及有助于找到从哪里改进代码的任何事物。

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。

3.5 Weblate 内部

注解： 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 行为的插件，请参见:ref:‘addons。

`api`

基于 Django REST framework 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`

字体提供检查模块。

`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*)

classmethod can_install(*component*, *user*)
检查附加组件是否与给定组件兼容。

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格。

get_settings_form(*user*, ***kwargs*)
返回此附加组件的配置表格。

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*) –插件操作是否应该跳过更改上游推送。通常可以将这个传递给潜在的方法，作为 commit_and_push 或 commit_pending。

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。

stay_on_create = False
Weblate 附加组件的基础类。

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。

这里是一个插件的例子：

```
#
# Copyright © 2012 - 2021 Michal Čihař <michal@cihar.com>
#
# This file is part of Weblate <https://weblate.org/>
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/>.
#


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 addon should receive
    events = (EVENT_PRE_COMMIT,)
    # Addon unique identifier
    name = "weblate.example.example"
    # Verbose name shown in the user interface
    verbose = _("Example addon")
    # Detailed addon description
    description = _("This addon does nothing it is just an example.")

    # Callback to implement custom behavior
```

(下页继续)

(续上页)

```
    def pre_commit(self, translation, author):
        return
```

3.7 Weblate 前端

前端当前使用 Bootstrap、jQuery 和一些第三方库来构建。

3.7.1 支持的浏览器

Weblate 支持所有主要浏览器和平台的最新的、稳定的版本。

不明确支持使用最新版的 WebKit、Blink 或 Gecko 的替换浏览器，无论是否直接还是通过平台的 web 视图 API。然而，Weblate 应该（在多数情况下）页在这些浏览器中正常显示并工作。

其它浏览器也能工作，但一些特性会受到限制。

3.7.2 依赖性管理

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，所有的第三方前端膘满都位于那里。

3.7.3 编码风格

Weblate 依赖于 Prettier 来进行 JavaScript 和 CSS 文件的代码格式化。

我们还是用 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:

Translation topic in the Django documentation

3.7.5 图标

Weblate 目前使用 material design 图标。如果你想找新的符号，检查 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

我们的 issue tracker 在 GitHub 上提供主机：

欢迎报告 Weblate 的任何问题，或改进建议。如果您所发现的是 Weblate 中的安全问题，请咨询下面的 "Security issues" 章节。

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。

另外，可以汇报给 security@weblate.org，最后也会到 HackerOne 上。

如果因为任何原因不想使用 HackerOne，可以将报告通过电子邮件发送给 michal@cihar.com。可以选择使用这个 PGP 密钥 *3CB 1DF1 EF12 CF2A C0EE 5A32 9C27 B313 42B7 511D* 来加密。还以从 Keybase 得到密钥。

注解： Weblate 在很多部分上依赖于第三方组件。在发现通常影响这些组件之一的漏洞的情况下，请直接会报告给各自的项目。

这些中的一些是：

- Django
- Django REST 框架
- Python Social Auth

3.9 Weblate 测试套件与连续集成

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 连续集成

当前的测试结果可以在 GitHub Actions 上找到，并且覆盖范围在 Codecov 上报告。

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 中：

```
# Simple way to configure test database from environment

# 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
./ci/run-migrate
./ci/run-test
./ci/run-docs
./ci/run-setup
```

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*连续集成*），并且可以使用环境变量调整：

```
# Simple way to configure test database from environment

# 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
```

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 Testing in Django。

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	*array*		
项目	翻译记忆项		
	类型	对象	
	属性		
	• **类别**	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
		例子	1
		最小	0
		默认	1
	• **来源**	字符串源头	
		文件名或组件名	
		类型	字符串
		例子	检测
		默认	
	• **源**	源字符串	
		类型	字符串
		例子	你好
		最小长度	1
		默认	
	• **源 _ 语言**	源语言	
		ISO 639-1 / ISO 639-2 / IETF BCP 47	
		类型	字符串
		例子	英语
		模式	^[^]+$
		默认	
	• **目标**	目标字符串	
		类型	字符串
		例子	Ahoj
		最小长度	1
		默认	
	• **目标 _ 语言**	目标语言	
		ISO 639-1 / ISO 639-2 / IETF BCP 47	
		类型	字符串
		例子	cs
		模式	^[^]+$
		默认	
	额外属性	假	

下页继续

表 1 – 续上页

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			
• **基本**	基础		
	类型	对象	
	属性		
	• **用户名**	用户名	
		类型	字符串
		例子	管理员
		默认	
	• **完整 _ 名称**	完整名称	
		类型	字符串
		例子	Weblate 管理员
		默认	
	• **电子邮箱**	电子邮件	
		类型	字符串
		例子	noreply@example.com
		默认	
	• **加入 _ 日期**	加入日期	
		类型	字符串
		例子	2019-11-18T18:53:54.862Z
		默认	
• **个人资料**	个人资料		
	类型	对象	
	属性		
	• **语言**	语言	
		类型	字符串
		例子	cs
		模式	^.*$
		默认	
	• **已建议**	建议的字符串数量	
		类型	整数
		例子	1
		默认	0
	• **已翻译**	已翻译字符串数量	
		类型	整数
		例子	24
		默认	0
	• **已上传**	已上传的截屏数	
		类型	整数
		例子	1
		默认	0
	• **hide_completed**	在控制面板上隐藏已完成的翻译	
		类型	布尔值
		例子	假
		默认	真
	• **secondary_in_zen**	在禅模式下显示第二翻译	

下页继续

表 2 – 续上页

		类型	布尔值	
		例子	真	
		默认	真	
	• **hide_source_secondary**	如果有第二翻译则隐藏原文		
		类型	布尔值	
		例子	假	
		默认	真	
	• **编辑者 _ 链接**	编辑者链接		
		类型	字符串	
		例子		
		模式	^.*$	
		默认		
	• **翻译 _ 模式**	翻译编辑器模式		
		类型	整数	
		例子	0	
		默认	0	
	• **zen_mode**	禅编辑器模式		
		类型	整数	
		例子	0	
		默认	0	
	• **特殊 _ 字符**	特殊字符		
		类型	字符串	
		例子		
		模式	^.*$	
		默认		
	• **控制面板 _ 视图**	控制面板默认视图		
		类型	整数	
		例子	1	
		默认	0	
	• **dashboard_component_list**	默认组件列表		
		默认	空	
		anyOf	类型	空
			类型	整数
	• **语言**	已翻译的语言		
		类型	*array*	
		默认		
		项目	语言代码	
			类型	字符串
			例子	cs
			模式	^.*$
			默认	
	• **第二 _ 语言**	第二语言		
		类型	*array*	
		默认		
		项目	语言代码	
			类型	字符串
			例子	sk
			模式	^.*$
			默认	
	• **已关注**	已关注项目		
		类型	*array*	
		默认		
		项目	项目标识串	
			类型	字符串
			例子	weblate
			模式	^.*$

下页继续

表 2 – 续上页

			默认	
• **审计日志**	审计日志			
	类型	*array*		
	默认			
	项目	项目		
		类型	对象	
		属性		
		• **地址**	*IP* 地址	
			类型	字符串
			例子	127.0.0.1
			模式	^.*$
			默认	
		• 用户 _ 代理	用户代理	
			类型	字符串
			例子	PC / Linux / Firefox 70.0
			模式	^.*$
			默认	
		• **时间戳**	时间戳	
			类型	字符串
			例子	2019-11-18T18:58:30.845Z
			模式	^.*$
			默认	
		• **活动**	活动	
			类型	字符串
			例子	登录
			模式	^.*$
			默认	
定义				

参见:

用户个人资料, `dumpuserdata`

3.11 发布 Weblate

3.11.1 发布日程

Weblate 有两个月的发布周期，版本（x.y）。

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

到来的版本的特性使用 Github 里程碑来收集，可以在 <https://github.com/WeblateOrg/weblate/milestones> 看到路线图。

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 -C docs update-screenshots**。

执行发布：

4. 创建一个发布 **./scripts/create-release --tag**（要求请参见下面）。

张贴发布手动步骤：

5. 更新 Docker 图像。
6. 管理 GitHub 里程碑。
7. 一旦检测到 Docker 图像，则添加标签并推送。
8. 将 Helm 图表更新到新的版本。
9. 在 `.github/workflows/migrations.yml` 中包含新的版本，从而在合并监测中覆盖它。
10. 由 **./scripts/set-version** 在仓库中增加版本。

要使用 **./scripts/create-release** 脚本来创建标记的话，需要后面的：

- 带有私钥的 GnuPG 用于为发布签名
- 推送访问 Weblate git 仓库（它推送标记）
- 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（Weblate 下载复制到那里）

3.12 关于 Weblate

3.12.1 项目目标

基于 Web 的连续本地化工具，通过版本控制集成 紧密支持广大范围的支持的文件格式，使翻译者容易地做出贡献。

3.12.2 项目名称

“Weblate” 由单词” web” 和” translate” 融合而来。

3.12.3 项目网站

着陆页面是 <https://weblate.org/> 而云在 <https://hosted.weblate.org/> 上作为服务的主机。本文档可以在 <https://docs.weblate.org/> 找到。

3.12.4 项目标识

项目标识和其它图片在 <https://github.com/WeblateOrg/graphics/> 仓库中得到。

3.12.5 领导

本项目由 Michal Čihař <michal@cihar.com> 来维护。

3.12.6 作者

Weblate 由 Michal Čihař <michal@cihar.com> 启动。自 2012 年创始以来，上千名人员做出了贡献。

3.13 许可协议

CHAPTER 4

Change History

4.1 Weblate 4.4.1

Released on January 13th 2021.

- 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 格式的清理插件的行为。
- 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 上复制拉取请求。
- 改进了显示 ghost 翻译时的性能。
- 重新实现翻译编辑器，使用原生浏览器文本区。
- Fixed cleanup addon breaking adding new strings.
- 添加了附加组件 API。

4.2 Weblate 4.4

发布于 2020 年 12 月 15 日。

- 改进了新建组件时的验证。
- 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告。
- 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工作流程。
- 扩展了翻译统计 API。
- 改进了翻译时“其他翻译”标签页。
- 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。

4.3 Weblate 4.3.2

发布于 2020 年 11 月 4 日。

- 修复了某个组件文件掩码的崩溃。
- 改善了对连续重复的单词进行准确性检查的精度。
- 增加了对 Pagure 拉取请求的支持。
- 改进了注册失败的错误消息。
- 恢复了将开发者注释解析为 Markdown 格式。
- 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“源”，来明确其意义。
- 修复了路径较长代码仓库的 GitLab 拉取请求。

4.4 Weblate 4.3.1

发布于 2020 年 10 月 21 日。

- 改进了自动翻译性能。
- 修复了授权用户会话到期问题。
- 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。
- 改进了象限视图的性能。
- 增加了将一名用户从一个项目移除前的确认功能。

4.5 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组件。
- 为词汇表确定了语言。
- 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。
- 扩展了公共的用户配置文件。
- 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组件。
- 添加了 Vue I18n 格式检查。
- 一般占位符的检查现在支持了正则表达式。
- 改进了象限模式的外观。
- 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，以及词汇表。
- 单元 API 现在能正常处理多个字符串。
- 组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。
- 为安卓自动启动 java 格式。
- 增加了对本地化截图的支持。
- 增加了对 Python 3.9 的支持。
- 修复了某些条件下翻译 HTML 文件。

4.6 Weblate 4.2.2

发布于 2020 年 09 月 02 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。
- 修复了启动预告时的 Git 提交变形。
- 修复了使用 API 创建本地版本控制系统（VCS）组件。

4.7 Weblate 密钥

发布于 2020 年 08 月 21 日。

- 修复了在安装资源中一些区域设置存储复数。
- 修复了为一些 XLIFF 文件清理插件的崩溃。
- 允许在 Docker 映像中设置本地化 CDN。

4.8 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。
- 可以配置字符串附近显示的数字。
- 增加了对锁定遇到存储库错误的组件的支持。
- 简化了主导航（用图表替换按钮）。
- 改进了 Google Translate 集成中的语言代码处理。
- Git 变形插件可以创建 “Co-authored-by:“预告。
- 改进了查询搜索解析。
- 改进了格式字符串检查的用户反馈。

- 改进了大量的状态更改的性能。
- 添加了项目或组件重命名后重定向的兼容性。
- 为字符串的统一、组件的锁定和许可的更改添加了通知。
- 为 ModernMT 添加了支持。
- 允许在文件上传时避免覆盖已同意的翻译。
- 去掉了一些对兼容 URL 重定向的支持。
- 添加了对 ECMAScript 模板文本的检查。
- 添加了监视组件的选项。
- 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组件。
- 允许配置 `Content-Security-Policy` HTTP 标头。
- 在项目层为语言别名添加支持。
- 现在插件帮助 HTML 或 JavaScript 本地化，请参见*JavaScript 本地化 CDN*。
- Weblate 域现在在设置中配置，请参见`SITE_DOMAIN`。
- 增加了对按照组件和项目进行搜索的支持。

4.9 Weblate 4.1.1

发布于 2020 年 06 月 19 日。

- 修复了 Docker 中更改自动修复或插件配置。
- 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。
- 修复了向词汇表添加单词。
- 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*。

4.10 Weblate 4.1

发布于 2020 年 06 月 15 日。

- 为使用包含的国家代码新建新的翻译添加了支持。
- 增加了对用截图搜索源字符串的支持。
- 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。

- 在项目新建表格中包括了源语言。
- 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。
- 改进了词汇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。
- 为声明中的标记和通知配置添加了支持。
- 扩展了检查列表的细节。
- 为新的文件格式：*Laravel PHP 字符串*, *HTML files*, *OpenDocument Format*, *IDML Format*, *Windows RC files*, *INI translations*, *Inno Setup INI 翻译*, *GWT properties*, *go-i18n JSON files*, :ref:'arb'添加了支持。
- 一致地使用了放弃作为放弃检查的状态。
- 为了配置默认启动的插件添加了支持。
- 修复了编辑器对放弃检查的键盘快捷键。
- 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量。
- 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。
- 修复了忽略检查源的选项。
- 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。
- 添加了对组件层访问限制的能力。
- 为翻译标记中的空白字符和其它特殊字符添加了支持，请参见定制行为。
- 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。
- 为项目之间分享词汇表添加了支持。

4.11 Weblate 4.0.4

发布于 2020 年 05 月 07 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 新建组件的问题。
- 修复了移动导航中爆发的 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。

4.12 Weblate 4.0.3

发布于 2020 年 05 月 02 日。

- 修复了报告中可能的崩溃。
- 用户在注释中的提及现在不区分大小写。
- 修复了非超级用户的 PostgreSQL 迁移。
- 修复了新建组件时更改仓库 URL。
- 修复了上游仓库丢失时的崩溃。

4.13 Weblate 4.0.2

发布于 2020 年 04 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。
- 改进了大量编辑的性能。
- 改进了翻译记忆库的性能。
- 修复了组件删除时可能的崩溃。
- 修复了一些极端情况下显示翻译更改的问题。
- 改进了 celery 队列过长的警告。
- 改进了一致性检查中的误报。
- 修复了更改连接的组件仓库时的死锁。
- 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。
- 改进了双语言源上传。

- 为 DeepL 机器翻译自动检测支持的语言。
- 修复了一些极端情况下的进度条显示。
- 修复了非翻译字符串出发的一些检查。

4.14 Weblate 4.0.1

发布于 2020 年 04 月 16 日。

- 修复了来自 Pypi 的软件包安装。

4.15 Weblate 4.0

发布于 2020 年 04 月 16 日。

- Weblate 现在需要 Python 3.6 或更新版本。
- 添加了组件提醒的管理概述。
- 添加了断裂的仓库浏览器 URLs 的组件提醒。
- 改进了登陆和注册页面。
- 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。
- 显示的建议未能通过检查。
- 在历史中记录源字符串更改。
- 将 Microsoft Translator 升级为版本 3 的 API。
- 重新应用翻译记忆库后端。
- 为在*Searching* 中查找几个 `is:` 添加了支持。
- 允许*未更改的翻译* 避免内部黑名单。
- 改进了从单语言 po 文件中提取注释。
- 重新命名要宣布的白板消息。
- 修复了注册邮件偶尔出现的问题。
- 改进了 LINGUAS 更新插件来处理更多的语法变量。
- 修复了编辑单语言 XLIFF 源文件。
- 为*Searching* 中的准确匹配添加了支持。
- 扩展了 API 覆盖屏幕截图、用户、用户组、组件列表，并扩展了新建项目。
- 为双语言翻译上传的源添加支持。
- 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。

4.16 Weblate 3.x 系列

4.16.1 Weblate 3.11.3

发布于 2020 年 03 月 11 日。

- 修复了以某种优先性来搜索字段。
- 修复了近期添加的字符串的预定义队列。
- 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。
- 添加了到摘要通知中的事件的链接。
- 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。
- 修复了插件的文档链接。
- 降低了为更改翻译检查的误报。
- 提高了处理 CVE-2020-6802 的 bleach 依赖性。
- 修复了在历史中列出项目层的更改。
- 修复了一些极端情况下的统计数据被验证为非法。
- 修复了搜索某个字符串状态。
- 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。

4.16.2 Weblate 3.11.2

发布于 2020 年 02 月 22 日。

- 修复了提出建议的问题。
- 修复了一些字符串被错误地报告为没有单词的问题。

4.16.3 Weblate 3.11.1

发布于 2020 年 02 月 20 日。

- 存档的 Celery 设置更改。
- 改进了新建组件时文件名的验证。
- 修复了一些依赖性的最低版本。
- 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。
- 改进了词汇表匹配。

4.16.4 Weblate 3.11

发布于 2020 年 02 月 17 日。

- 允许通过 API 新建组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像。
- 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。
- 在系统检查时推荐更新为新的 Weblate 版本。
- 对重复的语言提醒提供了更具体的分析。
- 在项目页面上包括更具体的版本信息。
- 如果需要的话自动非浅复制本地复件。
- 修复了需要动作的字符串下载。
- 新的提醒，警告使用相同框架两次。
- 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。
- 添加了选项来解决注释。
- 添加了标记的大量编辑。
- 为 labels 添加了支持。
- 添加了大量编辑插件。
- 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。
- 修复了已被翻译 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 插件现在在 weblate 中完全同步翻译。

4.16.5 Weblate 3.10.3

发布于 2020 年 01 月 18 日。

- 支持 translate-toolkit 2.5.0。

4.16.6 Weblate 3.10.2

发布于 2020 年 01 月 18 日。

- 为项目添加了锁定指示。
- 修复了在一些 web 浏览器中促使闪退的 CSS 缺陷。
- 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 Git 钱克隆的配置。
- 改进了后台通知处理。
- 修复了在 web 浏览器导航回去时表格提交中断。
- 配置 YAML 格式化的新插件。
- 修复了单复数形式语言中没有进行相同的复数检查。
- 修复了相同字段的 regex 搜索。

4.16.7 Weblate 3.10.1

发布于 2020 年 01 月 09 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。
- 改进了更改日志的面包屑（breadcrubs）。
- 改进了面板上的翻译列表。
- 改进了 Webhooks 的 HTTP 响应。
- 在 Docker 容器中添加了 GitLab 和并请求。

4.16.8 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。
- 对只读字符串添加了支持。
- 为注释中的标记添加了支持。
- 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 仓库取回性能。

- 添加了搜索字符串时间的查找。
- 为所有翻译显示源语言。
- 显示字符串附近的语境。
- 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。
- 对连接的组件提醒避免发送重复的通知。
- 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。

4.16.9 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。

4.16.10 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。
- 改进了失败检查的 UI。
- 在格式检查中指示了丢失的字符串。
- 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、组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。
- 允许配置来为分享的翻译记忆库做出贡献。
- 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。
- 对未支持的配置添加了组件提醒。
- 添加了自动翻译插件来引导翻译。
- 扩展了自动翻译来添加建议。
- 在概览上显示插件参数。
- 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。
- 为 RESX 分离了清理插件，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组件。
- 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。

4.16.11 Weblate 3.8

发布于 2019 年 08 月 15 日。

- 为简化创建相似的组件提供了支持。
- 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。
- 改进了仓库范围插件的性能。
- 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。
- 为将插件克隆到发现的组件添加了支持。
- 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 widget 图像而在社交网络上使用。

- 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周期。
- 为 Azure Repos 添加了 webhook 支持。
- 待定建议或未翻译字符串的新的选择使用通知。
- 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。

4.16.12 Weblate 3.7.1

发布于 2019 年 06 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。

4.16.13 Weblate 3.7

发布于 2019 年 06 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。
- 在报告中包括了同意的统计数据。
- 当更新翻译组件时的报告过程。
- 允许终止运行后台组件的更新。
- 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。

- 刷新了新贡献者的登录页面。
- 为配置 msgmerge 插件添加支持。
- 当发送通知时延迟打开 SMTP 连接。
- 改进了错误日志。
- 允许 MO 生成插件中的客户位置。
- 添加了插件来清理旧的建议或注释。
- 添加了选项来启动禅编辑器的水平模式。
- 提高了许多被链接组件的导入性能。
- 修复了一些情况下的示例安装。
- 改进了更改中提供提醒。
- 添加了新水平统计数据 widget。
- 改进了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。

4.16.14 Weblate 3.6.1

发布于 2019 年 04 月 26 日。

- 改进了单语言 XLIFF 文件的处理。
- 修复了一些极端情况下的摘要通知。
- 修复了插件脚本错误提醒。
- 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。

4.16.15 Weblate 3.6

发布于 2019 年 04 月 20 日。

- 为下载用户数据添加了支持。
- 现在在安装时自动触发插件。
- 改进了解决合并冲突的指示。
- 清理插件现在与 app 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 04 月终止。
- 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，来反映源计数和目标计数。
- 现在即使对用户禁止，管理员和一致性插件也可以添加翻译。
- 修复了禁止 `Language-Team` 标头操作切换的描述。
- 通知注释中提到的用户。
- 从组件设置中删除了文件格式自动检查。
- 修复了为单语言 PO 文件生成 MO 文件。
- 添加了摘要通知。
- 为组件通知静音添加了支持。
- 为新的提醒、白板消息或组件添加了通知。
- 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。

4.16.16 Weblate 3.5.1

发布于 2019 年 03 月 10 日。

- 修复了 Celery systemd 单元例子。
- 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。
- 在日志中包括插件执行失败的输出。
- 改进了添加新语言选择的验证。
- 在组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 Microsoft Terminology 服务与当前的 Zeep 发布版本兼容。
- 本地化更新。

4.16.17 Weblate 3.5

发布于 2019 年 03 月 03 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。
- 改进了坏的组件状态的提醒。
- 添加了管理白板消息的用户界面。
- 现在可以配置提交消息的插件。
- 减少更新上游仓库时的提交数量。
- 修复了在项目之间移动组件时可能的元数据丢失。
- 改进了缠模式的导航。
- 添加了几个新的质量检查（相关的标记和 URL ）。

- 对 app 商店元数据文件添加了支持。
- 为切换 GitHub 或 Gerrit 集成添加了支持。
- 为卡什达字母添加了检查。
- 添加选项来根据作者压缩提交。
- 改进了对 XLSX 文件格式的支持。
- 与 Tesseract 4.0 兼容。
- 记账插件现在在未支付账单 45 天后删除项目。

4.16.18 Weblate 3.4

发布于 2019 年 01 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。
- 为项目与组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组件添加向导。
- 可定制的 Git 合并提交消息。
- 在导航中添加了组件提醒的视觉指示。
- 改进了装入翻译文件的性能。
- 在推送前压缩提交的新插件。
- 改进了翻译更改的显示。
- 更改了默认合并风格来 rebase 并且使之可配置。
- 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。

4.16.19 Weblate 3.3

发布于 2018 年 11 月 30 日。

- 为组件和项目删除添加了支持。
- 改进了一些单语言翻译的性能。
- 添加了翻译组件提醒，使翻译的问题高亮。
- 当可用时暴露 XLIFF 字符串 resname 作为语境。
- 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。

4.16.20 Weblate 3.2.2

发布于 2018 年 10 月 20 日。

- 删除了不再需要的 Babel 依赖性。
- 更新了语言定义。
- 改进了插件、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。
- 修复了一些极端情况下失败的翻译上传 API。
- 修复了手动更改情况下更新 Git 配置。

4.16.21 Weblate 3.2.1

发布于 2018 年 10 月 18 日。

- Python 2.7 的 backports.csv 的文档依赖性。
- 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。
- 改进了新建新组件时的验证。
- 为清理旧的建议添加了支持。

4.16.22 Weblate 3.2

发布于 2018 年 10 月 06 日。

- 为自动插件安装添加 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。
- 改进了多组件发现插件情况下的组件清理。
- 重写了 Microsoft Terminology 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。
- 删除了对客户提交消息的支持。

- 可配置的编辑模式（禅模式/全模式）。
- 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。
- 为用户新建项目和组件添加了支持。
- 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。

所提到问题的具体列表请参见 3.2 milestone on GitHub 。

4.16.23 Weblate 3.1.1

发布于 2018 年 07 月 27 日。

- 修复了一些设置中失败的测试套件。

4.16.24 Weblate 3.1

发布于 2018 年 07 月 27 日。

- 不再支持从早于 3.0.1 的更旧版本的升级。
- 允许覆盖从设置中默认提交的消息。
- 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。
- 删除了将很快关闭的 libravatar 服务的支持。
- 现在插件将未更改的翻译标记为需要编辑。
- 为翻译时跳到特定位置添加支持。
- 现在可以定制下载翻译。
- 改进了翻译记忆库匹配中的字符串相似性的计算。
- 添加了由 GnuPG 签发 Git 提交的支持。

4.16.25 Weblate 3.0.1

发布于 2018 年 06 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例子。
- 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。

4.16.26 Weblate 3.0

发布于 2018 年 06 月 01 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。
- 自动组件发现的新插件。
- 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。
- 新的插件，将贡献者姓名存储在 PO 文件标头中。
- 删除了每个组件的钩子，转而使用插件替代。
- 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。
- 现在插件确保项目中的所有组件具有相同的翻译。
- 在提交消息模板中支持更多变量。
- 为提供另外的文本语境添加支持。

4.17 Weblate 2.x 系列

4.17.1 Weblate 2.20

发布于 2018 年 04 月 04 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。
- 为下载只需要动作的字符串添加了支持。
- 为立刻在几种语言中搜索添加了支持。
- 现在插件来配置 gettext 输出换行。
- 新的插件来配置 JSON 格式。
- 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。
- 为面板添加了组件概况和每个组件列表概况的页面。
- 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。

4.17.2 Weblate 2.19.1

发布于 2018 年 02 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。

4.17.3 Weblate 2.19

发布于 2018 年 02 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。
- 改进了大字符串的词汇表查询速度。
- 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。
- 在空白字符自动修复中 Honor 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 Translation Hub 服务添加了支持。
- 为 Microsoft Terminology 服务添加了支持。
- 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。
- 改进了不同复数形式的支持。
- 添加了对不使用 stdlayout 的 Subversion 仓库的支持。
- 添加了插件类定制化翻译流程。

4.17.4 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，较少地于浏览器/系统的快捷键冲突。
- 改进了对 XLIFF 文件中的同意标记的支持。
- 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。

- 对专门复核人员的翻译流程添加了支持。

4.17.5 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。

4.17.6 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件地址添加了支持。
- 用户现在可以删除自己的注释。
- 为搜索和替换特性添加了预览步骤。
- 搜索和上传表格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。
- 添加了使用 Akismet 建议的可选垃圾邮件保护。

4.17.7 Weblate 2.16

发布于 2017 年 08 月 11 日。

- 更重新能改进。
- 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 widget 的外观。
- 对于表格中的文本长度现在强制进行最大长度检测。
- 每个组件进行 commit_pending 时间配置。
- 各种用户界面清理。
- 修复了组件/项目/网站范围的翻译搜索。

4.17.8 Weblate 2.15

发布于 2017 年 06 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。
- 当从词汇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。

4.17.9 Weblate 2.14.1

发布于 2017 年 05 月 24 日。

- 修复了将搜索结果分页时可能的错误。
- 修复了一些极端情况下从较老的版本合并的问题。
- 修复了项目有人或无人监守时可能的 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。

4.17.10 Weblate 2.14

发布于 2017 年 05 月 17 日。

- 使用 AJAX 添加词汇表条目。
- 登出现在使用 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动作，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行为。
- 当删除账户或添加新的团体时，询问密码确认。
- 当做出建议时显示时间。
- 新的质量检查来跟踪分号。

- 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。

4.17.11 Weblate 2.13.1

发布于 2017 年 04 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的列表。
- 当用户没有特权而尝试访问项目时返回 HTTP 404。

4.17.12 Weblate 2.13

发布于 2017 年 04 月 12 日。

- 修复了翻译模板的质量检查。
- 添加了质量检查，在丢失翻译时触发。
- 添加选项来查看用户待定的建议。
- 添加选项来自动建立组件列表。
- 未认证用户可以配置的默认面板。
- 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。
- 每个项目的 ACLs 现在使用 Group ACL 来实施。
- 添加了更精细颗粒的特权控制。
- 各种小的 UI 改进。

4.17.13 Weblate 2.12

发布于 2017 年 03 月 03 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。
- 添加了项目和组件范围的搜索。
- 添加了项目和组件范围的搜索与替换。
- 改进了将不一致翻译待定。
- 在本地编辑器中添加了开源文件的支持。
- 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。

- 默认提交消息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。

4.17.14 Weblate 2.11

发布于 2017 年 01 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组件优先性。
- 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 导出插件中与各个 Git 版本更好地兼容。
- 包括每个项目的和组件的统计数据。
- 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。
- 修复了具有单个复数形式的语言的复数源显示。
- 改进了 import_project 中的错误处理。
- 更重新能改进。

4.17.15 Weblate 2.10.1

发布于 2017 年 01 月 20 日。

- 不泄露密码重置表格上存在账户（CVE-2017-5537）。

4.17.16 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。
- 简化了项目和组件用户界面。
- 添加了自动修复来去除控制字符。
- 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，以快速概述所有翻译。
- 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。

4.17.17 Weblate 2.9

发布于 2016 年 11 月 04 日。

- 扩展了 createadmin 管理命令的参数。
- 扩展了 import_json，而能够处理现有组件。
- 为 YAML 文件添加了支持。
- 项目所有者现在可以配置翻译组件和项目细节。
- 使用“Watched”（关注的）而不是“Subscribed”（订阅的）项目。
- 可以从项目页面直接关注项目。
- 添加多语言状态 widget。
- 如果没有显示源的话，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 UX。
- 修复了显示组件的白板信息。
- 存储后保持偏好标签被选择。
- 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。
- 为上传可视内容（屏幕截图）用于翻译添加了支持。

4.17.18 Weblate 2.8

发布于 2016 年 08 月 31 日。

- 文档改进。
- 翻译。
- 更新了捆绑的 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。
- 从 XML 合法性检查中分出不匹配的 XML 标签。
- 修复了 API 来接受 HIDE_REPO_CREDENTIALS 设置。
- 在禅模式中显示源更改。
- Alt+PageUp/PageDown/Home/End 现在也在缠模式中工作良好。
- 添加提示来显示更改的准确时间。
- 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。

4.17.19 Weblate 2.7

发布于 2016 年 07 月 10 日。

- 去除了 Google web translation 机器翻译。
- 改进了添加翻译时的提交消息。
- 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。

4.17.20 Weblate 2.6

发布于 2016 年 04 月 28 日。

- 修复了语言筛选程序的验证组件。
- 改进了对 XLIFF 文件的支持。
- 修复了非英语源的机器翻译。
- 添加了 REST API。
- Django 1.10 的兼容性。
- 为白板纤细添加了分类。

4.17.21 Weblate 2.5

发布于 2016 年 03 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。
- 为生成翻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组件添加支持。
- 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。
- 修正了全站检查失败的问题。

- 添加选型来指定源语言。
- 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方式。
- 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）。
- 明确需要动作的字符串和不需要翻译的字符串的术语。
- 支持 Python 3。
- 去掉了对 Django 1.7 的支持。
- 去掉了用于新建新的 gettext PO 文件的 msginit 依赖性。
- 添加了可配置的面板视图。
- 改进了解析错误的通知。
- 添加了选项将带有复制名称的组件导入 import_project。
- 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。
- 为可配置的自动群组指定添加了支持。
- 改进了新翻译的添加。

4.17.22 Weblate 2.4

发布于 2015 年 09 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 VCS root 作为当前目录执行。
- 钩子脚本以描述当前组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。
- 改进了几种语言的字典匹配。
- 改进了多数页面的布局。
- 支持在翻译时将单词添加入字典。
- 为筛选语言由 Weblate 管理添加了支持。
- 为翻译并导入 CSV 文件添加了支持。
- 重写了静态文件的处理。
- 第三放服务的直接登录/注册链接，如果是唯一一个的话。
- 账户删除时提交待定的更改。
- 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。

4.17.23 Weblate 2.3

发布于 2015 年 05 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。

4.17.24 Weblate 2.2

发布于 2015 年 02 月 19 日。

- 性能改进。
- 在所在地和注释位置的全文检索。
- 新的基于行为图标的 SVG/javascript。
- 支持 Django 1.8。
- 支持删除注释。
- 添加自己的 SVG 徽章。
- 为 Google Analytics 添加支持。
- 改进了翻译文件名的处理。
- 为单语言 JSON 翻译添加了支持。
- 在历史中记录组件锁定。
- 支持为单语言翻译编辑源（模板）语言。
- 为 Gerrti 添加了基本支持。

4.17.25 Weblate 2.1

发布于 2014 年 12 月 05 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。
- 允许从面板上隐藏完成的翻译。

4.17.26 Weblate 2.0

发布于 2014 年 11 月 06 日。

- 使用 Bootsctrap 的新的仓库 UI。
- 重写了 VCS 后端。
- 文档改进。
- 为网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。
- 为翻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。
- 为每种语言引用和其它特殊字符添加了按钮。
- 支持导出统计数据作为 JSONP。

4.18 Weblate 1.x 系列

4.18.1 Weblate 1.9

发布于 2014 年 05 月 06 日。

- Django 1.6 兼容性。
- 不在维护与 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。
- 合并第一个和最后一个名称属性。
- 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。

4.18.2 Weblate 1.8

发布于 2013 年 11 月 07 日。

- 关于升级的指示，请查看手册。
- 更好地列出了项目概况。
- 用于分享的更好的可视化选型。
- 对匿名用户特权的更多控制。
- 支持使用第三方服务登录，更多细节请查看手册。
- 用户可以通过电子邮件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。

4.18.3 Weblate 1.7

发布于 2013 年 10 月 07 日。

- 关于升级的指示，请查看手册。
- 支持检查 Python brace 格式字符传。
- 每个组件的质量检查定制。
- 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消息。
- 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 来提供 Widget 和图表。
- 添加状态徽章 Widget。
- 去掉了不合法的文本方向检查。
- 字典中的更改现在在历史中记入日志。
- 翻译视图的性能改进。

4.18.4 Weblate 1.6

发布于 2013 年 07 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。
- 各种表格验证改进。
- 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。

4.18.5 Weblate 1.5

发布于 2013 年 04 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。
- 为词汇表的 TBX 导出添加了支持。
- 为 Bitbucket 通知添加了支持。
- 行为图标现在可用于每个翻译、语言或用户。
- 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。
- 更重新能改进。
- 对项目或同样对语言，在进度条中指示失败的检查或模糊字符串。
- 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。

4.18.6 Weblate 1.4

发布于 2013 年 01 月 23 日。

- 修复了在字符串删除时删除检查/注释。
- 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。
- 正确地导入需要客户 ttkit 导入程序的项目。
- 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 widgets 和接合页面。
- 改进了翻译锁定处理。
- 为 widgets 以更多变量显示代码段。
- 在进度挑中指示失败的检测或模糊字符串。
- 用于将提交信息格式化的更多选项。
- 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。

4.18.7 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（BB 代码、XML 标记和新的行）。
- 支持可选地 rebasing 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 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注释。
- 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。

4.18.8 Weblate 1.2

发布于 2012 年 08 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。
- 现在将人员与翻译建立联系的简介页面使用 Weblate。
- 添加了小程序，可以用于推动翻译项目。
- 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组件。
- 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。
- 在配置编辑器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。
- 支持翻译者锁定翻译。
- 可选地维护 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。
- 缓存检查失败的字符串的计数。
- 在设置中检查要求。
- 文档改进。

4.18.9 Weblate 1.1

发布于 2012 年 07 月 04 日。

- 改进了几个翻译。
- 新建组件时更好的验证。
- 为在组件之间分享 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。

4.18.10 Weblate 1.0

发布于 2012 年 05 月 10 日。

- 改进了添加/存储组件时的验证。
- 试验支持安卓组件文件（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。

4.19 Weblate 0.x 系列

4.19.1 Weblate 0.9

发布于 2012 年 04 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。

- 为更多的几种语言添加了定义。
- 各种编码清理。
- 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。
- 新建组件时更好的验证。
- 将完全设置集成到 syncdb 中。
- 为所有翻译页面添加了最近更改的列表。
- 检查未翻译的字符串，忽略格式字符串只检查消息。

4.19.2 Weblate 0.8

发布于 2012 年 04 月 03 日。

- 用 Whoosh 代替自己的全文搜索。
- 对检查的各种修改和改进。
- 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/。
- 机器翻译页面不再阻碍装入页面。
- 管理界面现在也包含有用的行为来更新数据。
- 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。
- 浏览失败检查的可能性。
- 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。

4.19.3 Weblate 0.7

发布于 2012 年 02 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。

4.19.4 Weblate 0.6

发布于 2012 年 02 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格的排序。
- 在升级时请检查升级指示。

4.19.5 Weblate 0.5

发布于 2012 年 02 月 12 日。

- **通过使用后面的服务对机器翻译的支持：**
  - Apertium
  - 微软的 Translator
  - 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。

4.19.6 Weblate 0.4

发布于 2012 年 02 月 08 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。

4.19.7 Weblate 0.3

发布于 2012 年 02 月 08 日。

- 对于复数翻译更海底显示源。
- 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。

4.19.8 Weblate 0.2

发布于 2012 年 02 月 07 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。

4.19.9 Weblate 0.1

发布于 2012 年 02 月 06 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/glossary

/api/groups

/api/languages

/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

G

H

I

J

L

M

T

U

V

W

X

Y

◆

◆