渲染Web视图

1 视图解析

控制器方法美哟直接占城浏览器中渲染所需要的HTLML。只是讲一些数据填充到模型中共你，然后将模型传递个一个用来渲染的视图。这些方法返回一个String类型的值，这个值是视图的逻辑名称，不会直接引用具体的视图实现。

将控制器中请求处理的逻辑和视图中的渲染实现解耦是SpringMVC的一个重要特性。

通过视图名称从而来确定渲染模型

```
@Bean
public ViewResolver viewResolver() {
    FreeMarkerViewResolver viewResolver = new FreeMarkerViewResolver();
    viewResolver.setPrefix("/WEB-INF/views/");
    viewResolver.setSuffix(".html");
    viewResolver.setContentType("text/html; charset=UTF-8");

    viewResolver.setExposeSpringMacroHelpers(true);
    return viewResolver;
}
```

```
public interface ViewResolver {
   @Nullable
   View resolveViewName(String viewName, Locale locale) throws Exception;
}
```

```
public interface View {
   void render(@Nullable Map<String, ?> model, HttpServletRequest request,
HttpServletResponse response)
         throws Exception;
}
```

View接口的任务就是接受模型以及Servlet的request和response对象，并将输出结果渲染到response。

Spring自带13个视图解析器，能将逻辑视图名转换为物理实现：

视图解析器	描述
BeanNameViewResolver	将视图解析为Spring应用上下文中共的bean，其中bean的ID与视图的名字相同
ContentNegotiatingViewResolver	通过考虑客户端需要的内容类型来解析视图，委托给另外一个能够产生对应类型的属兔解析器
FreeMarkerViewResolver	将视图解析为FreeMarker模板
InternalResourceViewResolver	将视图解析为Web引用的内部资源(一般为jsp)
JasperReportViewResolver	将是视图解析为JasperReports定义
ResourceBoundleViewResolver	将视图解析为资源bundle(一般为属性文件)
TilesViewResolver	将驶入解析为Apache Tile定义，其中 tile ID与视图名称相同
UrlBasedViewResolver	直接根据视图的名称解析视图，视图的名称会匹配一个物理视图的定义
VelocityLayoutViewResolver	将视图解析为Velocity布局，从不同的Velotics模板中组合页面
VelocityViewResolver	将是如解析为Velocity模板
XmlViewResolver	将视图解析为特定XML文件中的bean定义。类似于BeanNameViewResolver
XsltViewResolver	将视图解析为XSTL转换后的结果

Thymeleaf是一种用来替代JSp的新兴技术，Spring提供了与Thymeleaf的远射改模板下作的视图解析器。这主公模板更像是最终共产生的HTML，而不是驱动他们的Java代码。

2 JSP

3 ApacheTilles 视图定义布局

4 使用Thymeleaf

JSP规范是与Servlet规范耦合的，意味着他只能用在基于Servlet的Web应员中，。JSp模板不能作为通用的模板，也不能用于非SErvlet的Web应用。

4.1 配置Thymeleaf视图解析器

为了在Spring中是哟共Thymeleaf，需要配置三个启用与Spring集成的bean

- **ThymeleafViewResolver 将逻辑视图名称解析为Thymeleaf模板视图**
- **SpringTemplateEngine 处理模板并渲染结果**
- **TemplateResolver 加载Thymeleaf模板**

```
@Bean
public ViewResolver viewResolver(ISpringTemplateEngine engine) {
    ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
```

```
    viewResolver.setTemplateEngine(engine);
    return viewResolver;
}

@Bean
public ISpringTemplateEngine springTemplateEngine(ITemplateResolver
templateResolver) {
    SpringTemplateEngine springTemplateEngine = new SpringTemplateEngine();
    springTemplateEngine.addTemplateResolver(templateResolver);
    return springTemplateEngine;
}

@Bean
public ITemplateResolver templateResolver(ServletContext servletContext) {
    ServletContextTemplateResolver resolver = new
ServletContextTemplateResolver(servletContext);
    resolver.setPrefix("WEB-INFO/views/");
    resolver.setSuffix(".html");
    resolver.setCharacterEncoding("UTF-8");
    resolver.setTemplateMode("HTML5");
    return resolver;
}
```

- ISpringTemplateEngine 会在Spring中启用Thymeleaf引擎，用来解析模板，并给予这些斑斑渲染结果
- ITemplateResolver 会最终定位和查找模板，setTemplateMode属性设置成HTML5，表明预期要借些的模板会渲染成HTML5输出。

4.2 定义Thymeleaf模板

Thymeleaf很大程度上是HTML文件，Thymeleaf之所以能够发挥作用，是它通过自定义的命名空间，为标准的HTML标签集合添加Thymeleaf属性。

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"> 声明Thymeleaf命名空间
<head>
    <title>Spitter</title>
    <link rel="stylesheet"
          type="text/css"
          th:href="@{/resources/style.css}"></link>到样式表的th:href连接
</head>
<body>
<div id="header" th:include="page :: header"></div>

<div id="content">
    <h1>Welcome to Spitter</h1>

    <a th:href="@{/spittles}">Spittles</a> |  到页面的th:href连接
    <a th:href="@{/spitter/register}">Register</a>

    <br/>

    View: <span th:text="${view}">unknown</span>
</div>
<div id="footer" th:include="page :: copy"></div>
</body>
```

```
</html>
```

th:href属性与原生HTML属性类似，也就是href属性，并且可以按照形同的而方式来使用，th:href特殊在于他的值可以包含Thymeleaf表达式，用来计算动态的值。会袁燃成一个标准的href属性，其中会包含在渲染时动态创建得到的值。

这是Thymeleaf命名空间中共很多属性的运行方式：他们对应标准的HTML属性，并且具有相同的名称，但是会渲染一些计算后得到的值。用到的"@{}"表达式，用来计算相对于URL的路径(就想在JSp页面中，可以是哟共的JSTLc:url标签或Springs:url标签类似)

这意味着Thymeleaf模板与JSp不同，它能够按照原始的方式进行编辑甚至渲染，而不必经过任何类型的处理。当然，我们需要Thymeleaf来处理模板并渲染得到最终共期望的输出。即便如此，如果没有任何特殊的处理，home.html也能够加载到Web浏览器中，并且看上去与完整渲染的效果类似

4.2.2 借助Thymeleaf实现表单绑定

表当绑定是Spring MVC的一项重要特性。它能够将表单提交的数据填充到命令对象中，并将其传递给控制器，而在展现表单的时候，表单中也会填充命令对象中的值。如果没有i表单绑定功能的化，我们需要确保HTML表单域要映射后端命令对象中的属性，并且在校验失败后展现表单的时候，还要负责确保输入域中值要设置为命令对象的属性

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Spitter</title>
    <link rel="stylesheet" type="text/css"
          th:href="@{/resources/style.css}"></link>
</head>
<body>
<div id="header" th:include="page :: header"></div>

<div id="content">
    <h1>Register</h1>

    <form method="POST" th:object="${spitter}" enctype="multipart/form-data">
        <div class="errors" th:if="${#fields.hasErrors('*')}">
            <ul>
                <li th:each="err : ${#fields.errors('*')}"
                    th:text="${err}">Input is incorrect
                </li>
            </ul>
        </div>
        <label th:class="${#fields.hasErrors('firstName')}? 'error'">First
Name</label>:
        <input type="text" th:field="*{firstName}"
               th:class="${#fields.hasErrors('firstName')}? 'error'"/><br/>

        <label th:class="${#fields.hasErrors('lastName')}? 'error'">Last
Name</label>:
        <input type="text" th:field="*{lastName}"
               th:class="${#fields.hasErrors('lastName')}? 'error'"/><br/>

        <label th:class="${#fields.hasErrors('email')}? 'error'">Email</label>:
        <input type="text" th:field="*{email}"
               th:class="${#fields.hasErrors('email')}? 'error'"/><br/>

        <label th:class="${#fields.hasErrors('username')}?
'error'">Username</label>:
```

```
        <input type="text" th:field="*{username}"
               th:class="${#fields.hasErrors('username')}? 'error'"/><br/>

        <label th:class="${#fields.hasErrors('password')}?
'error'">Password</label>:
        <input type="password" th:field="*{password}"
               th:class="${#fields.hasErrors('password')}? 'error'"/><br/>

        <label>Profile Picture</label>:
        <input type="file"
               name="profilePicture"
               accept="image/jpeg,image/png,image/gif"/><br/>

        <input type="submit" value="Register"/>
    </form>
</div>
<div id="footer" th:include="page :: copy"></div>
</body>
</html>
```

th:class属性会渲染为一个class属性，他的值是更具给定的表达式计算得到的。在例子中，他会检查firstName域有没有校验错误，如果有，class属性在渲染时的值为error。如果这个域没有错误，将不会渲染class属性

 标签使用了th:field属性，用来应用后端对象的firstName域。在Thymeleaf模板这中，很多情况下所使用的属性都对应于标准的HTML属性

${} 表达式是变量表达式。他们会是对象导航语言表达式。但是在使用Spring的时候，他们呢是SpEL表达式。在${spitter}中，他会解析为key为spitter的model属性

*{} 表达式，他们呢是选择表达式。变量表达式是基于整个SpEL上下文计算的，而选择表达式是基于某一个选中对象的。在本例表单中，选中对象就是

标签中th:object属性所设置的对象:模型中的Spitter对象。因此”**{firstName}“就会计算为Spitter对象的firstName属性。