国际化与本地化

国际化与本地化
1. Flask-Babel扩展库使用
    1.1. 扩展库安装
    1.2. 扩展库注册
    1.3. 配置支持语言列表及默认语言
    1.4. 编写获取本地语言函数
2. 原代码改造
    2.1. python代码改造
    2.2. 模板代码改造
    2.3. 提取需翻译文本
    2.4. 生成语言目录
    2.5. 编译翻译文件
    2.6. 测试翻译效果
    2.7. 翻译文件更新
    2.8. 翻译日期时间
3. 翻译命令行增强
    3.1. 创建翻译命令组
    3.2. 创建初始化子命令
    3.3. 创建更新和编译子命令
    3.4. 注册翻译命令组
    3.5. 测试注册的命令行

1. Flask-Babel扩展库使用

对于该项目可能会面向于多地区用户，因此对于国际化和本地化（缩写为I18n和L10n）的处理是很有必要的，可以通过`flask-babel`扩展库来实现。

1.1. 扩展库安装

```
pip install flask-babel
```

1.2. 扩展库注册

修改`app/__init__.py`脚本，进行`flask-babel`扩展库的注册。

```
......
from flask_babel import Babel
......
# 实例化flask_babel
babel = Babel()


def create_app(test_config=None):
......
    # 初始化flask_babel
    babel.init_app(application)
```

1.3. 配置支持语言列表及默认语言

修改`app/config.py`脚本，增加支持语言列表及默认语言等配置参数。其中，默认本地语言中文应设置为`zh_CN`。

```
# ----------国际化、本地化配置------- #
# 支持语言列表
LANGUAGES = ['en', 'zh_CN']
# 默认本地语言
BABEL_DEFAULT_LOCALE = 'zh_CN'
```

1.4. 编写获取本地语言函数

修改`app/__init__.py`脚本，增加获取本地语言函数，`Babel`实例提供了一个`localeselector`装饰器。通过该装饰器进行修饰可以实现客户端本地语言。Flask中`request`对象提供了一个高级接口，用于处理客户端发送的带Accept-Language头部的请求。 该头部指定了客户端语言和区域设置首选项。 该头部的内容可以在浏览器的首选项页面中配置，默认情况下通常从计算机操作系统的语言设置中导入。

```
from flask import request
......
def create_app(test_config=None):
......
    @babel.localeselector
    def get_locale():
        return request.accept_languages.best_match(application.config['LANGUAGES'])

    return application
```

2. 原代码改造

2.1. python代码改造

对于python代码中国际化与本地化改造，可以引入`from flask_babel import _`，通过函数`_()`对需要翻译的字符串进行封装。

以`app/login.py`脚本为例，对于`flash()`函数以及`render_template`中对`title`赋的值，均可以通过`_()`函数进行处理：

```
from flask_babel import _


class LoginView(View):
    methods = ['GET', 'POST']

    def dispatch_request(self):
......
            if not user or not user.check_password(form.password.data):
                flash(_('用户名或密码有误'))
                return redirect(url_for('login'))
......
        # GET请求，直接展示登录页面
        return render_template('login/login.html', title=_('登录'), form=form)
```

对于字符串拼接传值方式的封装，需要进行特殊改造。`'用户{}未找到'.format(username)`此种代码需要修改为`_('用户%(username)s未找到', username=username)`。

以`app/user.py`脚本为例：

```
class FollowView(View):
......
        if not user:
            flash(_('用户%(username)s未找到', username=username))
            return redirect(url_for('index'))
        elif user == current_user:
            flash(_('不能关注自己！'))
            return redirect(url_for('user.user_info', username=username))
        current_user.follow(user)
        db.session.commit()
        flash(_('您已关注%(username)s！', username=username))
        return redirect(url_for('user.user_info', username=username))
```

有些字符串文字并非是在发生请求时分配的，而是在应用启动时。因此在评估这些文本时，无法知道要使用哪种语言。 一个例子是与表单字段相关的标签，处理这些文本的唯一解决方案是找到一种方法来延迟对字符串的评估，直到它被使用，比如有实际上的请求发生了。 Flask-Babel提供了一个称为`lazy_gettext()`的`_()`函数的*延迟评估*的版本。

以`app/forms.py`脚本为例：

```
from flask_babel import lazy_gettext as _l


class LoginForm(FlaskForm):
    """登录表单"""
    # DataRequired：数据不可为空的验证器
    username = StringField(_l('用户名'), validators=[DataRequired()])
    password = PasswordField(_l('密码'), validators=[DataRequired()])
    remember_me = BooleanField(_l('记住我'), default=False)
    submit = SubmitField(_l('登录'))
```

2.2. 模板代码改造

对于python代码中国际化与本地化改造，可以直接在模板文件中使用`_()`函数。

以`app/templates/login/login/html`文件为例：

```
{% extends 'base.html' %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block app_content %}
    <h1>{{ _('登录') }}</h1>
    <div class="row">
        <div class="col-md-4">
            {{ wtf.quick_form(form) }}
        </div>
    </div>
    <br>
    <p>{{ _('新用户? ') }}<a href="{{ url_for('register') }}">{{ _('点击注册! ') }}</a></p>
    <p>{{ _('忘记密码? ') }}<a href="{{ url_for('reset_password_request') }}">{{ _('重置密码')
}}</a></p>
{% endblock %}
```

也可以使用参数传入方式，以`app/templates/index/index.html`为例：

```
......
    <nav aria-label="...">
        <ul class="pager">
            <li class="previous{% if not prev_url %} disabled {% endif %}">
                <a href="{{ prev_url or '#' }}">
                    <span aria-hidden="true">&larr;</span> {{ _('上一页') }}
                </a>
            </li>
            <li>{{ _('第%(page)s页', page=page) }}</li>
            <li class="next{% if not next_url %} disabled {% endif %}">
                <a href="{{ next_url or '#' }}">
                     {{ _('下一页') }} <span aria-hidden="true">&rarr;</span>
                </a>
            </li>
        </ul>
    </nav>
{% endblock %}
```

对于`app/templates/common/post.html`文件中，帖子内容展示改造如下：

原代码：

```
    <table class="table table-hover">
......
            <td>
                <a href="{{ url_for('user.user_info', username=post.author.username) }}">
                    {{ post.author.username }}
                </a>
                说 {{ moment(post.timestamp).fromNow() }}:
                <br>
                {{ post.body }}
            </td>
        </tr>
    </table>
```

改造后代码：

```
    <table class="table table-hover">
......
            <td>
                {% set user_link %}
                    <a href="{{ url_for('user.user_info', username=post.author.username)
}}">
                        {{ post.author.username }}
                    </a>
                {% endset %}
                {{ _('%(username)s 说 %(when)s: ', username=user_link,
when=moment(post.timestamp).fromNow()) }}
                <br>
                {{ post.body }}
            </td>
        </tr>
    </table>
```

2.3. 提取需翻译文本

新增 `babel.cfg` 配置文件。前三行分别定义了Python和Jinja2模板文件的文件名匹配模式。 第四行定义了Jinja2模板引擎提供的两个扩展，以帮助Flask-Babel正确解析模板文件。

```
[python: app/**.py]
[jinja2: app/templates/**.html]
[jinja2: app/templates/**.txt]
extensions=jinja2.ext.autoescape,jinja2.ext.with_
```

使用以下命令来将所有文本提取到* .pot *文件：

```
(venv) D:\Projects\learn\flask-mega-tutorial>pybabel extract -F babel.cfg -k _l -o
messages.pot .
extracting messages from app\__init__.py
extracting messages from app\config.py
extracting messages from app\email.py
extracting messages from app\forms.py
extracting messages from app\hello.py
```

```
extracting messages from app\index.py
extracting messages from app\logger.py
extracting messages from app\login.py
extracting messages from app\models.py
extracting messages from app\user.py
extracting messages from app\templates\base.html
extracting messages from app\templates\common\post.html
extracting messages from app\templates\email\reset_password.html
extracting messages from app\templates\email\reset_password.txt
(extensions="jinja2.ext.autoescape,jinja2.ext.with_")
extracting messages from app\templates\error\404.html
extracting messages from app\templates\error\500.html
extracting messages from app\templates\hello\hello.html
extracting messages from app\templates\index\index.html
extracting messages from app\templates\login\login.html
extracting messages from app\templates\login\register.html
extracting messages from app\templates\login\reset_password.html
extracting messages from app\templates\login\reset_password_request.html
extracting messages from app\templates\user\user_info.html
extracting messages from app\templates\user\user_info_edit.html
writing PO template file to messages.pot
```

`pybabel extract` 命令读取 `-F` 选项中给出的配置文件，然后从命令给出的目录（当前目录或本处的 `.`）扫描与配置的源匹配的目录中的所有代码和模板文件。 默认情况下，`pybabel` 将查找 `_()` 以作为文本标记，但我也使用了重命名为 `_l()` 的延迟版本，所以我需要用 `-k _l` 来告诉该工具也要查找它 。 `-o` 选项提供输出文件的名称。

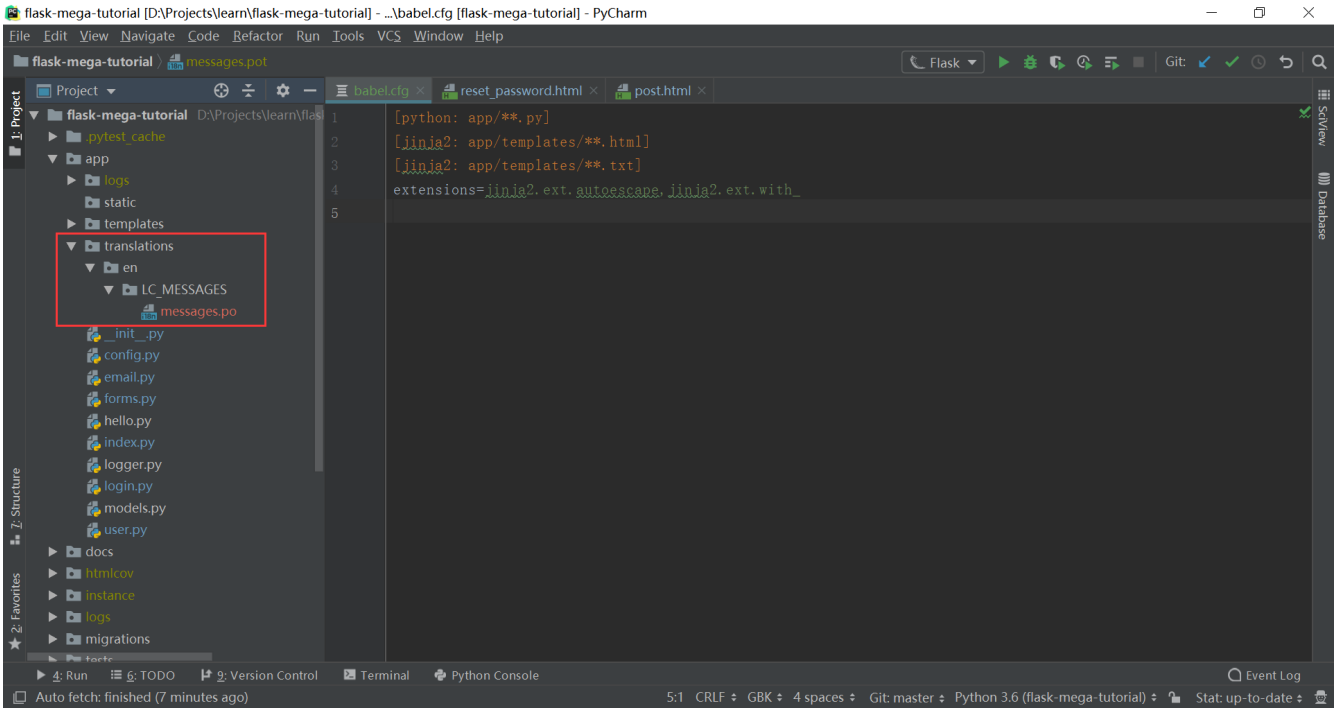
应该注意，生成的*messages.pot*文件不需要合并到项目中。 这是一个只要再次运行上面的命令，就可以在需要时轻松地重新生成的文件。 因此，不需要将该文件提交到源代码管理。

2.4. 生成语言目录

该过程的下一步是在除了原始语言（在本例中为中文）之外，为每种语言创建一份翻译。 现在先从添加英语（语言代码 `en`）开始，所以这样做的命令是：

```
(venv) D:\Projects\learn\flask-mega-tutorial>pybabel init -i messages.pot -d
app/translations -l en
creating catalog app/translations\en\LC_MESSAGES\messages.po based on messages.pot
```

`pybabel init` 命令将*messages.pot*文件作为输入，并将语言目录写入 `-d` 选项中指定的目录中，`-l` 选项中指定的是翻译语言。 我将在*app/translations*目录中安装所有翻译，因为这是Flask-Babel默认提取翻译文件的地方。 该命令将在该目录内为英语数据文件创建一个*en*子目录。 特别是，将会有一个名为*app/translations/en/LC_MESSAGES/messages.po*的新文件，是需要翻译的文件路径。



在每个语言存储库中创建的`messages.po`文件使用的格式是语言翻译的事实标准，使用的格式为gettext。修改`messages.po`文件，设置翻译内容，`msgid`行包含原始语言的文本，后面的`msgstr`行包含一个空字符串。 这些空字符串需要被编辑，以使目标语言中的文本内容被填充。

```
# English translations for PROJECT.
# Copyright (C) 2019 ORGANIZATION
# This file is distributed under the same license as the PROJECT project.
# FIRST AUTHOR <EMAIL@ADDRESS>, 2019.
#
msgid ""
msgstr ""
"Project-Id-Version: PROJECT VERSION\n"
"Report-Msgid-Bugs-To: EMAIL@ADDRESS\n"
"POT-Creation-Date: 2019-05-07 11:27+0800\n"
"PO-Revision-Date: 2019-05-07 13:09+0800\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language: en\n"
"Language-Team: en <LL@li.org>\n"
"Plural-Forms: nplurals=2; plural=(n != 1)\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Generated-By: Babel 2.6.0\n"

#: app/__init__.py:61
msgid "请登录后访问此页面"
msgstr "Please log in to access this page."

#: app/email.py:43
msgid "[博客] 重置您的密码"
msgstr "[Microblog] Reset Your Password"
```

```
#: app/forms.py:19 app/forms.py:27 app/forms.py:48
msgid "用户名"
msgstr "Username"
......
```

2.5. 编译翻译文件

*messages.po*文件是一种用于翻译的源文件。 若想使用这些翻译后的文本，那这个文件就需要被编译成一种格式，这种格式在运行时可以被应用程序使用。 要编译应用程序的所有翻译，可以使用`pybabel compile`命令。

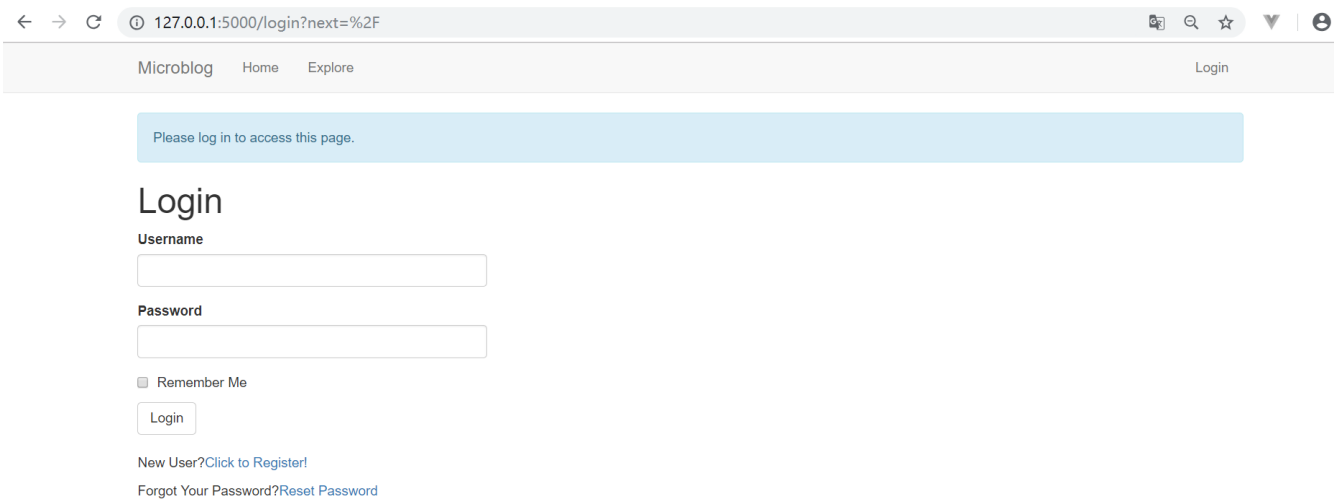
此操作在每个语言存储库中的*messages.po*旁边添加*messages.mo*文件。 *.mo*文件是Flask-Babel将用于为应用程序加载翻译的文件。

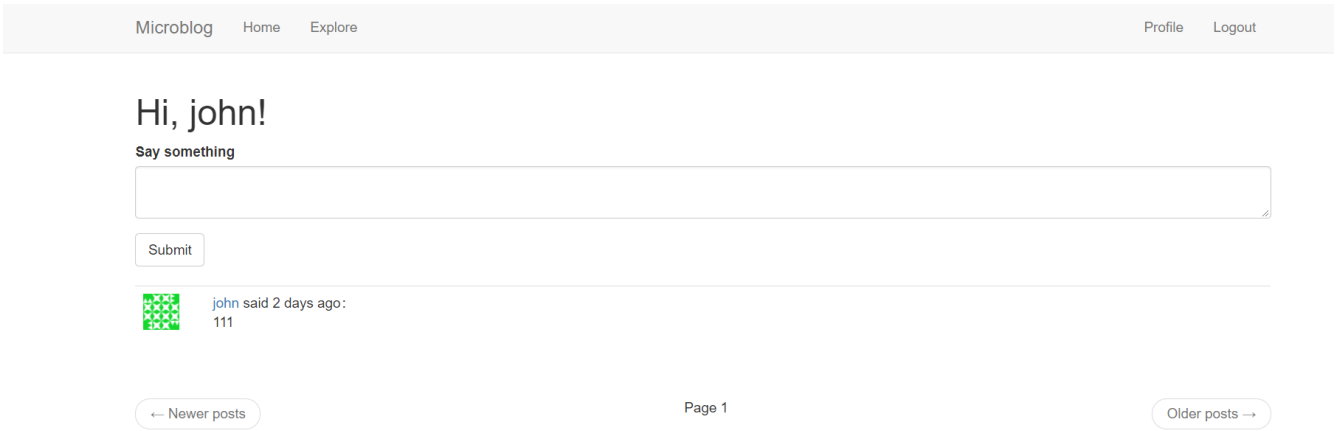
```
(venv) D:\Projects\learn\flask-mega-tutorial>pybabel compile -d app/translations
compiling catalog app/translations\en\LC_MESSAGES\messages.po to
app/translations\en\LC_MESSAGES\messages.mo
```

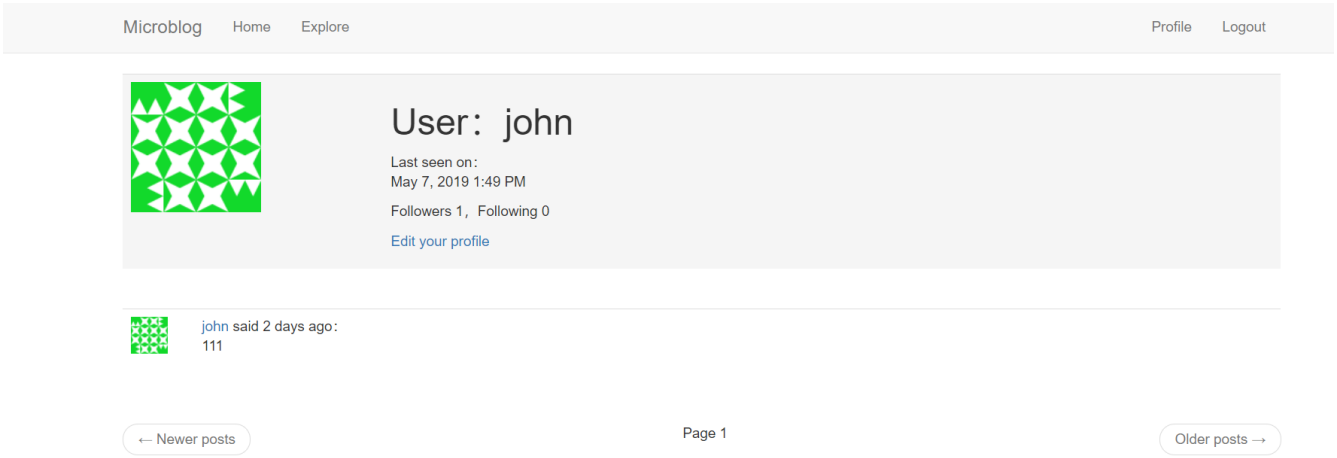
2.6. 测试翻译效果

修改`app/__init__.py`文件，设置本地语言返回固定的en，进行测试。

```
    @babel.localeselector
    def get_locale():
        # return request.accept_languages.best_match(application.config['LANGUAGES'])
        return 'en'
```







2.7. 翻译文件更新

在翻译后，可能会发现有部分遗漏，需要进行再次将其用`_()`或`_l()`包装，然后执行更新过程，这包括两个步骤：

```
(venv) $ pybabel extract -F babel.cfg -k _l -o messages.pot .
(venv) $ pybabel update -i messages.pot -d app/translations
```

`extract`命令与我之前执行的命令相同，但现在它会生成*messages.pot*的新版本，其中包含所有以前的文本以及最近用`_()`或`_l()`包装的文本。

`update`调用采用新的`messages.pot`文件并将其合并到与项目相关的所有*messages.po*文件中。 这将是一个智能合并，其中任何现有的文本将被单独保留，而只有在*messages.pot*中添加或删除的条目才会受到影响。

2.8. 翻译日期时间

目前，在中文语言环境下，所有的日期时间全部显示为英文方式，所以需要设置其当前的语言环境。首先修改`app/__init__.py`脚本，修改`before_request()`函数，使得在请求时，将本地的语言环境放置到`g`对象中，以便可以在模板中直接获取。

由于`get_locale()`函数会返回一个对象，其中，`language`属性为语言，中文即`zh`；`territory`属性为地域，中文即`CN`；`script`属性为脚本代码，中文即`Hans`。若直接通过`str(get_locale())`的方式，获取的结果为`zh_Hans_CN`，但是`flask-moment`扩展库需要的格式为`zh-CN`，所以以下做了拼接处理。

```
......
    @application.before_request
    def before_request():
        """请求前周期函数"""
        # 用户已登录则登记用户请求时间
        if current_user.is_authenticated:
            current_user.last_seen = datetime.datetime.utcnow()
            db.session.commit()

        # 设置本地语言环境参数
        from flask_babel import get_locale
        locale = get_locale()
        language = locale.language + '-' + locale.territory if locale.territory else
locale.language
        g.locale = language
```

修改`app/templates/base.html`文件，为`moment.js`设置本地语言。

```
......
{% block scripts %}
    {{ super() }}
    {{ moment.include_moment() }}
    {{ moment.lang(g.locale) }}
{% endblock %}
```

翻译效果如下：

3. 翻译命令行增强

由于`pybabel`命令行太长，比较难记，因此创建一些命令行，并用特定的参数触发特定的`pybabel`命令。主要实现的命令如下：

- `flask translate init LANG`用于添加新语言
- `flask translate update`用于更新所有语言存储库
- `flask translate compile`用于编译所有语言存储库

3.1. 创建翻译命令组

新建`app/cli.py`脚本，用于创建翻译命令行，并在文件中创建翻译命令组。

```
def register(app):
    """翻译命令行注册"""
    @app.cli.group()
    def translate():
        """翻译命令组"""
        pass
```

3.2. 创建初始化子命令

修改`app/cli.py`脚本，新增初始化命令。该命令使用`@click.argument`装饰器来定义语言代码。 Click将命令中提供的值作为参数传递给处理函数，然后将该参数并入到`init`命令中。

```
import os
import click


def register(app):
    """翻译命令行注册"""
    @app.cli.group()
    def translate():
        """翻译命令组"""
        pass

    @translate.command()
    @click.argument('lang')
    def init(lang):
        """
        初始化新语言
        :param lang: 语言代码
        :return:
        """
        if os.system('pybabel extract -F babel.cfg -k _l -o messages.pot .'):
            raise RuntimeError('extract命令失败')
        if os.system('pybabel init -i messages.pot -d app/translations -l ' + lang):
            raise RuntimeError('init命令失败')
        os.remove('messages.pot')
```

3.3. 创建更新和编译子命令

修改`app/cli.py`脚本，新增更新子命令、编译子命令。

```
import os


def register(app):
    """翻译命令行注册"""
    @app.cli.group()
    def translate():
        """翻译命令组"""
```

```
        pass

    @translate.command()
    def update():
        """更新子命令"""
        if os.system('pybabel extract -F babel.cfg -k _l -o messages.pot .'):
            raise RuntimeError('extract命令失败')
        if os.system('pybabel update -i messages.pot -d app/translations'):
            raise RuntimeError('update命令失败')
        os.remove('messages.pot')

    @translate.command()
    def compile():
        """编译子命令"""
        if os.system('pybabel compile -d app/translations'):
            raise RuntimeError('compile命令失败')
```

3.4. 注册翻译命令组

修改`app/__init__.py`脚本，将翻译命令组注册到应用中。

```
def create_app(test_config=None):
......
    # 翻译命令组注册
    from app.cli import register
    register(application)
```

3.5. 测试注册的命令行

查看命令行说明

```
(venv) D:\Projects\learn\flask-mega-tutorial>flask translate --help
[2019-05-07 15:48:51,003] INFO in logger: 微博已启动
Usage: flask translate [OPTIONS] COMMAND [ARGS]...

  翻译命令组

Options:
  --help  Show this message and exit.

Commands:
  compile  编译子命令
  init     初始化新语言 :param lang: 语言代码 :return:
  update   更新子命令
```

执行初始化命令

```
(venv) D:\Projects\learn\flask-mega-tutorial>flask translate init zh_CN
[2019-05-07 15:50:55,306] INFO in logger: 博客已启动
extracting messages from app\__init__.py
```

```
extracting messages from app\cli.py
extracting messages from app\config.py
extracting messages from app\email.py
extracting messages from app\forms.py
extracting messages from app\hello.py
extracting messages from app\index.py
extracting messages from app\logger.py
extracting messages from app\login.py
extracting messages from app\models.py
extracting messages from app\user.py
extracting messages from app\templates\base.html
extracting messages from app\templates\common\post.html
extracting messages from app\templates\email\reset_password.html
extracting messages from app\templates\email\reset_password.txt
(extensions="jinja2.ext.autoescape,jinja2.ext.with_")
extracting messages from app\templates\error\404.html
extracting messages from app\templates\error\500.html
extracting messages from app\templates\hello\hello.html
extracting messages from app\templates\index\index.html
extracting messages from app\templates\login\login.html
extracting messages from app\templates\login\register.html
extracting messages from app\templates\login\reset_password.html
extracting messages from app\templates\login\reset_password_request.html
extracting messages from app\templates\user\user_info.html
extracting messages from app\templates\user\user_info_edit.html
writing PO template file to messages.pot
creating catalog app/translations\zh_CN\LC_MESSAGES\messages.po based on messages.pot
```