Symfony2框架实战教程——第三天：用KnpPaginatorBundle实现翻页

34条回复

昨天我们已经创建好了新闻的首页。今天我们来实现添加新闻并且显示新闻的功能，并且学会使用Composer添加第三方Bundle来加速开发。

创建业务数据模型

新闻数据算是我们业务模型里必不可少的模型之一。根据我们之前对需求的分析，我们可以很容易想到，新闻模型News需要的属性：

- 标题属性
- 文本属性

接下来，我们要在AppBundle里创建它，但是这些数据还需要一个持久层来保存数据，例如之前配置的Mysql。目前流行的开发方式，无论是Java还是ROR，都会使用ORM将数据库字段和类属性关联起来。

Symfony2框架本身并不包含ORM工具（严格意义上来说，Symfony2框架，即FrameworkBundle，不包含ORM，安全组件，模板引擎，日志工具，邮件组件等一系列工具），只不过Symfony installer将一些推荐的，Web开发常用的工具，都默认安装了。如果你已经知道如何用Mysql来存储/获取数据，也不一定非要用ORM。这里我们为了快速开发，也为了省一些精力，就使用默认提供的Doctrine2 ORM，它会给我们的开发带来许多便利。

决定了使用Doctrine2，我们除了定义好News类，还需要写配置文件，让Doctrine2 ORM将News类同数据库某个表关联起来。听起来要做的工作不少，不过且慢，DoctrineBundle里自带的代码生成工具能让我们的开发再快一点点：

```
1 $ php app/console doctrine:generate:entity
2
```

此时会有欢迎提示出现，并且让你输入一个模型的”短名字“，为什么说是要输入短名字？News并不是全名，全名应该是包含命名空间的，比如我们的News全名应该是AppBundle\Entity\News，Entity是什么？Entity只是一个习惯性叫法。在Doctrine2的世界里，只要是ORM过的模型，都叫Entity（除此之外还有用MongoDB作为

存储方案的ODM，ODM过的模型习惯称之为Document）。

后面我们就按照他们的提示，分别输入：

```
1  The Entity shortcut name: AppBundle:News
2  Configuration format (yml, xml, php, or annotation) [annotation]:
3  New field name (press <return> to stop adding fields): title
4  Field type [string]: #直接回车
5  Field length [255]: 70 #这里我们很明确只需要70的长度
6  New field name (press <return> to stop adding fields): body
7  Field type [string]: text
8  New field name (press <return> to stop adding fields): #直接回车停止
9  Do you want to generate an empty repository class [no]? yes 是否创建
10 Do you confirm generation [yes]? #回车完成代码生成
11
```

刷新src/AppBundle目录，多了一个Entity目录，此目录包含了两个文件：News.php和NewsRepository.php。

打开News.php我们可以看到，News类已经生成好了，并且还有用注解格式写的ORM配置。我的建议是可以用生成代码工具尽量用，一是快，二是不容易写错字。

此时，我们就可以用DoctrineBundle的数据库操作工具来生成数据的数据库和表了：

```
1  $ php app/console doctrine:database:create
2  $ php app/console doctrine:schema:create
3
```

数据库创建好之后，我们应该创建”新建新闻“页和”新闻详情“页、以及更新我们之前写的”新闻首页“。不过且慢，DoctrineBundle不仅仅能生成Entity，它还能根据Entity直接生成相关的Controller！这里我们先把之前的NewsController类文件删掉。然后使用下面命令：

```
1  $ php app/console doctrine:generate:crud
2
```

如同之前的命令，我们只用回答它的问题就行了：

```
The Entity shortcut name: AppBundle:News
Do you want to generate the "write" actions [no]? yes
Configuration format (yml, xml, php, or annotation) [annotation]:
Routes prefix [/news]:
Do you confirm generation [yes]?

```

我们会发现，被删除的NewsController又被重新生成，并且多了好多代码。先不管每个控制器方法里写了啥，我们先检查路由配置。我们会发现，一切都很好，除了首页的路由名字从以前的news_index变成了news，此时我们可以将news改回news_index，也可以考虑将之前的news_index定为news。我们这里选择后者，毕竟已经为我们定义了一套规则，并且也不赖，何必再去折腾别的命名方式？还好代码也不多，目前我们只用把首页模板default/index.html.twig里的news_index改成news就行了。

doctrine:genearte:crud做的事情就比较多了，它不仅生成了控制器，所有的模板文件也都生成了，并且还生成了表单类。我们先不管表单类，先访问新闻首页/news/试试，没有意外的话，你们可以看到一个从新建、显示、编辑、删除都完全可用的新闻功能。

需要注意的是：从Symfony2.6开始，模板文件推荐是放在app/Resources下的，但是doctrine:generate:crud命令还是将模板文件放在了AppBundle的Resources目录。不仅如此，也不推荐使用@Template注解来猜模板路径（官方说法：主要因为性能问题），所以这里我们得把生成的src/AppBundle/Resources目录移到app目录，并且去掉控制器类里的所有@Template注解，而直接使用$this->render方法。

有一些我们暂时用不着的方法，可以先去掉。比如删除相关的方法deleteAction以及createDeleteForm，以及相关的调用全部去掉。模板里面有用到delete_form的地方，也都删掉。为了节省代码，我们把newAction/createAction，以及editAction/updateAction合并为一个方法，这也是Symfony官方所推荐的最佳实践：

```
/**
 * @Route("/new", methods={"POST", "GET"}, name="news_new")
 */
public function newAction(Request $request)
{
    $entity = new News();
    $form = $this->createCreateForm($entity);
    $form->handleRequest($request);
```

```
 9
10      if ($form->isValid()) { // 注意如果不是POST请求，isValid方法会返回
11          $em = $this->getDoctrine()->getManager();
12          $em->persist($entity);
13          $em->flush();
14
15          return $this->redirect($this->generateUrl('news_show', ['i
16      }
17
18      return $this->render('news/new.html.twig', [
19          'entity' => $entity,
20          'form'   => $form->createView(),
21      ]);
22  }
23
24  /**
25   * @Route("/{id}/edit", methods={"GET", "PUT"}, name="news_edit")
26   */
27  public function editAction(Request $request, $id)
28  {
29      $em = $this->getDoctrine()->getManager();
30
31      $entity = $em->getRepository('AppBundle:News')->find($id);
32
33      if (!$entity) {
34          throw $this->createNotFoundException('Unable to find News
35      }
36
37      $editForm = $this->createEditForm($entity);
38      $editForm->handleRequest($request);
39
40      if ($editForm->isValid()) {
41          $em->flush();
42
43          return $this->redirect($this->generateUrl('news_edit', ['i
44      }
45
46      return $this->render('news/edit.html.twig', [
47          'entity'      => $entity,
48          'edit_form'   => $editForm->createView(),
49      ]);
```

```
50 }
51
```

注意合并之后，之前的两个路由news_create和news_update已经没有了，所以NewsController里相关的代码，都需要更新。

另外细心的同学会发现注解@Method也被我删除了，因为@Route其实是包含@Method的功能的。这里看个人爱好。

如果你已经清理干净了@Method和@Template注解，那么以下的代码就可以删除了：

```
1 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Template;
2 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
3
```

反过来也说明一个需要小心的点：如果你要使用某个注解功能，是需要use到当前代码文件的

使用KnpPaginatorBundle实现翻页功能

现在有一个小问题，列表总是需要翻页的。此时大家如果心里在琢磨”翻页算法“，那说明你的开发方式还不够Symfony，或者说还不够敏捷。像翻页这种常见的需求，现成的库一定是很多而且已经非常成熟，一个人花时间做设计和开发，让成千上万的其他人节省时间，正是开源软件伟大之处之一，除非是自己个人爱好，省时间的事情是一定要做的，毕竟工程师的主要职责是实现而不是钻研。目前支持Doctrine2的翻页库也有二三，而且大都已经Bundle化了。这里我推荐一款用得比较多的：KnpPaginatorBundle

安装第三方Bundle需要使用Composer。安装Composer不在本篇范围内，大家可以上网搜搜，也可以看看我这篇博客。这里我只用提醒大家一点：因为某种众所周知的原因，Composer的速度不是很快，大家可以试试镜像。

准备好Composer以后，执行以下命令（假设已经把composer.phar改名为composer）

```
1 $ composer require "knplabs/knp-paginator-bundle"
2
```

昨天说过，之后将Bundle的”代表类“注册到AppKernel里，才能使用Bundle所提供的配置文件以及”服务“，什

么是服务，稍后会说。

我们先完成KnpPaginatorBundle的注册：

```
// app/AppKernel.php
public function registerBundles()
{
    return array(
        // ...
        new Knp\Bundle\PaginatorBundle\KnpPaginatorBundle(),
        // ...
    );
}

```

然后新闻首页的代码做一点更新：

```
/**
 * @Route("/", name="news")
 */
public function indexAction(Request $request)
{
    $em = $this->getDoctrine()->getManager();

    $qb = $em->getRepository('AppBundle:News')->createQueryBuilder

    $paginator = $this->get('knp_paginator');
    $pagination = $paginator->paginate($qb, $request->query->getIn

    return $this->render('news/index.html.twig', [
        'pagination' => $pagination,
    ]);
}

```

其中`$this->get('knp_paginator')`得到的就是一个服务。目前不用深究服务是什么，只用知道服务也是一个对象，此对像会依赖一些其他的对象。如果手工初始化，需要写很多代码（依赖的对像的初始化，依赖的对象的依赖对象的初始化……），而Symfony2的Service Container组件，可以通过配置文件来描述这种对象之间的依赖关系，进一步在实际使用的时候，只需用`$this->get`方法通过一个设置好的唯一名字去获取组件就

行了（$this->get()其实是$this->container->get()的快捷方式）。依赖注入和服务容器是Symfony2框架中运用得很多的概念之一，目前有个感觉就行，以后还会提到。

相应的模板文件app/Resources/views/news/index.html.twig也做一点调整：

```
<tbody>
    {% for entity in pagination %}
        <tr>
            ...
        </tr>
    {% endfor %}
</tbody>
<tfoot>
    <tr>
        <td colspan="4">
            {{ knp_pagination_render(pagination) }}
        </td>
    </tr>
</tfoot>

```

分页我们就实现了。

这里我给大家介绍一下createQueryBuilder方法。首先大家应该知道的是，从数据库里获取数据，需要向数据库发起数据请求，一般来说是一段字符串，比如大家最熟悉不过的SELECT * FROM xxx这样的SQL语句，这个请求就叫“Query”，而“Query Builder”就是指创建Query的东西。其实有了QueryBuilder这个东西，在拼装SQL语句的时候，会方便许多，也不容易出错。大家可以对比一下以下代码：

```
$sql = "SELECT * FROM user WHERE is_active = 1";
if ($isAdmin) {
    $sql .= " AND user_group = 'admin'"; // AND前面的空格很容易忘
}

if ($sortByUsername) {
    $sql .= " ORDER BY username"; // 顺序还不能错，order必须在where之
}

// 如果使用QueryBuilder

```

```
11 $qb = $doctrine->createQueryBuilder('u')->where('u.isActive = 1');
12
13 if ($sortByUsername) {
14     $qb->orderBy('u.username');
15 }
16
17 if ($isAdmin) {
18     $qb->addWhere('u.userGroup = 'admin'');
19 }
20
```

需要各位读者注意的是，如果记录条数不满每页记录数，分页控件是不会出现的（KnpPaginator默认是10条，您也可以自己设置每页记录数，至于怎么调，留给大家当作业吧）