如何定义404

404，说白了就是找不到页面，那么如何定义“找不到”呢？

我们可以通过源代码来看看Spring MVC如何定义“404”的：

[java]view plaincopy

```
1. // Determine handler for the current request.
2. mappedHandler = getHandler(processedRequest, false);
3. if (mappedHandler == null || mappedHandler.getHandler() == null) {
4.     noHandlerFound(processedRequest, response);
5. return;
6. }
```

getHandler是根据请求的url，通过handlerMapping来匹配到Controller的过程。

如果匹配不到，那么就执行noHandlerFound方法。这个方法很简单，返回一个404的错误代码。

我们的Web容器，比如tomcat，会根据这个错误代码来生成一个错误界面给用户。

那么，我们如何自定义这个界面呢？

重写noHandlerFound方法

最先想到的肯定是重写noHandlerFound方法，这个方法是protected，可以重写。

我们需要将页面重定向到我们自定义的404界面，那么只需要

[java]view plaincopy

```
1. @Override
2. protectedvoid noHandlerFound(HttpServletRequest request,
3.         HttpServletResponse response) throws Exception {
4.     response.sendRedirect(request.getContextPath() + "/notFound");
5. }
```

这里我们的Controller里需要定义一个@requestMapping("/notFound")的这么一个方法，用来返回一个404页面

或者，这里应该可以采用直接访问静态文件的方法。

另外，也可以通过抛出一个异常NoSuchRequestHandlingMethodException

这样我们就实现了自定义的404页面。那么，还有别的方法吗？

利用Spring MVC的最精确匹配

Spring MVC对于url的匹配采用的是一种叫做“最精确匹配的方式”，举个例子

比如我们同时定义了“/test/a”，"/test/*"，那么若请求的url结尾为/test/a，那么则会匹配精确的那个，也就是"/test/a"

我们是不是可以利用这个特点来找到那些找不到的页面？

1、首先我们定义一个拦截所有url的规则@requestMapping("*")，那么实际上不存在找不到的页面了，也就是永远不会进入noHandlerFound方法体内

2、后面的步骤和平时一样，为别的请求都配置上@requestMapping

那么请求过来，要么进入我们精确匹配的method（也就是找的到的），要么进入@requestMapping("*)拦截的方法体内（也就是找不到的）

那么我们只要让@requestMapping("*)拦截的这个方法返回一个自定义的404界面就OK了~

利用web容器提供的error-page

还记得之前提到的web容器会提供一个404的默认界面吗？

其实我们完全可以替换成我们自己的界面，那么看起来这种方法应该是最简单的了。

只需要在web.xml文件中写上如下代码就可以了：

[html]view plaincopy

```
1. <error-page>
2. <error-code>404error-code>
3. <location>/resource/view/404.htmlocation>
4. error-page>
```

不过值得注意的是，这里配置的的location其实会被当成一个请求来访问。

那么我们的DispatcherServlet会拦截这个请求而造成无法访问，此时的结果是用户界面一片空白。

所以这里的404.htm其实是一个静态资源，我们需要用访问静态资源的方式去访问。

而在我的Spring MVC里，resource目录下的文件都是不会被拦截的

比较三种方式的区别

1、最方便：那肯定是第三种了，我们只需要提供一个静态页面即可

2、最快捷：第一种肯定最慢，因为它会发起2个请求。第二种和第三种应该差不多

3、最灵活：从灵活性上来看，第三种肯定是最缺乏的，但是其实对于404来说并不是需要经常变化的，不

过也保不准可能可以允许用户自定义404界面等，这里一、二两种方式则提供了灵活性。

4、通用性：第三种应该是最通用了，而一、二 两种则要依赖Spring MVC