

View

浙江大学城市学院

彭彬

pengb@zucc.edu.cn

View

View可以简单的理解为用户交互界面，而从互联网时代开始，View最核心的技术表达方式就是基于浏览器的，浏览器中核心运行的东西大概包括下面这么几种：

- 1) HTML：超文本标注语言，浏览器理解HTML语言标签的含义，构造一颗DOM树，然后显示出来；
- 2) CSS：层叠样式表，用于刻画显示的一些效果；
- 3) JavaScript：提供动态性，操纵DOM元素；
- 4) 其他媒体：文字、图片、音视频等

The screenshot shows the Baidu homepage with the developer tools open. The Elements tab displays the DOM tree for the page. A callout box highlights a specific element in the DOM tree, showing its bounding box dimensions as 1250 x 776 pixels. The right side of the screen shows the actual Baidu search results page, featuring news articles and a banner about medical aid to海外.

View演进及其模式

前后端不分的时代~原始CGI模式、JSP/ASP模式

互联网发展的早期，前端开发是一体的，前端代码是后端代码的一部分。

后端收到浏览器的请求

生成静态页面

发送到浏览器

后端MVC的开发模式~基于JSP技术的标签模式，各种模板技术

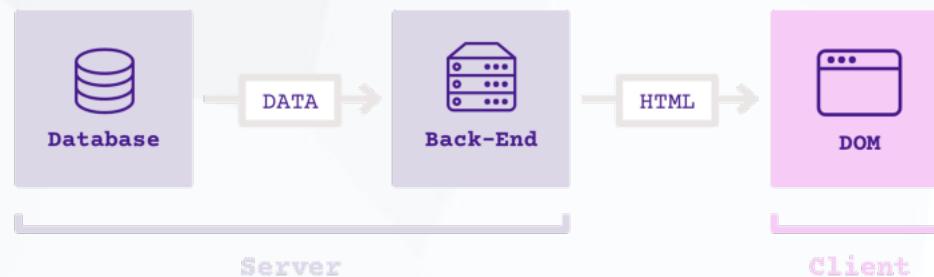
那时的网站开发，采用的是后端MVC模式。

模型（模型层）：提供/保存数据

控制器（控制层）：数据处理，实现业务逻辑

视图（视图层）：展示数据，提供用户界面

前端只是后端MVC的V.



View演进及其模式

Ajax技术诞生

2004年：Gmail；2005年：谷歌地图

前端不再是后端的模板，可以独立得到各种数据并执行复杂的交互操控，背后是浏览器越来越强大以及JavaScript工程化。

前端 MVC 框架

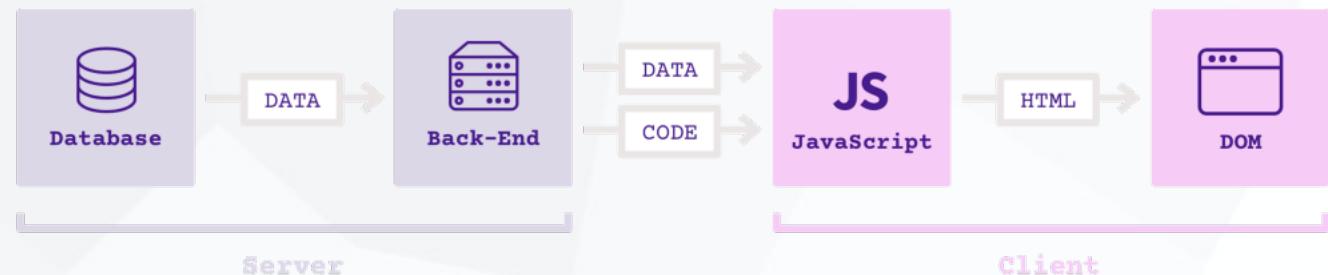
前端View通过 Ajax 得到数据，自身也分为了MVC的层次。

前端代码变得也需要保存数据、处理数据、生成视图，从而使得前端View演进为了前端App，后端退回到纯服务设计（RESTful）。

SPA (Single-page application) /App

App时代，前端的多样化，加速了前端

与后端的分离，使得后端越来越服务化。



深入理解一个小问题

我们在开始讨论View之前，我们先深入来回答一下这个问题

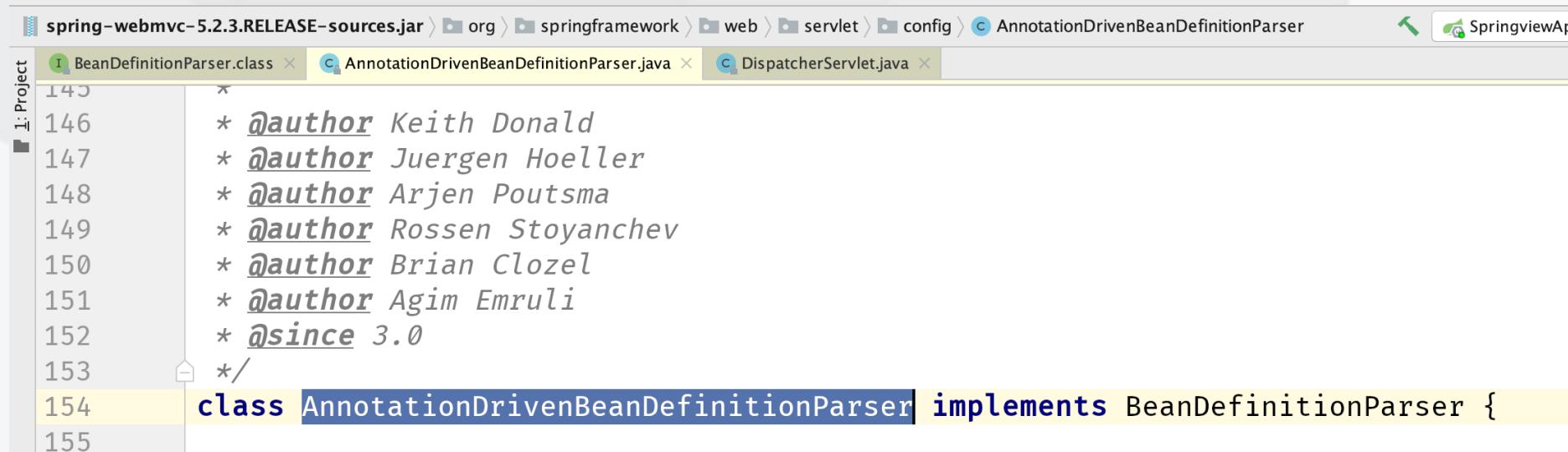
前面的例子中*Spring MVC*为什么能自动返回json报文？

这个问题的答案就是

- 1) Spring Boot做了大量自动配置，我们不需要进行配置处理，所以隐藏了很多工作细节
- 2) Spring MVC有默认的消息转换机制，在最常见的情况下帮我们做了转换处理，所以隐藏的更深了

Spring MVC为什么能自动返回json报文~默认配置器

Spring MVC提供了一个默认的配置类AnnotationDrivenBeanDefinitionParser，其对Spring MVC 提供了非常多默认配置



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left): Shows the project structure.
- Java Editor (center): Displays the `AnnotationDrivenBeanDefinitionParser.java` file content.
- Toolbar (top): Includes icons for back, forward, search, and refresh.
- Status Bar (bottom): Shows the file path: `spring-webmvc-5.2.3.RELEASE-sources.jar > org > springframework > web > servlet > config > AnnotationDrivenBeanDefinitionParser`.

The code in the editor is as follows:

```
145 *
146 * @author Keith Donald
147 * @author Juergen Hoeller
148 * @author Arjen Poutsma
149 * @author Rossen Stoyanchev
150 * @author Brian Clozel
151 * @author Agim Emruli
152 * @since 3.0
153 */
154 class AnnotationDrivenBeanDefinitionParser implements BeanDefinitionParser {
155 }
```

我们来看看AnnotationDrivenBeanDefinitionParser类的注释说了什么

Spring MVC为什么能自动返回json报文~默认配置器

配置了两个默认的HandlerMapping，用于处理controller和url的映射关系

- * A `{@link BeanDefinitionParser}` that provides the configuration for the
- * `{@code <annotation-driven/>}` MVC namespace element.
- *
- * `<p>This class registers the following {@link HandlerMapping HandlerMappings}:</p>`
- * ``
- * `{@link RequestMappingHandlerMapping}`
- * `ordered at 0 for mapping requests to annotated controller methods.`
- * `{@link BeanNameUrlHandlerMapping}`
- * `ordered at 2 to map URL paths to controller bean names.`
- * `/ul`

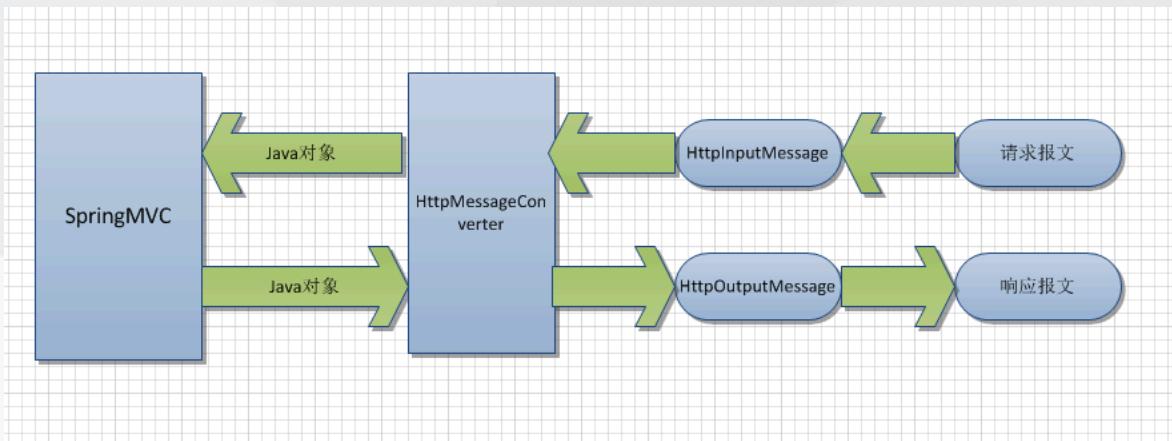
Spring MVC为什么能自动返回json报文~默认配置器

配置了三个默认的 *RequestHandlerAdapter*, 用于处理如何选择controller类 (处理Request的逻辑过程)。其中 *RequestMappingHandlerAdapter* 就是实现了我们当前 controller 代码通过 @GetMapping, @PostMapping 来标注的功能

```
* <p>This class registers the following {@link HandlerAdapter HandlerAdapters}:  
* <ul>  
* <li>{@link RequestMappingHandlerAdapter}  
* for processing requests with annotated controller methods.  
* <li>{@link HttpRequestHandlerAdapter}  
* for processing requests with {@link HttpRequestHandler HttpRequestHandlers}.  
* <li>{@link SimpleControllerHandlerAdapter}  
* for processing requests with interface-based {@link Controller Controllers}.  
* </ul>
```

Spring MVC为什么能自动返回json报文~消息转换器

Spring MVC使用消息转换器在Java对象和报文之间进行转换



在整个初始化过程中，Spring MVC配置了很多默认转换器

A screenshot of an IDE showing code related to message converters in Spring. The tabs at the top include "AnnotationDrivenBeanDefinitionParser.java", "RequestMappingHandlerAdapter.java", "ByteArrayHttpMessageConverter.class", "StringHttpMessageConverter.class", "SourceHttpMessageConverter.class", and "HandlerAdapter.java". The current file is "HandlerAdapter.java". A search bar at the top shows the query "jackson2Present". The code in the editor is as follows:

```
179     static {
180         ClassLoader classLoader = AnnotationDrivenBeanDefinitionParser.class.getClassLoader();
181         javaxValidationPresent = ClassUtils.isPresent("javax.validation.Validator", classLoader);
182         romePresent = ClassUtils.isPresent("com.rometools.rome.feed.WireFeed", classLoader);
183         jaxb2Present = ClassUtils.isPresent("javax.xml.bind.Binder", classLoader);
184         jackson2Present = ClassUtils.isPresent("com.fasterxml.jackson.databind.ObjectMapper", classLoader);
185             ClassUtils.isPresent("com.fasterxml.jackson.core.JsonGenerator", classLoader);
186         jackson2XmlPresent = ClassUtils.isPresent("com.fasterxml.jackson.dataformat.xml.XmlMapper", classLoader);
187         jackson2SmilePresent = ClassUtils.isPresent("com.fasterxml.jackson.dataformat.smile.SmileFactory", classLoader);
188         jackson2CborPresent = ClassUtils.isPresent("com.fasterxml.jackson.dataformat.cbor.CBORFactory", classLoader);
189         gsonPresent = ClassUtils.isPresent("com.google.gson.Gson", classLoader);
190     }
```

Spring MVC为什么能自动返回json报文~消息转换器

当在环境中存在Jackson库的时候，Spring MVC就设置了基于Jackson的报文转换器，所以一个对象就可以默认通过Jackson库进行和json之间的转换。我们的controller就“自动”的可以返回json报文了

```
AnnotationDrivenBeanDefinitionParser.java x RequestMappingHandlerAdapter.java x ByteArrayHttpMessageConverter.class x StringHttpMessageConverter.class x SourceHttpMessageConverter.class x HandlerAdapter.java x
jackson2Present
private ManagedList<?> getMessageConverters(Element element, @Nullable Object source, ParserContext context) {
    Element convertersElement = DomUtils.getChildElementByTagName(element, childEleName: "message-converters");
    ManagedList<Object> messageConverters = new ManagedList<>();
    if (convertersElement != null) {...}

    if (convertersElement == null || Boolean.parseBoolean(convertersElement.getAttribute(name: "register-default")) {
        messageConverters.setSource(source);
        messageConverters.add(createConverterDefinition(ByteArrayHttpMessageConverter.class, source));

        RootBeanDefinition stringConverterDef = createConverterDefinition(StringHttpMessageConverter.class, source);
        stringConverterDef.getPropertyValues().add(propertyName: "writeAcceptCharset", propertyValue: false);
        messageConverters.add(stringConverterDef);

        messageConverters.add(createConverterDefinition(ResourceHttpMessageConverter.class, source));
        messageConverters.add(createConverterDefinition(ResourceRegionHttpMessageConverter.class, source));
        messageConverters.add(createConverterDefinition(SourceHttpMessageConverter.class, source));
        messageConverters.add(createConverterDefinition(AllEncompassingFormHttpMessageConverter.class, source))

        if (romePresent) {...}

        if (jackson2XmlPresent) {...}
        else if (jaxb2Present) {...}

        if (jackson2Present) {
            Class<?> type = MappingJackson2HttpMessageConverter.class;
            RootBeanDefinition jacksonConverterDef = createConverterDefinition(type, source);
            GenericBeanDefinition jacksonFactoryDef = createObjectMapperFactoryDefinition(source);
            jacksonConverterDef.getConstructorArgumentValues().addIndexedArgumentValue(index: 0, jacksonFactoryDef);
            messageConverters.add(jacksonConverterDef);
        }
    }
}
```

Spring MVC能自动返回xml报文吗？

我们在上述代码片段中，也看见了“xml”字样的很多配置，其实Spring MVC也可以“自动”返回xml格式报文，我们只要在请求头部加上一个Accept: application/xml的头部，然后提供一个xml的root标注就能实现了

```
@GetMapping("/question/xml/{id}")
QuestionXml viewRaw(@PathVariable Integer id) {
    QuestionEntity entity = repository.getOne(id);
    QuestionXml ret = new QuestionXml();
    BeanUtils.copyProperties(entity,ret);
    return ret;
}
```

```
@XmlRootElement
public class QuestionXml {
    private int sid;
    private String title;
    private Date createDate;
    private int creator;
```

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: http://localhost:8080/question/xml/6
- Headers tab selected, showing:
 - Accept: application/xml
 - Key: Value: Description
- Status: 200 OK
- Body tab selected, showing a pretty-printed XML response:

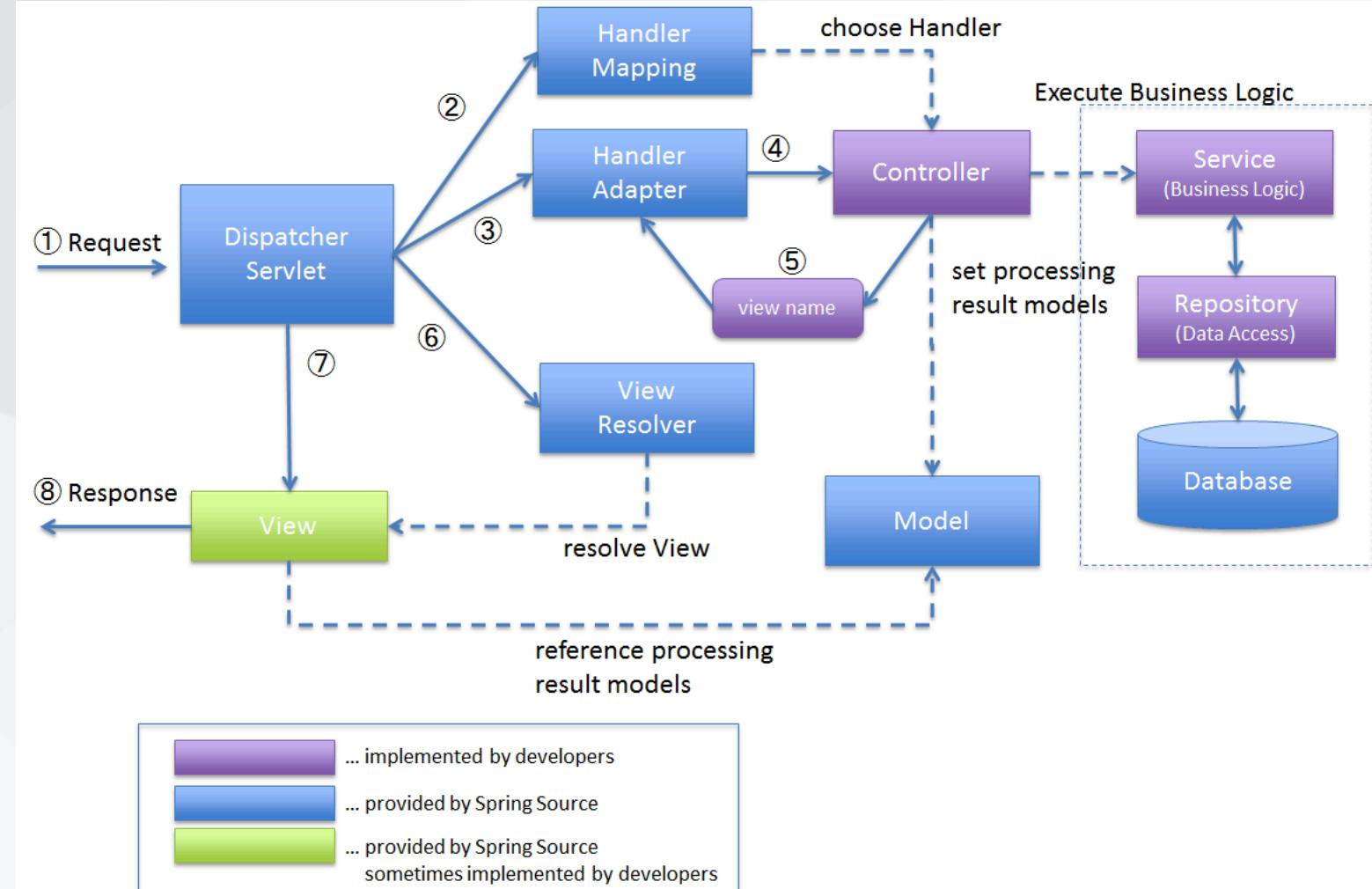
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<questionXml>
    <createDate>2020-04-14T17:42:57+08:00</createDate>
    <creator>1</creator>
    <sid>6</sid>
    <title>title-fyhS</title>
</questionXml>
```

Spring MVC的工作流程

在讨论完上面的小问题后，我们来看看
Spring MVC的工作流，如右图

- 1) 首先是Spring MVC用DispatcherServlet接管了我们请求的入口
([org.springframework.web.servlet.DispatcherServlet](#))
- 2) DispatcherServlet核心处理分为三步：
 - a. 寻找处理器 (Handler)
 - b. 寻找View
 - c. 调用View渲染出界面

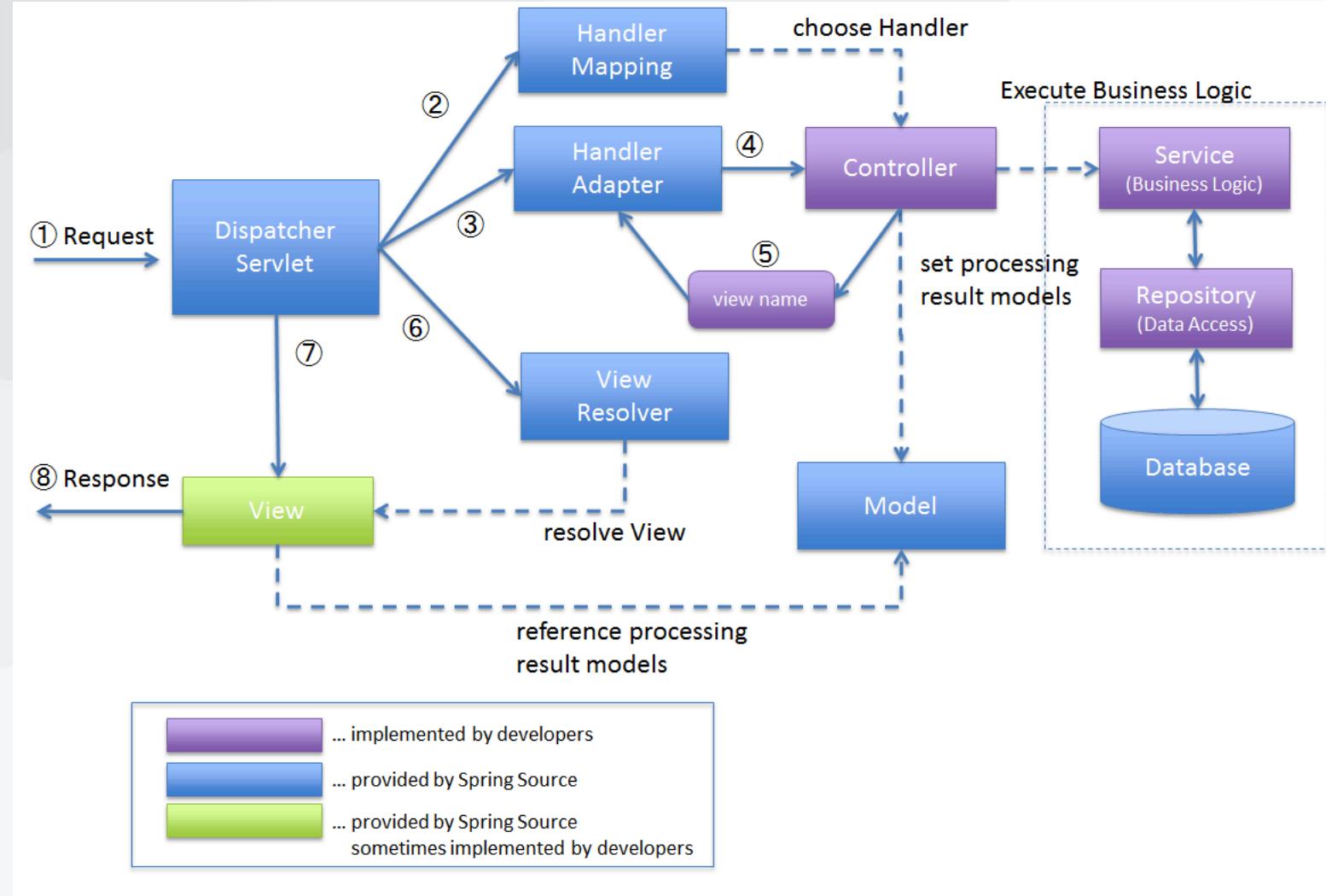
*大家回忆一下Servlet，这个是Java EE的标准组件



Spring MVC工作流程中我们要编写什么呢？

所以基于Spring MVC流程，我们从编写完整的Servlet变成：

- 1) 编写Controller，专注如何组合Service完成具体的业务功能；
- 2) 编写View，专注如何提供界面；



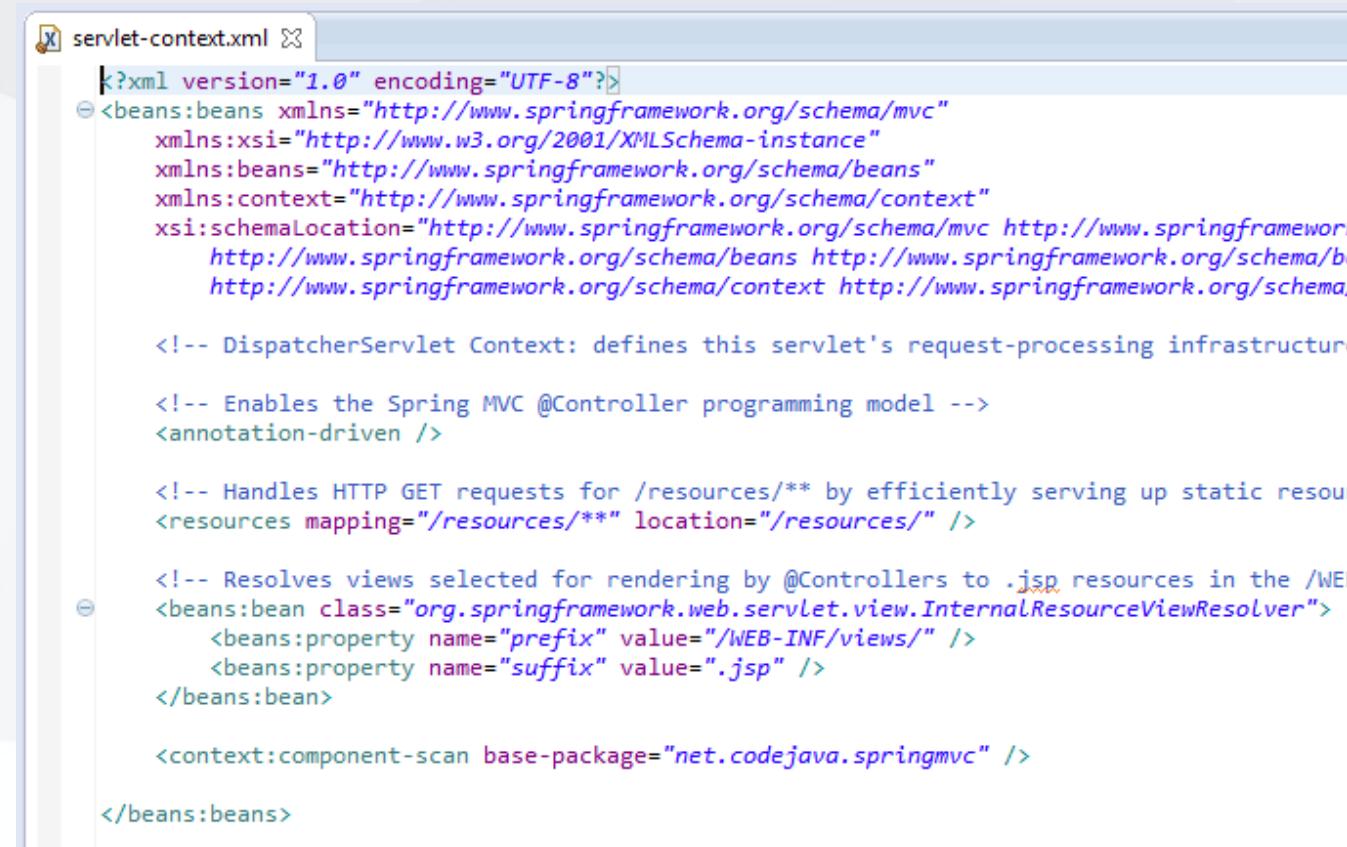
Spring MVC工作流程中对组件进行配置

我们编写完各种Controller, Service, View后需要把这些东西组合起来，告诉Spring MVC如何组装映射。所以最核心的是我们要提供一个配置文件，dispatcher-Servlet.xml。下面是一个“古老”的配置文件，里面需要指明各种Spring MVC需要的“bean”，指示Spring进行注入。

```
Dispatcher-Servlet-old.xml ×
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:p="http://www.springframework.org/schema/p"
5      xmlns:aop="http://www.springframework.org/schema/aop"
6      xmlns:tx="http://www.springframework.org/schema/tx"
7      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
8          http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
9          http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
10
11     <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>
12
13     <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
14         <property name="urlMap">
15             <map>
16                 <entry key="/index.html">
17                     <ref bean="student"/>
18                 </entry>
19             </map>
20         </property>
21     </bean>
22
23     <bean id="student" class="Student">
24         <constructor-arg index="0" type="java.lang.String" value="Ganesh"/>
25         <constructor-arg index="1" type="int" value="20"/>
26         <constructor-arg index="2" type="java.lang.String" value="Computer Science"/>
27     </bean>
28
29     <bean id="viewResolver"
30         class="org.springframework.web.servlet.view.InternalResourceViewResolver"
31         p:prefix="/WEB-INF/jsp/"
32         p:suffix=".jsp" />
33 </beans>
```

Spring MVC工作流程中对组件进行简化配置~使用Annotation

从上面的配置文件我们可以看出，进行配置是一件非常繁琐，并且容易出错的事情，从Spring3.0开始，引入了基于java Annotation机制的配置，以减少配置工作量，但是我们还是需要一个配置文件，这个“比较古老”的配置文件大概像这样：



```
?xml version="1.0" encoding="UTF-8"?<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->
    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources -->
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <context:component-scan base-package="net.codejava.springmvc" />

</beans:beans>
```

- 1) 基本上最核心的是指明annotation-driven标注，让spring启用Java Annotation配置
- 2) 指明“component-scan”，以指示spring要进行Annotation扫描的包

Spring MVC工作流程中对组件进行“零”配置~使用Spring Boot

Spring Boot诞生的核心使命之一就是减少配置压力，所以引入Spring Boot后，我们基本就不需要配置文件了，正如我们示例工程中，直接通过maven引入若干需要的spring boot starter依赖后就可以直接通过Java Annotation进行开发工作了。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Spring Boot依赖

```
@RestController
public class QuestionController {

    private final QuestionRepository repository;
    QuestionController(QuestionRepository repository) { this.repository = repository; }

    @GetMapping("/question/json/{id}")
    Question viewJson(@PathVariable Integer id) {
        QuestionEntity entity = repository.getOne(id);
        Question ret = new Question();
        BeanUtils.copyProperties(entity,ret);
        return ret;
    }
}
```

标注RestController、GetMapping等

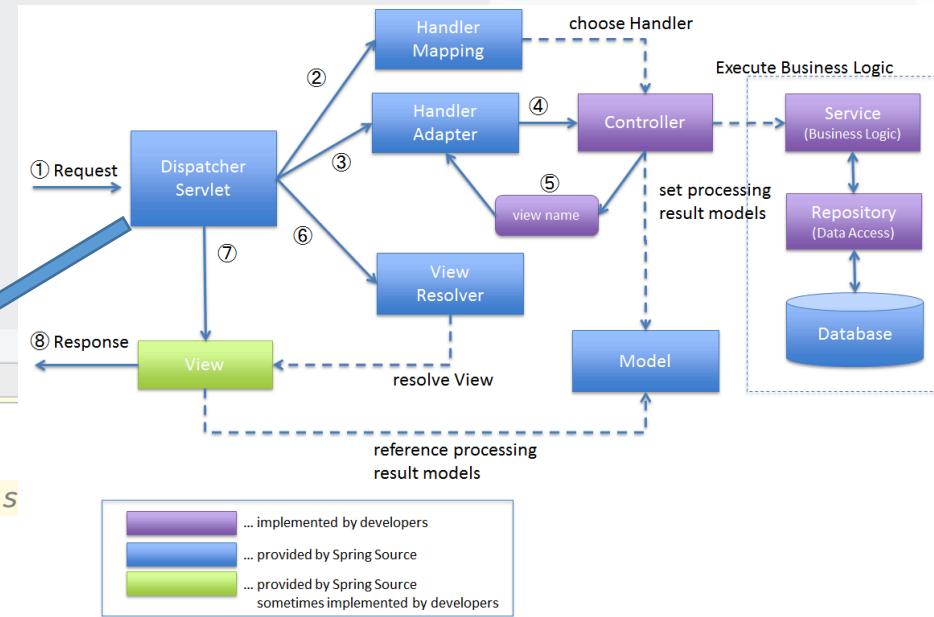
Spring MVC初始化

但是这个简化配置的过程，并没有改变Spring MVC的工作流程，但是使得我们可能被遮蔽了理解Spring MVC工作原理的机会。

Spring MVC的主控程序DispatcherServlet中，我们可以清楚的看见其初始化了很多角色，最核心的是

- HandlerMappings
- HandlerAdapters
- ViewResolvers

```
498     /**
499      * Initialize the strategy objects that this servlet uses.
500      * <p>May be overridden in subclasses in order to initialize further s
501      */
502     protected void initStrategies(ApplicationContext context) {
503         initMultipartResolver(context);
504         initLocaleResolver(context);
505         initThemeResolver(context);
506         initHandlerMappings(context);
507         initHandlerAdapters(context);
508         initHandlerExceptionResolvers(context);
509         initRequestToViewNameTranslator(context);
510         initViewResolvers(context);
511         initFlashMapManager(context);
512     }
```



Spring MVC 中的 View Resolution

和我们今天要讲解的View密切相关的， Spring MVC提供了两个中间角色来解耦框架与具体的视图技术直接关联

- 1) ViewResolver: 提供view的定义与实际的view直接的映射（解耦）
- 2) View: 提供独立于实际的View实现技术的数据表示

也就是说，如果我们如果不像我们之前那样，只返回Json数据提供给前端处理，而是要生成一个视图（View）的话，我们就需要为Spring MVC提供View，然后框架就可以帮助我们生成。

Spring MVC支持多种View实现技术

如同上面所述， Spring MVC的View支持多种视图系统， 官方主要支持的包括：

- 1) Thymeleaf: 模板引擎系统
- 2) FreeMarker: 模板引擎系统
- 3) Groovy Markup: 模板引擎系统
- 4) Script View: 基于脚本的模板库（比如Javascript、Ruby等）
- 5) JSP and JSTL: 古老的JSP(**不推荐使用**)
- 6) PDF and Excel
- 7) Jackson: 如同我们前面所分析， 返回Json/XML也属于一种View
- 8) 其他: 比如RSS、XSLT等等

View实现技术~以Thymeleaf为例

Thymeleaf /'taɪməlɛf/ (<https://www.thymeleaf.org/>) 是一种模板引擎系统

```
1  <table>
2  <thead>
3  <tr>
4  <th th:text="#{msgs.headers.name}">Name</th>
5  <th th:text="#{msgs.headers.price}">Price</th>
6  </tr>
7  </thead>
8  <tbody>
9  <tr th:each="prod: ${allProducts}">
10 <td th:text="${prod.name}">Oranges</td>
11 <td th:text="${numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
12 </tr>
13 </tbody>
14 </table>
```

Thymeleaf is a modern **server-side Java template engine** for both web and standalone environments.

Thymeleaf's main goal is to **bring elegant natural templates** to your development workflow — HTML that can be correctly displayed in browsers and also work as static prototypes, allowing for stronger collaboration in development teams. With modules for Spring Framework, a host of integrations with your favourite tools, and the ability to plug in your own functionality, Thymeleaf is ideal for modern-day HTML5 JVM web development — although there is much more it can do.

View实现技术~以Thymeleaf为例

从上面的工作原理我们知道，当Spring MVC完成controller的处理后，就会尝试获取视图处理对象，为了使用Thymeleaf，我们可以首先引入其Spring Boot依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

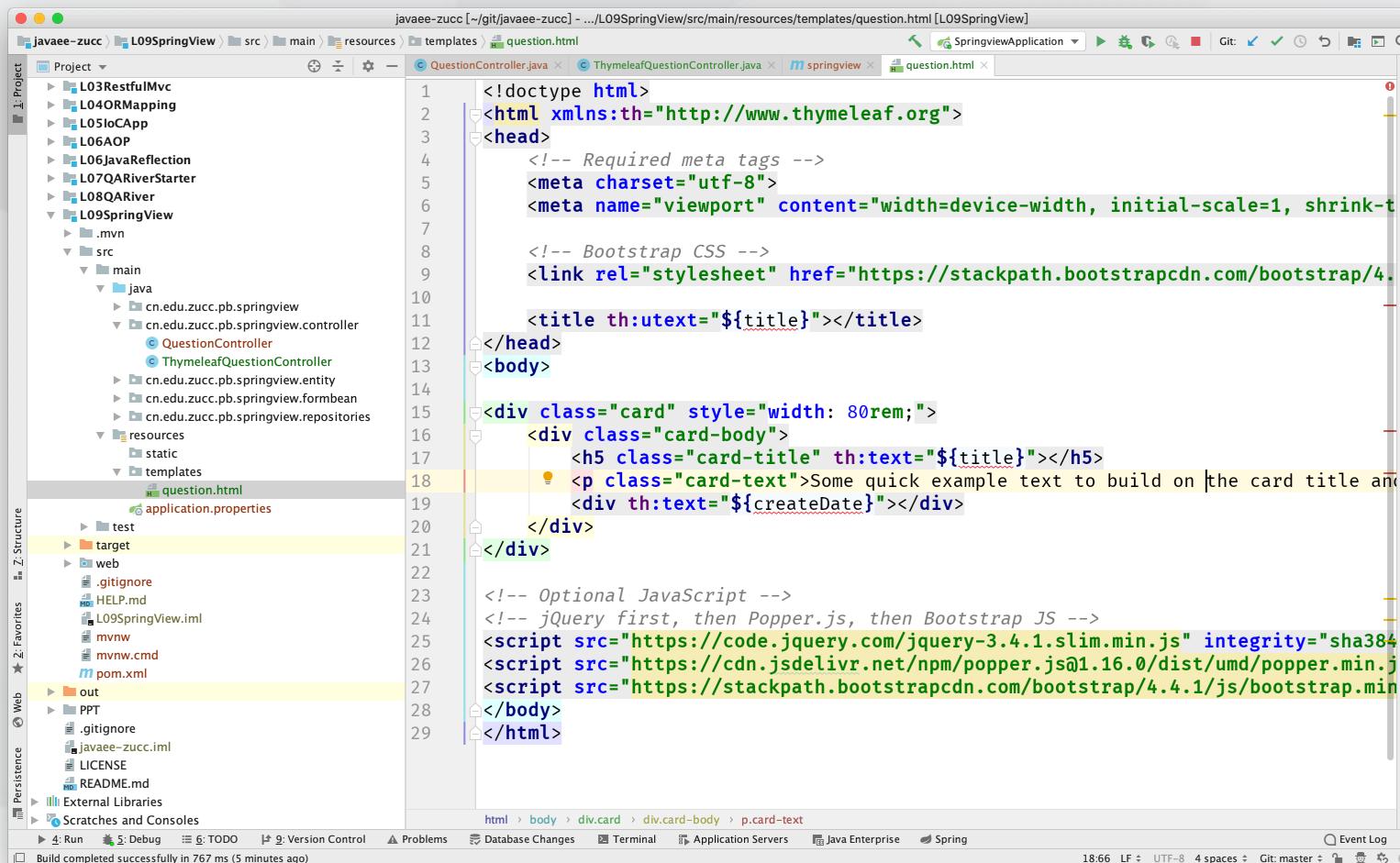
View实现技术~以Thymeleaf为例

如果使用默认的Thymeleaf配置，那么我们不需要使用任何配置，基本规则如下：

- 1) Thymeleaf文件是符合H5规范的html文件，后缀名html
- 2) 默认放到工程resources/template目录下
- 3) controller处理方法返回的字符串可以直接映射为对应的template文件

View实现技术~以Thymeleaf为例

模板文件，具体语法可以参考Thymeleaf官网



The screenshot shows an IDE interface with the following details:

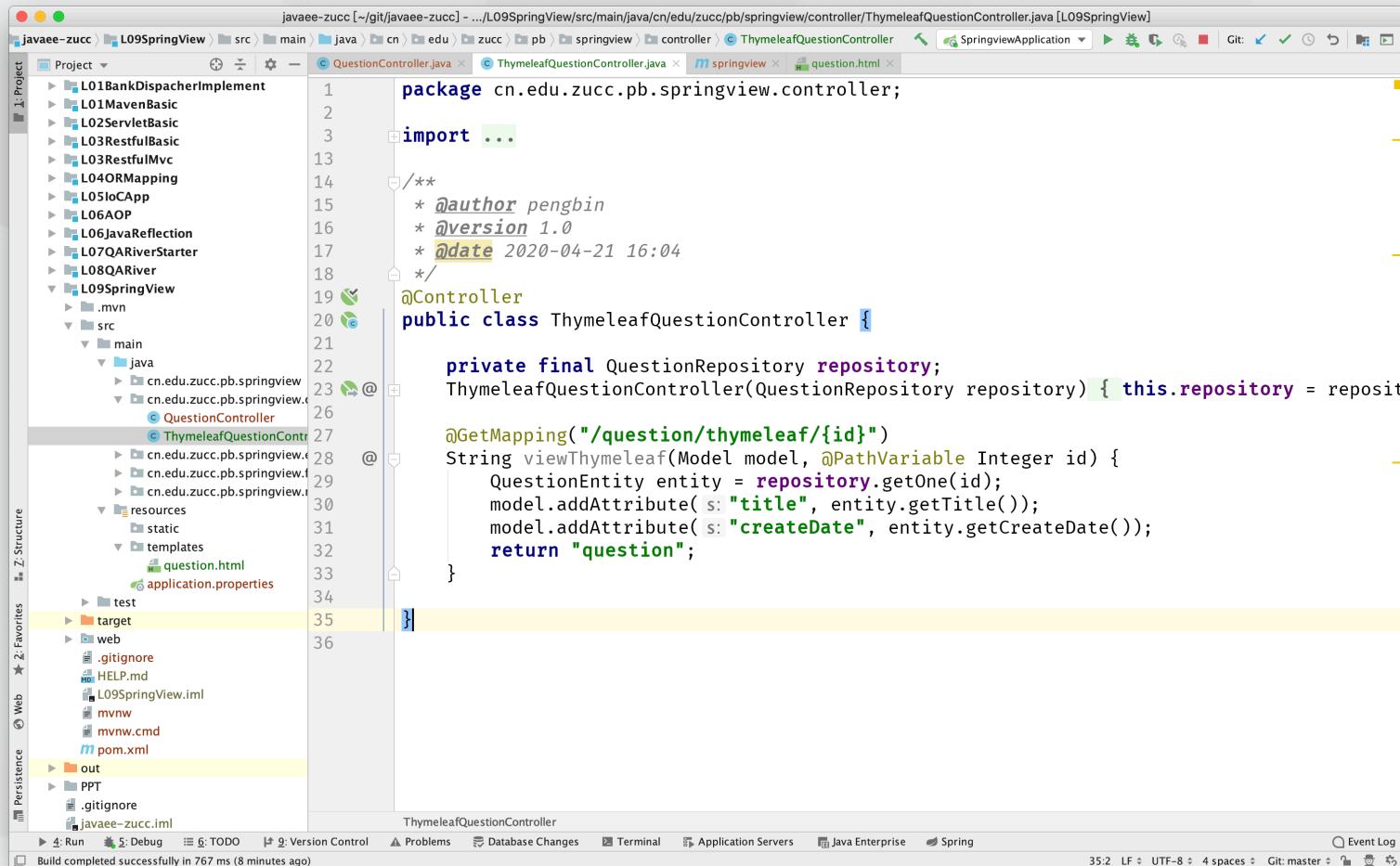
- Project Structure:** The project is named "L09SpringView". It contains several modules like L03RestfulMvc, L04ORMapping, etc., and a "src" directory with "main" and "resources" sub-directories.
- Code Editor:** The "question.html" file is open. It contains Thymeleaf syntax for generating HTML. Key parts include:
 - HTML declaration: `<!doctype html>`
 - Namespace definition: `<html xmlns:th="http://www.thymeleaf.org">`
 - Meta tags: `<meta charset="utf-8">` and `<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=false">`
 - Bootstrap CSS link: `<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" th:href="@{https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css}" />`
 - Title: `<title th:utext="\${title}"></title>`
 - Body content:
 - Card title: `<h5 class="card-title" th:text="\${title}"></h5>`
 - Card text: `<p class="card-text" th:text="Some quick example text to build on the card title and make it suitable for example purposes. You can replace it with actual content."></p>`
 - Card date: `<div th:text="\${createDate}"></div>`
 - JavaScript imports: `<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-IQRWD+DZJX9JqEa8rT/+IuQfAjwvBzWVgkHdPjyGKJUkMjCnJYQ==" th:src="https://code.jquery.com/jquery-3.4.1.slim.min.js" />` and `<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" th:src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" />`
- Java Controller:** The "ThymeleafQuestionController.java" file is also visible in the editor.

几个注意点：

- 1) html上要有命名空间定义
- 2) 使用\${}作为变量应用表达式，其中变量来自于Controller中的model对象
- 3) 本例用了bootstrap作为html模板

View实现技术~以Thymeleaf为例

对应的Controller类



```
javaee-zucc [~/git/javaee-zucc] - .../L09SpringView/src/main/java/cn/edu/zucc/pb/springview/controller/ThymeleafQuestionController.java [L09SpringView]
Project 1: Project
  L01BankDispatcherImplement
  L01MavenBasic
  L02ServletBasic
  L03RestfulBasic
  L03RestfulMvc
  L04ORMapping
  L05IoCApp
  L06AOP
  L06JavaReflection
  L07QRiverStarter
  L08QRiver
  L09SpringView
    .mvn
    src
      main
        java
          cn.edu.zucc.pb.springview
          cn.edu.zucc.pb.springview.controller
            QuestionController
            ThymeleafQuestionController
          cn.edu.zucc.pb.springview.controller
        resources
        static
          templates
            question.html
          application.properties
      test
      target
    web
      .gitignore
      HELP.md
      L09SpringView.iml
      mvnw
      mvnw.cmd
      pom.xml
    out
    PPT
    .gitignore
    javaee-zucc.iml
  Persistence 2: Favorites
  Web 3: Structure
  Persistence 4: Favorites
  Web 5: Debug
  6: TODO
  7: Version Control
  Problems
  Database Changes
  Terminal
  Application Servers
  Java Enterprise
  Spring
  Event Log
  Build completed successfully in 767 ms (8 minutes ago)
```

```
package cn.edu.zucc.pb.springview.controller;

import ...

/**
 * @author pengbin
 * @version 1.0
 * @date 2020-04-21 16:04
 */
@Controller
public class ThymeleafQuestionController {

    private final QuestionRepository repository;
    ThymeleafQuestionController(QuestionRepository repository) { this.repository = repository }

    @GetMapping("/question/thymeleaf/{id}")
    String viewThymeleaf(Model model, @PathVariable Integer id) {
        QuestionEntity entity = repository.getOne(id);
        model.addAttribute("title", entity.getTitle());
        model.addAttribute("createDate", entity.getCreateDate());
        return "question";
    }
}
```

几个注意点：

- 1) 类的标注为Controller，能够默认使用Thymeleaf模板系统作为View
- 2) 对应的接口有一个Model参数，用于绑定模板参数
- 3) 返回值“question”对应“question.html”模板

View实现技术~以Thymeleaf为例

执行结果

The screenshot shows the browser's developer tools with the "Elements" tab selected. On the left, the DOM tree displays the generated HTML code. On the right, the "Styles" panel shows the computed styles for the selected element, which is a card.

DOM Tree (Elements Tab):

```
<!doctype html>
<html>
  <head>...</head>
  <body>
    <div class="card" style="width: 80rem;">
      <div class="card-body">
        <h5 class="card-title">title-fyhS</h5>
        <p class="card-text">
          Some quick example text to build on the card title and make up
          the bulk of the card's content.
        </p>
        <div>2020-04-14 17:42:57.0</div>
      </div>
    </div>
    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
    integrity="sha384-J6q44849b1E2+poT4WnKh5vZFS5Rp0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
    crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/
    popper.min.js" integrity="sha384-
    06E9RHvbIyZFJoft+2mJbhaEWldlvI9I0Yy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
    crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/
    bootstrap.min.js" integrity="sha384-
    wfSDF2E50Y2DiuUdj003uMBjnjuuD4lh7wyDliqfkjt0Uod8GCExl30g8ifwB6"
    crossorigin="anonymous"></script>
    <audio controls="controls" style="display: none;"></audio>
  ... </body> == $0
  <style type="text/css">...</style>
</html>
```

Style Inspector (Styles Tab):

The "Styles" tab shows the computed styles for the selected element, a card. The styles include:

- body { margin: 0; font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helvetica Neue", Arial, "Noto Sans", sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Noto Color Emoji"; font-size: 1rem; font-weight: 400; line-height: 1.5; color: #212529; text-align: left; background-color: #fff; }
- * { box-sizing: border-box; }
- body { display: block; margin: 0; }
- Inherited from html
- root { root.scss:5 }

The main content area shows the rendered view with the title "title-fyhS" and some placeholder text. The timestamp "2020-04-14 17:42:57.0" is also visible.

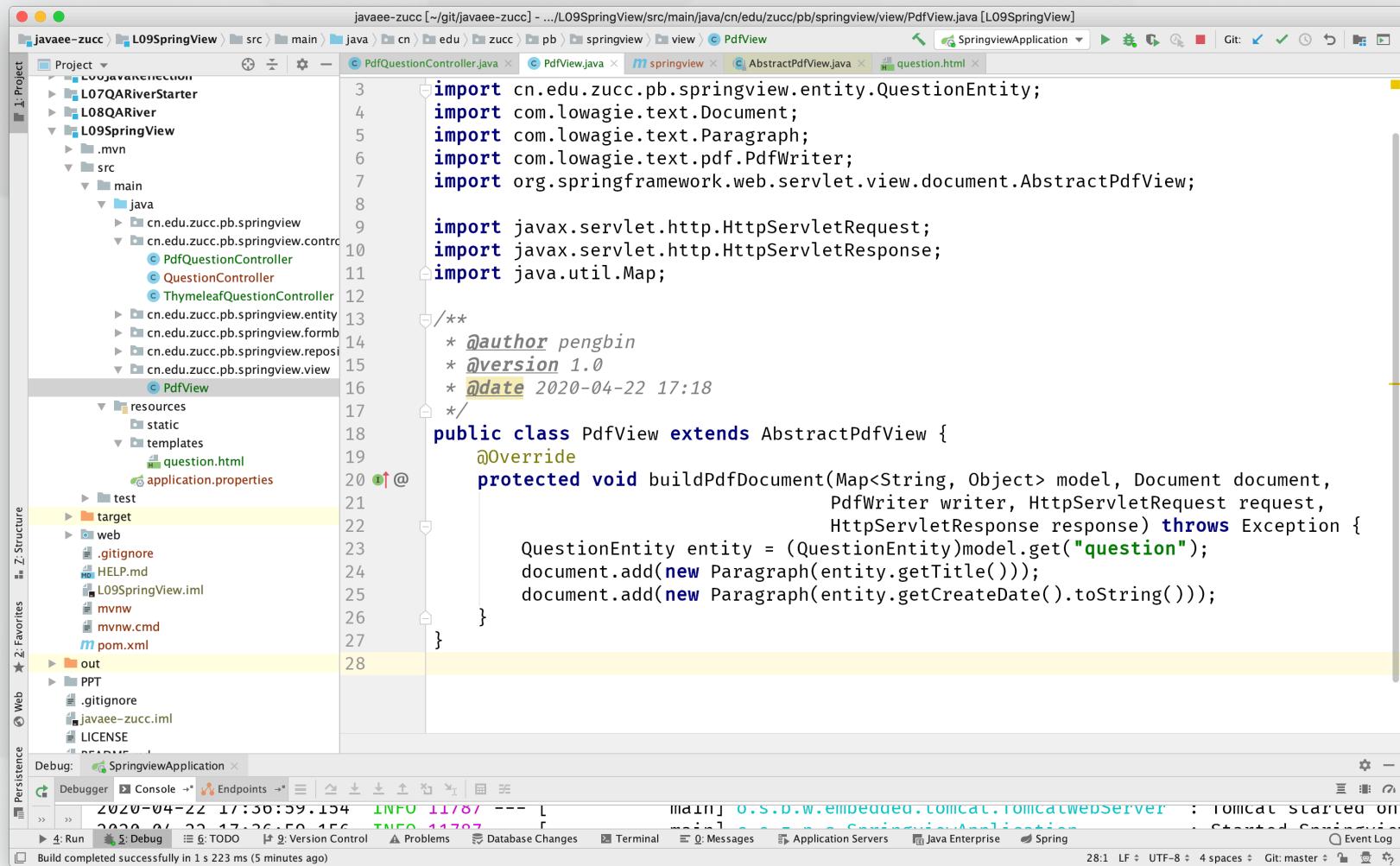
View实现技术~Pdf

如果我们要生成PDF呢，我们同样可以使用这套View机制进行构造。为了使用pdf，我们首先引入pdf生成库，然后使用标准的view生成过程就可以了

```
<dependency>
    <groupId>com.lowagie</groupId>
    <artifactId>iText</artifactId>
    <version>2.0.8</version>
</dependency>
```

View实现技术~Pdf

然后我构造一个生成PDF的view， Spring MVC提供了一个基类，用于生成PDF

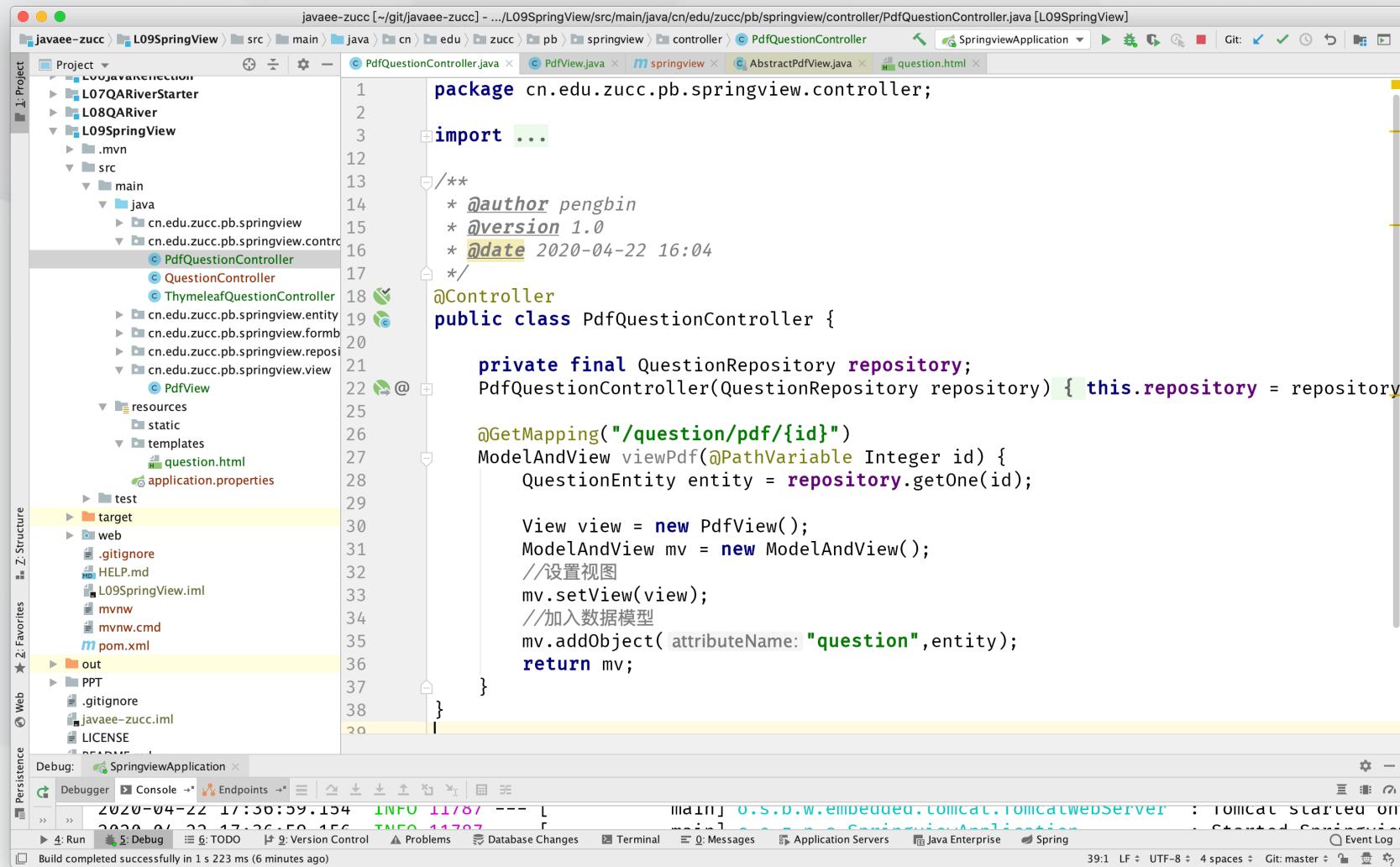


The screenshot shows the IntelliJ IDEA IDE interface with the following details:

- Project View:** Shows the project structure for "javaee-zucc" under "L09SpringView". The "src/main/java/cn/edu/zucc/pb/springview/view/PdfView.java" file is selected.
- Code Editor:** Displays the `PdfView.java` code, which extends `AbstractPdfView`. It includes imports for QuestionEntity, Document, Paragraph, PdfWriter, and AbstractPdfView, as well as HttpServletRequest and HttpServletResponse. The code overrides the `buildPdfDocument` method to add a Paragraph for the entity's title and create date.
- Toolbars and Menus:** Standard IntelliJ IDEA toolbars and menus are visible at the top.
- Bottom Status Bar:** Shows build status: "Build completed successfully in 1 s 223 ms (5 minutes ago)".

View实现技术~Pdf

然后编写一个Controller使用对应的view类，并且传递数据到view中



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** Shows the project tree with packages like L07QARiver, L08QARiver, and L09SpringView.
- Code Editor:** Displays the `PdfQuestionController.java` file. The code implements a `ModelAndView` object and returns it from the `@GetMapping` method. It uses annotations like `@Controller`, `@PathVariable`, and `@ModelAttribute`.
- Toolbars and Menus:** Standard IntelliJ IDEA toolbars and menus are visible at the top.
- Bottom Bar:** Shows the build status, terminal output, and other development tools.

```
package cn.edu.zucc.pb.springview.controller;

import ...

/**
 * @author pengbin
 * @version 1.0
 * @date 2020-04-22 16:04
 */
@Controller
public class PdfQuestionController {

    private final QuestionRepository repository;
    PdfQuestionController(QuestionRepository repository) { this.repository = repository }

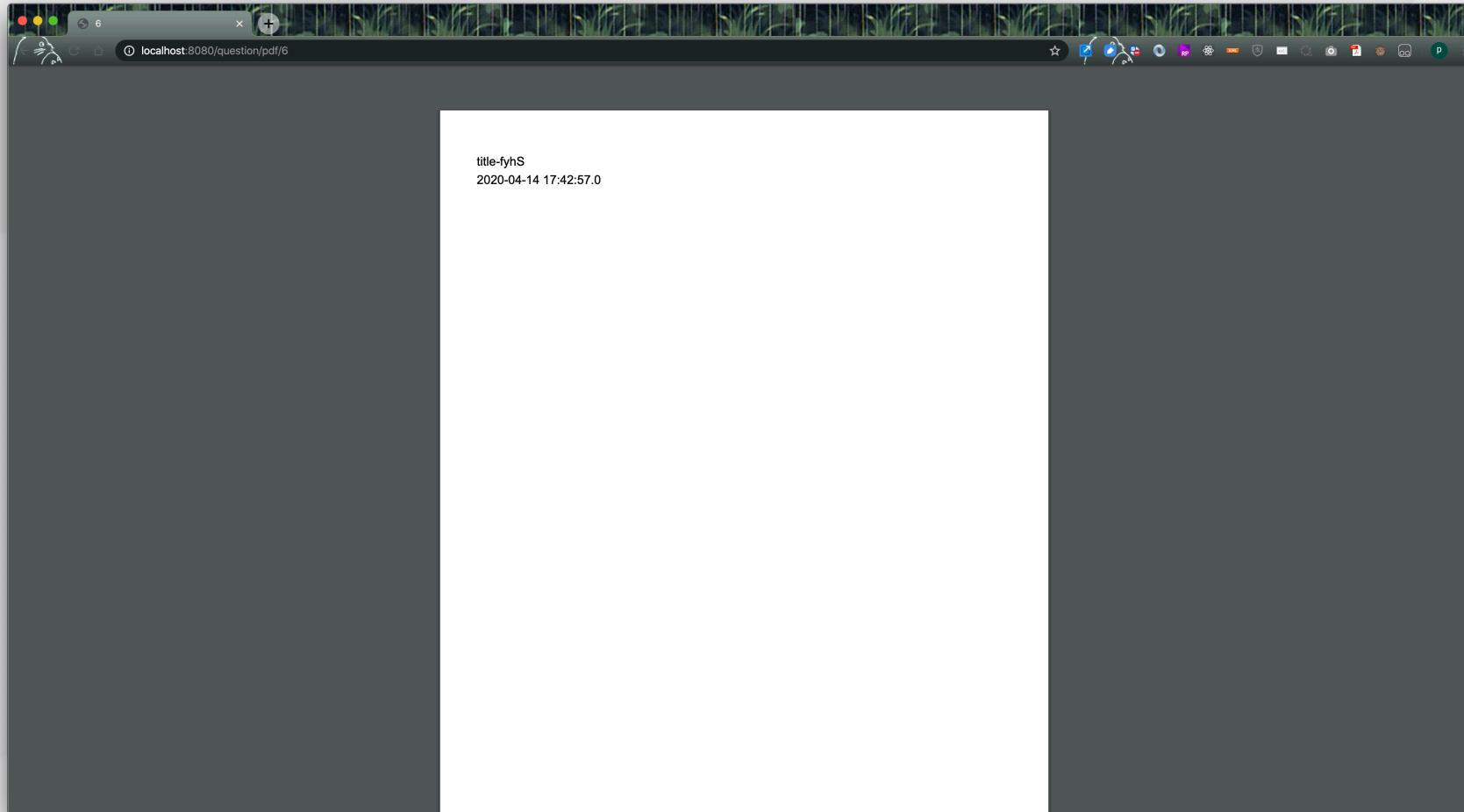
    @GetMapping("/question/pdf/{id}")
    ModelAndView viewPdf(@PathVariable Integer id) {
        QuestionEntity entity = repository.getOne(id);

        View view = new PdfView();
        ModelAndView mv = new ModelAndView();
        //设置视图
        mv.setView(view);
        //加入数据模型
        mv.addObject(attributeName: "question", entity);
        return mv;
    }
}
```

注意：返回ModelAndView类

View实现技术~Pdf

访问url，生成PDF



Spring MVC的View

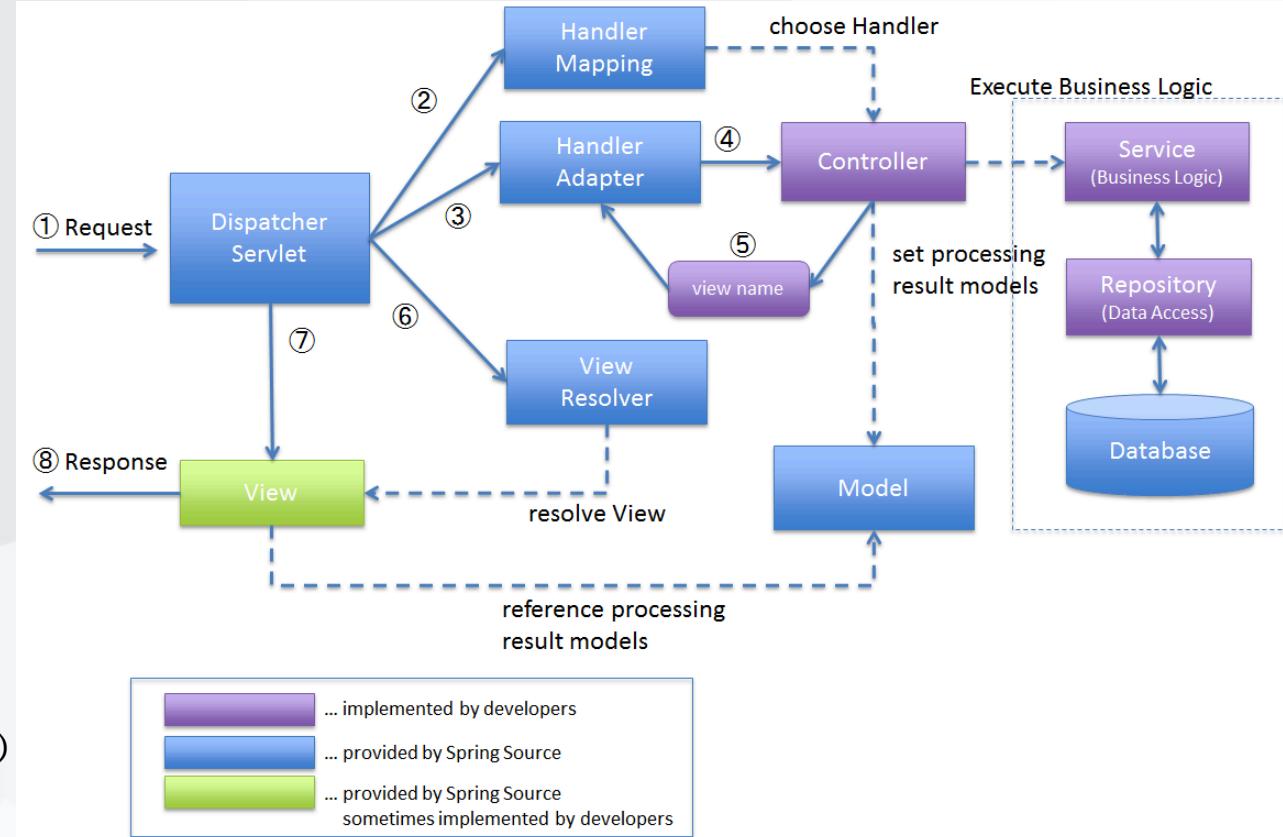
我们再次看看这个工作流程图，在Spring MVC中，使用View的基本方法为：

1) 编写Controller类，标准Url映射

2) 对应的方法返回一下类型变量

A. string:根据使用的不同模板系统，映射为对应的文件
(比如示例的Thymeleaf)

B. ModelAndView:该类携带了View和Model两种对象，分别代表View和View中使用的数据 (比如示例中的Pdf生成)



选用其他View实现系统，比如Freemarker, JSP等过程都遵循类似的过程。

Spring MVC中的View

通过上面的讨论，我们讨论Spring MVC中的View是讨论什么呢？

1) Spring MVC作为一个基于Web构建应用的框架，其提供了什么样的View的能力？

- 框架与View的实现技术分离，标注Controller，通过不同String、 ModelAndView对象，框架绑定的默认ViewResolver对象会使用不同的View实现（比如Thymeleaf模板系统， PDF生成等等）；
- 通过标注RestController可以通过默认配置的MessageConverter，完成Json或者XML和对象的转换；

2) 基于Spring MVC我们如何选择或者看待我们的View？

Spring MVC框架灵活的View架构，使得我们可以快速灵活的选择数据的表示框架，按照当前架构模式，主要的选择是：

- 使用Thymeleaf为代表的模板系统，构造良好支持服务器端生成的界面系统（方便搜索引擎搜索以及CDN发布加速）；
- 以JSON数据为代表的RESTful接口，使用多样化的前端技术构建的界面系统； WebApp（ Reactjs、Vuejs ），原生App、微信小程序。

参考文章

1. <https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html>
2. <https://github.com/ruanyf/jstraining/blob/master/docs/history.md>
3. <https://www.cnblogs.com/fangjian0423/p/springMVC-xml-json-convert.html>
4. <https://terasolunaorg.github.io/guideline/1.0.1.RELEASE/en/Overview/SpringMVCOversview.html>
5. <https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html>
6. <https://www.cnblogs.com/ityouknow/p/5833560.html>
7. https://blog.csdn.net/weixin_42218986/article/details/89846394



END

Pb&Lois