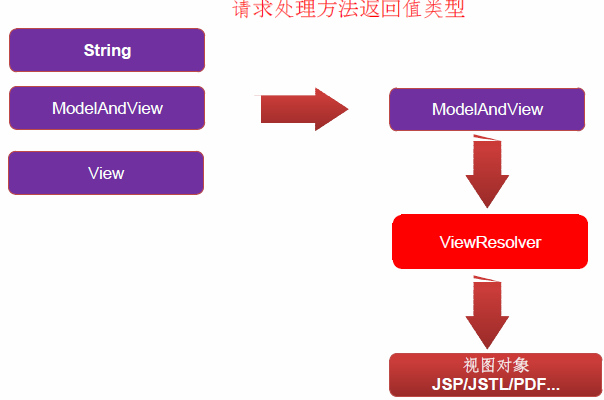
# 视图和视图解析器

## Spring MVC如何解析视图



@RequestMapping("/testViewAndViewResolver")

**public** String testViewAndViewResolver(){

System.*out*.println("testViewAndViewResolver");

**return** *SUCCESS*;

}

## 2. 视图和视图解析器

请求处理方法执行完成后，最终返回一个 ModelAndView对象。对于那些返回 String，View 或 ModeMap 等类型的处理方法，Spring MVC 也会在内部将它们装配成一个ModelAndView 对象，它包含了逻辑名和模型对象的视图

Spring MVC 借助视图解析器（ViewResolver）得到最终的视图对象（View），最终的视图可以是 JSP ，也可能是Excel、JFreeChart 等各种表现形式的视图

对于最终究竟采取何种视图对象对模型数据进行渲染，处理器并不关心，处理器工作重点聚焦在生产模型数据的工作上，从而实现 MVC 的充分解耦

## 3. 视图

视图的作用是渲染模型数据，将模型里的数据以某种形式呈现给客户。

为了实现视图模型和具体实现技术的解耦，Spring在org.springframework.web

.servlet 包中定义了一个高度抽象的 View接口：



视图对象由视图解析器负责实例化。由于视图是无状态的，所以他们不会有线程安全的问题

常用的视图实现类

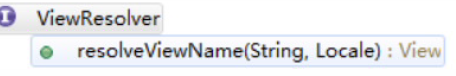


## 4. 视图解析器

SpringMVC 为逻辑视图名的解析提供了不同的策略，可以在 Spring WEB 上下文中配置一种或多种解析策略，并指定他们之间的先后顺序。每一种映射策略对应一个具体的视图解析器实现类。

视图解析器的作用比较单一：将逻辑视图解析为一个具体的视图对象。

所有的视图解析器都必须实现 ViewResolver 接口：



常用的视图解析器实现类



## 5. InternalResourceViewResolver

若项目中使用了 JSTL，则 SpringMVC 会自动把视图由InternalResourceView 转为 JstlView



若使用 JSTL 的 fmt 标签则需要在 SpringMVC 的配置文件中配置国际化资源文件

<!-- 配置国际化资源文件 -->

<bean id=*"messageSource"* class=*"org.springframework.context.support.ResourceBundleMessageSource"*>

<property name=*"basename"* value=*"i18n"*></property>

</bean>

<br>国际化：

<fmt:message key=*"i18n.username"*></fmt:message> <br>

<fmt:message key=*"i18n.password"*></fmt:message> <br>

若希望直接响应通过 SpringMVC 渲染的页面，可以使用 mvc:view-controller 标签实现

<!-- 配置直接转发的页面 -->

<!-- 可以直接相应转发的页面, 而无需再经过 Handler 的方法. -->

<mvc:view-controller path=*"/success"* view-name=*"success"*/>

<!-- 在实际开发中通常都需配置 mvc:annotation-driven 标签 -->

<mvc:annotation-driven></mvc:annotation-driven>

## 6.自定义视图

@RequestMapping("/testView")

**public** String testView(){

System.*out*.println("testView");

**return** "helloView";

}

<!-- 配置视图 BeanNameViewResolver 解析器: 使用视图的名字来解析视图 -->

<!-- 通过 order 属性来定义视图解析器的优先级, order 值越小优先级越高 -->

<bean class=*"org.springframework.web.servlet.view.BeanNameViewResolver"*>

<property name=*"order"* value=*"100"*></property>

</bean>

@Component

**public** **class** HelloView **implements** View{

@Override

**public** String getContentType() {

**return** "text/html";

}

@Override

**public** **void** render(Map<String, ?> model, HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

response.getWriter().print("hello view, time: " + **new** Date());

}

}

# 视图



# Springmvc多视图

## 导入xml格式支持jar



## 配置springmvc支持多视图

<bean class=*"org.springframework.web.servlet.view.ContentNegotiatingViewResolver"*>

<!-- 配置支持媒体类型 -->

<property name=*"contentNegotiationManager"*>

<bean class=*"org.springframework.web.accept.ContentNegotiationManagerFactoryBean"*>

<property name=*"mediaTypes"*>

<map>

<entry key=*"json"* value=*"application/json"*></entry>

<entry key=*"xml"* value=*"application/xml"*></entry>

</map>

</property>

</bean>

</property>

<!-- 指定默认视图 -->

<property name=*"defaultViews"*>

<!-- 支持多个视图 -->

<list>

<!-- 对josn格式视图支持 -->

<bean class=*"org.springframework.web.servlet.view.json.MappingJacksonJsonView"*></bean>

<!-- xml格式视图支持 -->

<bean class=*"org.springframework.web.servlet.view.xml.MarshallingView"*>

<constructor-arg>

<bean class=*"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

<property name=*"classesToBeBound"*>

<list>

<value>cn.itcast.domain.User</value>

</list>

</property>

</bean>

</constructor-arg>

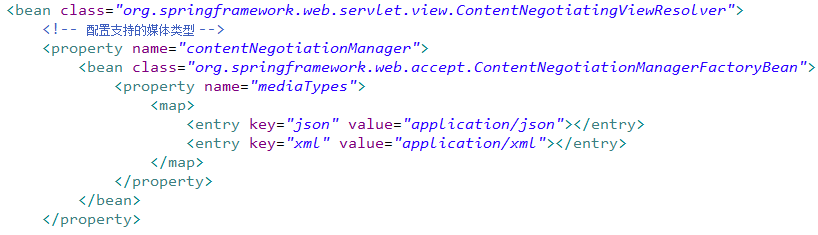
</bean>

</list>

</property>

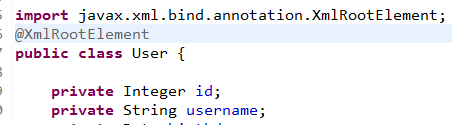
</bean>

### Springmxc.xml

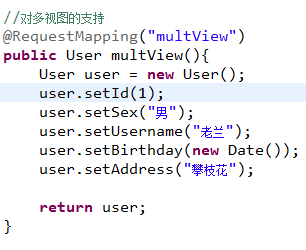




### User

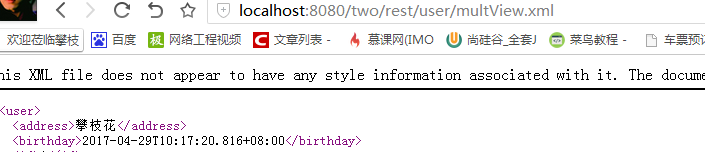


### UserCotroller



### RestFull风格访问

约定rest目录下所有以json和xml扩展名都支持相应的视图

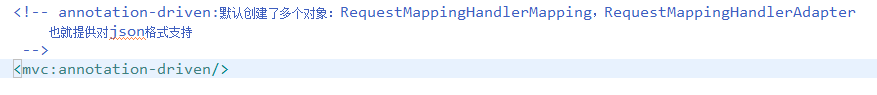




## <mvc:annotation-driven/>

<mvc:annotation-driven/>默认创建注解处理器映射器，注解处理器适配器。提供json格式支持。

在springmvc配置：



但是注意：javaBean不能添加@XmlRootElement。

@XmlRootElement提供对xml视图支持。