Symfony2框架实战教程——第六天：模板重载与翻译

2条回复

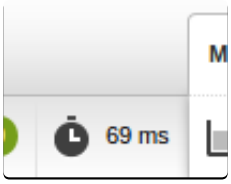
昨天我们已经完成了首页和新闻列表页的外观改造，剩下的新闻详情页，就可以留给大家当作业自己实践了。今天我们要改造的是登录页。

在未登录状态下点击“+发表新闻”按钮，也就是`/news/new`链接，会转跳到`/oauth/login/`链接，也就是第三方登录的链接（如果只实现了本地用户登录的同学，去的应该是另外一个界面，不过没关系，重载模板的原理都一样）。目前第三方登录页面只有一个可怜巴巴的“QQ”这个链接，让我们也给它加上页头页尾。

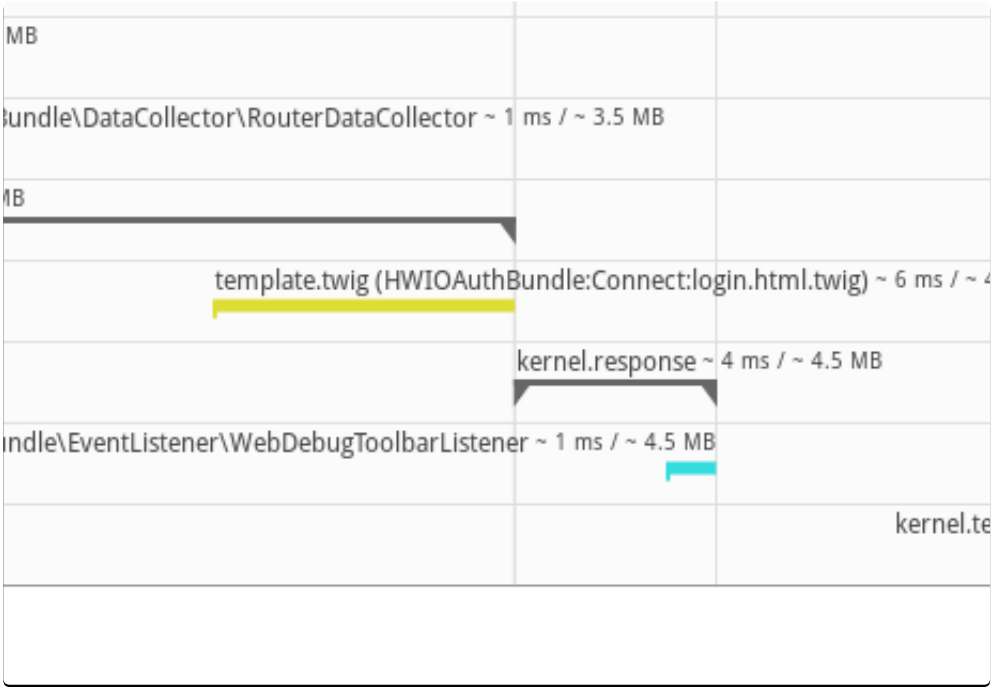
不过，这个模板文件很明显不是我们自己添加的，是第三方Bundle里自带的，难道我们要去修改第三方Bundle的代码吗？库文件随便修改这样好吗？当然不好，好消息是Symfony2框架实际是提供了框架模板重载机制的。

模板重载

在重载之前，我们需要知道模板的路径。那么如何找路径呢？很简单，只要先找到对应的路径对应的路由，然后找到对应的控制器，然后通过控制器代码就能知道加载的模板是……你以为我会跟你说这么傻的方法吗？其实你只需要点一下调试工具栏的这个像秒表的按钮：



是否看到某个`.html.twig`文件了呢？（友情提示：万一你电脑超级牛逼渲染模板连1ms都不到，是看不到模板渲染时间的，这个时候你可以将页面上方的`Threshold`改成`0.1`甚至更小）



得知模板的名称为`HWIOAuthBundle:Connect:login.html.twig`，我们就可以据此创建重载文件`app/Resources/HWIOAuthBundle/views/Connect/login.html.twig`。创建好了之后刷新一下登录页，因为我们什么都没写，如果重载成功，会返回一个空白页面。注意文件的命名规律，必须按此规律才能实现在app下的第三方模板重载

我们可以参考一下本身的模板代码，然后加上我们自己的`base.html.twig`的布局模板：

```
1  {# app/Resources/HWIOAuthBundle/views/Connect/login.html.twig %}
2  {% extends 'base.html.twig' %}
3
4  {% block body %}
5      {% if error is defined and error %}
6          <span>{{ error }}</span>
7      {% endif %}
8      {% for owner in hwi_oauth_resource_owners() %}
9          <a href="{{ hwi_oauth_login_url(owner) }}">{{ owner | tran
10     {% endfor %}
11 {% endblock body %}
12
```

光有一个链接其实依然很丑，我想把他做成一个带企鹅图标的按钮。我虽然不是设计师，但还好有开源素材给我用。fontawesome便是这么一个开源的图标库。它是以字体的方式存在的，这就意味着你可以不需要设计师做图，而是用CSS的方式让这些“图片”像字一样，随意变大，缩小，改变颜色，设置半透明效果，添加动

画……只要CSS能做到的都能实现，而且因为字体是矢量的，所以放得再大也不会失真。

现在把fontawesome引入到项目中来：

```
1 {# app/Resources/views/base.html.twig #}
2 ...
3 <link rel="stylesheet" href="//cdn.bootcss.com/font-awesome/4.3.0/c
4 ...
5
```

然后将之前的链接替换成创建按钮：

```
1 {# app/Resources/HWIOAuthBundle/views/Connect/login.html.twig #}
2 ...
3 <a class="btn btn-default" href="{{ hwi_oauth_login_url(owner) }}">
4 ...
5
```

当然，我们可以顺手把“qq”变成大写的（使用twig的upperfilter，像这样`{{ owner | tarns({}, 'HWIOAuthBundle') | upper }}`），不过这么做并不好，因为要是以后添加了微博，那按钮岂不会变成`WEIBO`。

翻译

其实HWIOAuth已经帮我们想到了这一点，所以用了一个叫`trans`的Twig filter。`trans`即translate即翻译的意思，我们如何实现翻译呢？首先我们需要一个文件，来定义翻译源和翻译结果的对应关系。比如目前我们的翻译源是`qq`，翻译结果是……为了效果明显点，就打算叫腾讯`QQ`吧。这样`trans`就会读取这个文件，如果发现翻译源是一个叫`qq`的字符串，他就会帮我们用腾讯`qq`来替换它（注意必须是只有`qq`的字符串，`this is qq`这样的字符串里的`qq`是不会被替换的）。

虽然翻译文件也可以有各种格式，比如最省事儿的YAML格式来做翻译，不过翻译也已经有业界标准，通常都是使用xliff格式。这个文件的存放位置，如果是第三方Bundle本身自带的翻译文件，放在`XxxBundle/Resources/translations/yyy.zh_CN.xliff`，但如果是项目自身的翻译，按官方的最佳实践的说法，放在`app/Resources/translations/yyy.zh_CN.xliff`。至于这个`yyy`具体是什么文件名，得看`trans`的第二个参数，比如这里是`HWIOAuthBundle`，那翻译文件名就是'HWIOAuthBundle.zh_CN.xliff'。如

果没有第二个参数，默认是'messages'，对应文件名就是messages.zh_CN.xliff。

使用xliff或者说用一种业界普遍使用的格式有一个最大的好处：相关的辅助软件多如牛毛，随便网上一搜就是一大把，不过这里我们先不用其他的软件来做翻译。

另外，文件名中间的zh_CN表示的是“中文-大陆”，也就是简体中文。语言代码一般由语言+区域的方式构成，比如en_US和en_GB，但如果有些语句不严格区分区域的话，区域代码也是可以省略的，比如en。Symfony2翻译的顺序是先找“语言+区域”文件，再查找“仅语言”文件，再查找默认语言文件，至于默认语言文件如何设置，后面会提到。小提示：以后千万记得中文应该是zh而不是cn，cn指的是中国，而不是中文。

那么我们就来创建翻译文件吧，且慢！像翻译这种常见的需求……好吧不多说，接下来我们将用到简化翻译的Bundle：JMSTranslationBundle

一如既往安装Bundle

```
1 $ composer require jms/translation-bundle
2
```

注册Bundle

```
1 // in AppKernel::registerBundles()
2 $bundles = array(
3     // ...
4     new JMS\TranslationBundle\JMSTranslationBundle(),
5     // ...
6 );
7
```

然后我们就可以让JMSTranslationBundle自动帮我们创建xliff文件了

```
1 $ php app/console translation:extract zh_CN --dir=app/Resources --c
2
```

好嘞，让我们去看看生成文件：打开app/Resources/translations/……说好的xliff文件呢？为啥连根毛都没有啊？因为，模板文件里被翻译的是一个变量（那个owner），JMSTranslationBundle依然不知道翻译源是啥。

为了让JMSTranslationBundle知道有一待翻译短语叫qq，那我们就手工创建一个好了！（好吧其实我也是为了演示JMSTranslationBundle的用法，文件完全可以自己创建，不过最好还是不要手写xliff，一是内容多，写起来麻烦，二是手写容易出错）。在app/Resources/views下的随便什么模板文件，在某个block里输入下面的内容：

```
1 {{ 'qq'|trans({}, 'HWIOAuthBundle') }}
2
```

然后再重新跑一次命令，我们便可以得到HWIOAuthBundle.zh_CN.xliff文件了：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xliff xmlns="urn:oasis:names:tc:xliff:document:1.2" xmlns:jms="ur
3    <file date="2015-04-01T01:13:48Z" source-language="en" target-la
4      <header>
5        <tool tool-id="JMSTranslationBundle" tool-name="JMSTranslati
6        <note>The source node in most cases contains the sample mess
7      </header>
8      <body>
9        <trans-unit id="bed4eb698c6eeea7f1ddf5397d480d3f2c0fb938" re
10         <jms:reference-file line="9">views/Connect/login.html.twig
11         <source>qq</source>
12         <target state="new">qq</target>
13       </trans-unit>
14     </body>
15   </file>
16 </xliff>
17
```

到此，之前为了生成文件而随便添加的代码可以删除了。

这个文件如果不看xliff相关文档，你敢随便改吗？没关系，JMSTranslationBundle还有另外一个给力的功能：带Web UI的翻译工具。

因为在生产环境我们是不可能向公众开放翻译工具的，所以以下配置应该加到routing_dev.yml里：

```
1 JMSTranslationBundle_ui:
2     resource: @JMSTranslationBundle/Controller/
3     type:     annotation
```

```
4     prefix:   /_trans
5
```

因为JMSTranslationBundle的WebUI使用了JMSDiExtra以及TwigExtensions两个库，我们也把它们加载进来：

```
1 $ composer require 'jms/di-extra-bundle'
2 $ composer require 'twig/extensions'
3
```

```
1 // in AppKernel::registerBundles()
2 $bundles = array(
3     // ...
4     new JMS\AopBundle\JMSAopBundle(),
5     new JMS\DiExtraBundle\JMSDiExtraBundle($this),
6     // ...
7 );
8
```

注意：因为TwigExtensions库不是Bundle，所以不需要进行注册工作。
另外JMSAopBundle是JMSDiExtraBundle需要的Bundle，虽然在安装过程中已经作为JMSDiExtraBundle的依赖自动被安装，但是注册还是需要我们手动的。

这个时候访问/_trans路径，还是会报错的，因为我们还有一些必要的设置没做，首先是JMSDiExtraBundle的设置，这个Bundle作用其实就一个：可以用annotation的方式来注册Service。其实还是挺有用的，大家如果有兴趣可以看看JMSTranslationBundle的TranslateController，看看它是如何用annotation的方式来注入一些服务到controller里的。其他不多说，先写配置：

```
1 jms_di_extra:
2     locations:
3         all_bundles: false
4         bundles: [JMSTranslationBundle]
5
```

另JMSTranslationBundle也需要通过配置文件得知一些信息，我们来将其配置上，比如我们要翻译成什么语言，在哪个路径里找翻译源，在哪个路径找翻译文件

```
1 jms_translation:
```

```
2     configs:
3         app:
4             dirs: ['%kernel.root_dir%/Resources', '%kernel.root_dir
5             output_dir: '%kernel.root_dir%/Resources/translations'
6
```

设置好所有配置后，再次访问`/_trans`，虽然还有报错，但是报错不一样了。这一次是twig的错误，说是找不到`truncate`的twig filter。那我们如何去注册一个twig filter呢？

注册自定义**Twig Filter**

在定义Symfony的服务的时候，有些服务可能在Symfony初始化的时候，就需要被运行，而不是普通服务那样，调用它的时候才运行，比如Twig的Filter服务，他需要在创建好服务对象之后，立马被Twig Enviroment注册，这样才能在twig模板中使用定义好的filter。而要实现此种目的，Symfony的服务容器是通过给服务定义tag来实现的。比如我们接下来要做的配置：

```
1 # app/config/services.yml
2 services:
3     twig.text_extension:
4         class: Twig_Extensions_Extension_Text
5         tags:
6             - name: twig.extension
7
```

注意到我们加了一个名字为`twig.extension`的tag，只要有此tag的服务，Symfony就会在服务创建好之后，立马将其注册到Twig Environment中。另外类似的还有Listener(tag名字为`kernel.event_listener`)等，只要初始化好服务，立马就被Event Dispatcher注册。

另外如果有兴趣，也可以看看Twig_Extension_Extension_Text是怎么定义Filter的。主要是`getFilters`里的`new Twig_SimpleFilter('truncate', 'twig_truncate_filter', array('needs_environment' => true))` 那句，其中`Twig_SimpleFilter构造函数的第二个参数可以是任意的callback，而callback的第一个参数，就是我们需要处理的字符串

回到正题，加上上面的配置之后，访问`/_trans`，就不会报错了，只不过样式根本就没有，一个连样式都没有功能还好意思发布成，你们信吗？当然不信。JMSTranslationBundle是自带了样式的，其实还包括JS文件。我们需要做的是将第三方Bundle中的这些静态文件，链接到`web`目录下：

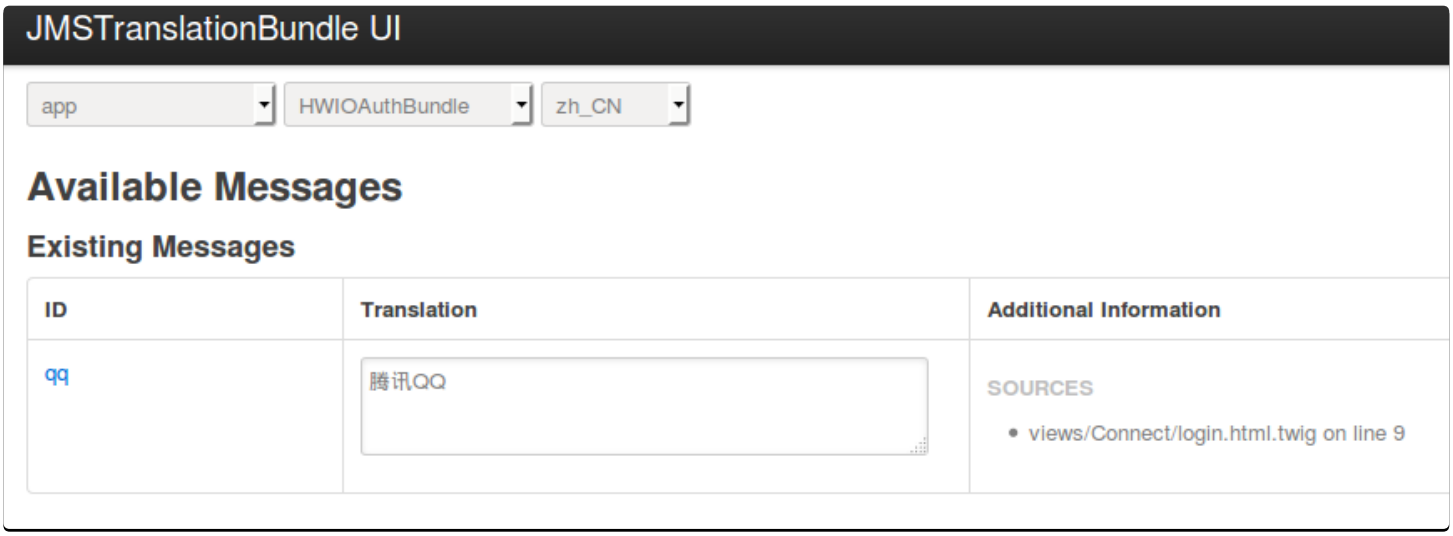
```
$ php app/console assets:install --symlink

```

此命令将所有Bundle里的静态文件（位于`XxxBundle/Resources/public`），都创建了一个软链接（类似`ln -s`命令）到Web目录的`bundle`目录下。

有些系统可能不支持软链接，可以尝试去掉`--symlink`参数，直接复制到`web`的`bundles`目录下。

我们再次刷新`/_trans`页面，可以看到这样的页面：



修改完中文翻译（没有确认按钮，只要输入框失去焦点就已经保存了），我们再刷新`/oauth/login`页面，按钮文字依然没有被翻译，这是因为我们还没有设置项目的默认语言，这需要在`config.yml`文件中指定：

```
framework:
    ...
    # 注意以下设置的区别：
    translator:
        fallbacks: ["%locale%"] # 如果某个语言没有对应的翻译，就显示fa

    default_locale: "%locale%" # 假如用户没有明确表示想要看哪个语言的页
                               # 就认为当前用户是想看%locale%指定语言
                               # 用户可通过HTTP请求中的`accept_langu
                               # 或者路由里的设置比如`domain.com/zh_

```

最后，修改`parameters.yml`文件

```
1 locale: zh_CN
2
```

再刷新页面，翻译已经完成