Spring 框架

尚硅谷 java 研究院

版本：V 1.0

第 1 章　Spring 概述

1.1 Spring 概述

1) Spring 是一个开源框架

2) Spring 为简化企业级开发而生，使用 Spring，JavaBean 就可以实现很多以前要靠 EJB 才能实现的功能。同样的功能，在 EJB 中要通过繁琐的配置和复杂的代码才能够实现，而在 Spring 中却非常的优雅和简洁。

3) Spring 是一个 **IOC**(DI)和 **AOP** 容器框架。

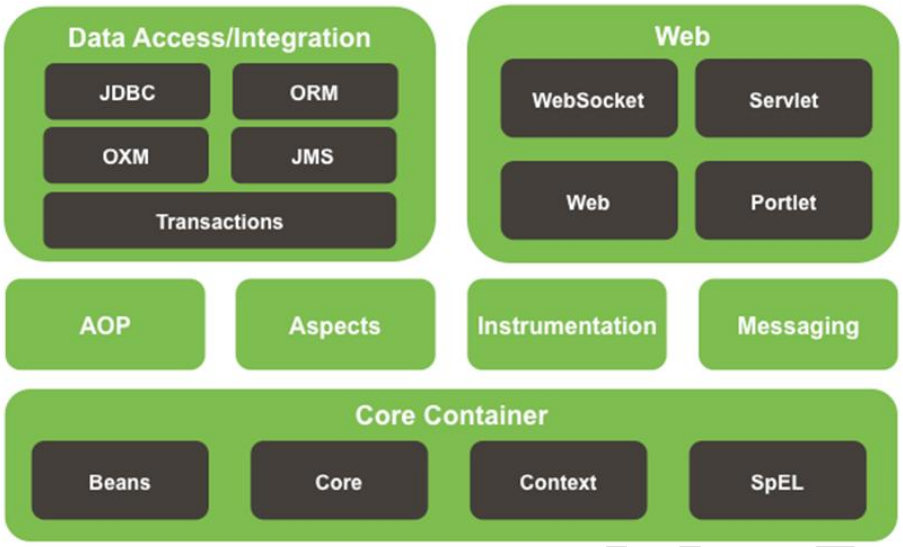
4) Spring 的优良特性

依赖注入：DI——Dependency Injection，反转控制(IOC)最经典的实现。

面向切面编程：Aspect Oriented Programming——AOP

一站式：在 IOC 和 AOP 的基础上可以整合各种企业应用的开源框架和优秀的第三方类库（实际上 Spring 自身也提供了表述层的 SpringMVC 和持久层的 Spring JDBC）。

5) Spring 模块



1.2 搭建 Spring 环境

1) 创建 Maven 版的 Java 工程

2) 加入 Spring 相关 jar 包的依赖

Tips: Spring 自身 JAR 包：

spring-beans-4.0.0.RELEASE.jar

spring-context-4.0.0.RELEASE.jar

spring-core-4.0.0.RELEASE.jar

spring-expression-4.0.0.RELEASE.jar

commons-logging-1.1.1.jar

在加入依赖时，实际只需要加入对 spring-context 的依赖即可,Maven 会根据依赖信息，将其他的 jar 包的依赖一并加入.

```
<dependencies>
    <!-- beans  -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>4.0.0.RELEASE</version>
    </dependency>
```

```
        <!-- context -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>4.0.0.RELEASE</version>
        </dependency>

        <!-- core -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>4.0.0.RELEASE</version>
        </dependency>

        <!-- expression -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-expression</artifactId>
            <version>4.0.0.RELEASE</version>
        </dependency>


    </dependencies>
```

3) 在 Spring Tool Suite 工具中通过如下步骤创建 Spring 的配置文件

① File->New->Spring Bean Configuration File

② 为文件取名字 例如：applicationContext.xml

1.3 HelloWorld

1) 目标：使用 Spring 创建对象，为属性赋值

2) 创建 HelloWorld 类

```
public class HelloWorld {

    private String name;

    public HelloWorld() {
        System.out.println("HelloWorld对象被创建了");
```

```
    }

    public void setName(String name) {
        System.out.println("name属性被赋值了");
        this.name = name;
    }

    public void sayHello() {
        System.out.println("Hello: "+name);
    }
}
```

3) 创建 Spring 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beansh
ttp://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 配置 bean
        id 属性：配置 bean 的名称，该属性值在 IOC 容器中是唯一的
        class 属性：配置 bean 的全类名，Spring 会利用反射技术实例化该 bean
    -->
    <bean id="helloWorld"
class="com.atguigu.spring.helloworld.HelloWorld">
        <!-- 通过 property 标签给 bean 的属性赋值 -->
        <property name="name" value="Spring"></property>
    </bean>
</beans>
```

4) 测试：通过 Spring 的 IOC 容器创建 HelloWorld 类实例

```
@Test
    void test() {
        //1.创建IOC容器对象
        ApplicationContext ioc = new
ClassPathXmlApplicationContext("applicationContext.xml");
        //2.从IOC容器中获取HelloWorld对象
        HelloWorld helloWorld = (HelloWorld) ioc.getBean("helloWorld");
        //3.调用HelloWorld中的sayHello方法
```

```
        helloWorld.sayHello();
    }
```

第 2 章 Spring Bean 的配置

2.1 IOC 和 DI 简介

2.1.1 IOC(Inversion of Control)：反转控制

在应用程序中的组件需要获取资源时，传统的方式是组件主动的从容器中获取所需要的资源，在这样的模式下开发人员往往需要知道在具体容器中特定资源的获取方式，增加了学习成本，同时降低了开发效率。

反转控制的思想完全颠覆了应用程序组件获取资源的传统方式：反转了资源的获取方向——改由容器主动的将资源推送给需要的组件，开发人员不需要知道容器是如何创建资源对象的，只需要提供接收资源的方式即可，极大的降低了学习成本，提高了开发的效率。这种行为也称为查找的被动形式。

2.1.2 DI(Dependency Injection)：依赖注入

IOC 的另一种表述方式：即组件以一些预先定义好的方式(例如：setter 方法)接受来自于容器的资源注入。相对于 IOC 而言，这种表述更直接。

IOC 描述的是一种思想，而 DI 是对 IOC 思想的具体实现.

2.2 Bean 配置解释

<bean>: 让 IOC 容器管理一个具体的对象.
id: 唯一标识
class: 类的全类名. 通过反射的方式创建对象.
Class cls = Class.forName("com.atguigu.spring.helloWorld.Person");

Object obj = cls.newInstance(); 无参数构造器

<property>: 给对象的属性赋值.

name: 指定属性名 ，要去对应类中的 set 方法.

value:指定属性值

2.3 获取 Bean 的方式

1) 从 IOC 容器中获取 bean 时，除了通过 id 值获取，还可以通过 bean 的类型获取。但如果同一个类型的 bean 在 XML 文件中配置了多个，则获取时会抛出异常，所以同一个类型的 bean 在容器中必须是唯一的。

```
HelloWorld helloWorld = cxt.getBean(HelloWorld.class);
```

2) 或者可以使用另外一个重载的方法，同时指定 bean 的 id 值和类型

```
HelloWorld helloWorld = cxt.getBean("helloWorld",HelloWorld.class);
```

2.4 给 bean 的属性赋值

2.4.1 普通类型的值

```
<bean id="employee" class="com.atguigu.spring.beans.Employee">
        <property name="id" value="1001"></property>
        <property name="LastName" value="大海哥"></property>
        <property name="email" value="hg@atguigu.com"></property>
        <property name="gender" value="1"></property>
</bean>
```

或者使用<value>子标签

```
<property name="id" >
            <value>1001</value>
</property>
```

2.4.2 引用类型的值

如果 Employee 类中定义了 Department 类型的成员变量.

```
<!-- 先配置Department -->
<bean id="department" class="com.atguigu.spring.beans.Department">
        <property name="id" value="101"></property>
        <property name="deptName" value="影视部"></property>
</bean>

    <!-- 再将配置好的Department注入到Employee中 -->
<bean id="employee" class="com.atguigu.spring.beans.Employee">
        <property name="dept" ref="department"></property>
</bean>
或者
<!—-
<bean id="employee" class="com.atguigu.spring.beans.Employee">
        <property name="dept">
            <ref bean="department"/>
        </property>
</bean>
-->
```

2.5 引用外部属性文件

当 bean 的配置信息逐渐增多时，查找和修改一些 bean 的配置信息就变得愈加困难。这时可以将一部分信息提取到 bean 配置文件的外部，以 properties 格式的属性文件保存起来，同时在 bean 的配置文件中引用 properties 属性文件中的内容，从而实现一部分属性值在发生变化时仅修改 properties 属性文件即可。这种技术多用于连接数据库的基本信息的配置。

2.5.1 直接配置

```
<!-- 直接配置 -->
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource ">
    <property name="user" value="root"/>
    <property name="password" value="root"/>
    <property name="jdbcUrl" value="jdbc:mysql:///test"/>
```

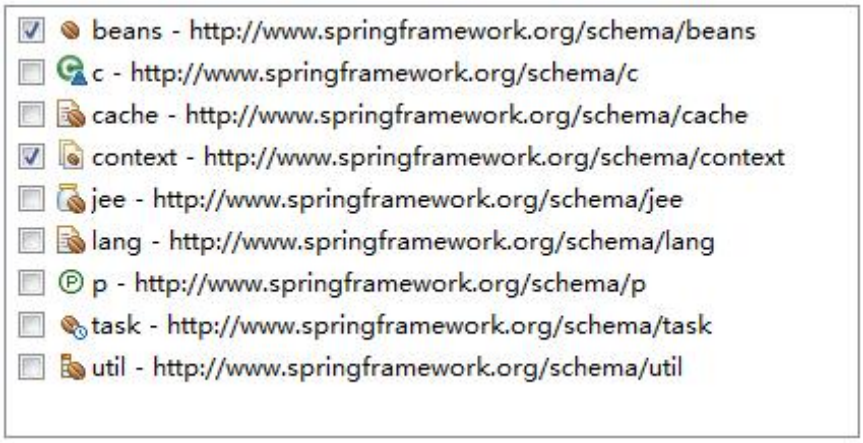
```
    <property name="driverClass" value="com.mysql.jdbc.Driver"/>
</bean>
```

2.5.2 使用外部的属性文件

1. 创建 properties 属性文件

```
prop.userName=root
prop.password=root
prop.url=jdbc:mysql:///test
prop.driverClass=com.mysql.jdbc.Driver
```

2. 引入 context 名称空间



3.指定 properties 属性文件的位置

```
<!-- 指定properties属性文件的位置 -->
<!-- classpath:xxx 表示属性文件位于类路径下 -->
<context:property-placeholder location="classpath:jdbc.properties"/>
```

4.从 properties 属性文件中引入属性值

```
<!-- 从properties属性文件中引入属性值 -->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="user" value="${prop.userName}"/>
    <property name="password" value="${prop.password}"/>
    <property name="jdbcUrl" value="${prop.url}"/>
    <property name="driverClass" value="${prop.driverClass}"/>
</bean>
```

第 3 章 基于注解配置 Bean

3.1 自动装配

3.1.1 自动装配的概念

1) 手动装配：以 value 或 ref 的方式**明确指定属性值**都是手动装配。
2) 自动装配：根据指定的装配规则，**不需要明确指定**，Spring **自动**将匹配的属性值**注入** bean 中。

3.1.2 装配模式

1) 根据**类型**自动装配：将类型匹配的 bean 作为属性注入到另一个 bean 中。若 IOC 容器中有多个与目标 bean 类型一致的 bean，Spring 将无法判定哪个 bean 最合适该属性，所以不能执行自动装配
2) 根据**名称**自动装配：必须将目标 bean 的名称和属性名设置的完全相同
3) 通过构造器自动装配：当 bean 中存在多个构造器时，此种自动装配方式将会很复杂。不推荐使用。

3.1.3 选用建议

相对于使用注解的方式实现的自动装配，在 XML 文档中进行的自动装配略显笨拙，在项目中更多的使用注解的方式实现。

3.2 通过注解配置 bean

3.2.1 概述

相对于 XML 方式而言，通过注解的方式配置 bean 更加简洁和优雅，而且和 MVC 组件化开发的理念十分契合，是开发中常用的使用方式。

3.2.2 使用注解标识组件

1) 普通组件：@Component
   标识一个受 Spring IOC 容器管理的组件
2) 持久化层组件：@Repository
   标识一个受 Spring IOC 容器管理的持久化层组件
3) 业务逻辑层组件：@Service
   标识一个受 Spring IOC 容器管理的业务逻辑层组件
4) 表述层控制器组件：@Controller
   标识一个受 Spring IOC 容器管理的表述层控制器组件
5) 组件命名规则
   ①默认情况：使用组件的简单类名首字母小写后得到的字符串作为 bean 的 id
   ②使用组件注解的 value 属性指定 bean 的 id

注意：事实上 Spring 并没有能力识别一个组件到底是不是它所标记的类型，即使将 @Respository 注解用在一个表述层控制器组件上面也不会产生任何错误，所以

@Respository、@Service、@Controller 这几个注解仅仅是为了让开发人员自己明确当前的组件扮演的角色。

3.2.3 扫描组件

组件被上述注解标识后还需要通过 Spring 进行扫描才能够侦测到。

1) 指定被扫描的 package

```
<context:component-scan base-package="com.atguigu.component"/>
```

2) 详细说明

①**base-package** 属性指定一个需要扫描的基类包，Spring 容器将会扫描这个基类包及其子包中的所有类。

②当需要扫描多个包时可以使用逗号分隔。

③如果仅希望扫描特定的类而非基包下的所有类，可使用 resource-pattern 属性过滤特定的类，示例：

```
<context:component-scan
    base-package="com.atguigu.component"
    resource-pattern="autowire/*.class"/>
```

④包含与排除

●<context:include-filter>子节点表示要包含的目标类

注意：通常需要与 use-default-filters 属性配合使用才能够达到“仅包含某些组件”这样的效果。即：通过将 use-default-filters 属性设置为 false，禁用默认过滤器，然后扫描的就只是 include-filter 中的规则指定的组件了。

●<context:exclude-filter>子节点表示要排除在外的目标类

●component-scan 下可以拥有若干个 include-filter 和 exclude-filter 子节点

●过滤表达式

类别	示例	说明
annotation	com.atguigu.XxxAnnotation	过滤所有标注了 XxxAnnotation 的类。这个规则根据目标组件是否标注了指定类型的注解进行过滤。
assignable	com.atguigu.BaseXxx	过滤所有 BaseXxx 类的子类。这个规则根据目标组件是否是指定类型的子类的方式进行过滤。

3) JAR 包依赖

需要加入对 spring-aop 的依赖，但因为 spring-context 已经对 spring-aop 进行了依赖，因此不需要重复加入.

3.2.4 组件装配

1) 需求

Controller 组件中往往需要用到 Service 组件的实例，Service 组件中往往需要用到 Repository 组件的实例。Spring 可以通过注解的方式帮我们实现属性的装配。

2) 实现依据[了解]

在指定要扫描的包时，<context:component-scan> 元素会自动注册一个 bean 的后置处理器：AutowiredAnnotationBeanPostProcessor 的实例。该后置处理器可以 自动装配标记了**@Autowired**、@Resource 或@Inject 注解的属性。

3) @Autowired 注解

①根据类型实现自动装配。

②构造器、普通字段(即使是非 public)、一切具有参数的方法都可以应用@Autowired 注解

③默认情况下，所有使用@Autowired 注解的属性都需要被设置。当 Spring 找不到匹配的 bean 装配属性时，会抛出异常。

④若某一属性允许不被设置，可以设置@Autowired 注解的 required 属性为 false

⑤默认情况下，当 IOC 容器里存在多个类型兼容的 bean 时，Spring 会尝试匹配 bean 的 id 值是否与变量名相同，如果相同则进行装配。如果 bean 的 id 值不相同，通过类 型的自动装配将无法工作。此时可以在@Qualifier 注解里提供 bean 的名称。Spring 甚至允许在方法的形参上标注@Qualifiter 注解以指定注入 bean 的名称。

⑥@Autowired 注解也可以应用在数组类型的属性上，此时 Spring 将会把所有匹配的 bean 进行自动装配。

⑦@Autowired 注解也可以应用在集合属性上，此时 Spring 读取该集合的类型信息，然后自动装配所有与之兼容的 bean。

⑧@Autowired 注解用在 java.util.Map 上时，若该 Map 的键值为 String，那么 Spring 将自动装配与值类型兼容的 bean 作为值，并以 bean 的 id 值作为键。

4) @Resource

@Resource 注解要求提供一个 bean 名称的属性，若该属性为空，则自动采用标注处的变量或方法名作为 bean 的名称。

5) @Inject

@Inject 和@Autowired 注解一样也是按类型注入匹配的 bean，但没有 reqired 属性。

第 4 章 Spring Web MVC (SpringMVC)

4.1 SpringMVC 概述

1) 一种轻量级的、基于 MVC 的 Web 层应用框架。偏前端而不是基于业务逻辑层。Spring

框架的一个后续产品。

2) Spring 为展现层提供的基于 MVC 设计理念的优秀的 Web 框架，是目前最主流的 MVC 框架之一

3) Spring MVC 通过一套 MVC 注解，让 POJO 成为处理请求的控制器，而无须实现任何接口

4.2 SpringMVC HelloWorld

1) 创建 Maven 版的 Web 工程
2) 在讲 Spring 时导入 的 jar 包依赖的基础上，加入对 web 相关 jar 包的依赖

```
<dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>4.0.0.RELEASE</version>
</dependency>
```

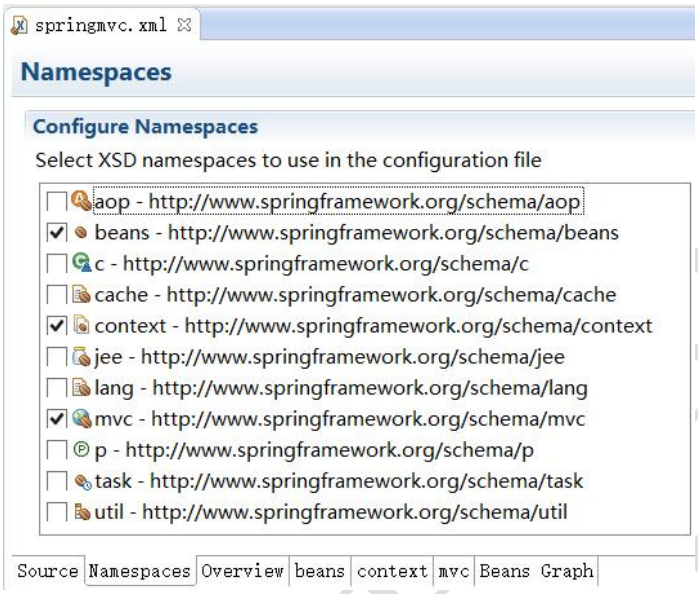
Tips: 实际需要加入 spring-web 与 spring-webmvc 的 jar 包，因为 spring-webmvc 依赖了 spring-web, Maven 会自动维护此依赖，因此只需加入对 spring-webmvc 的依赖.

3) 在 web.xml 中配置 DispatcherServlet

```
<!-- 前端控制器/核心控制器 :DispatcherServlet -->
    <servlet>
        <servlet-name>springDispatcherServlet</servlet-name>

    <servlet-class>org.springframework.web.servlet.DispatcherServ
let</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>

    </servlet>

    <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

4) 加入 SpringMVC 的配置文件 springmvc.xml

① 增加名称空间



② 配置组件扫描 和视图解析器

```
<!-- 组件扫描 -->
    <context:component-scan
base-package="com.atguigu.springmvc"></context:component-scan
>

    <!-- 视图解析器 -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceV
iewResolver">
        <property name="prefix"
value="/WEB-INF/views/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
```

5) 创建一个入口页面，index.jsp

在页面中编写超链接：Hello Springmvc

6) 编写请求处理器

```
/**
 * 请求处理器
 */
@Controller
```

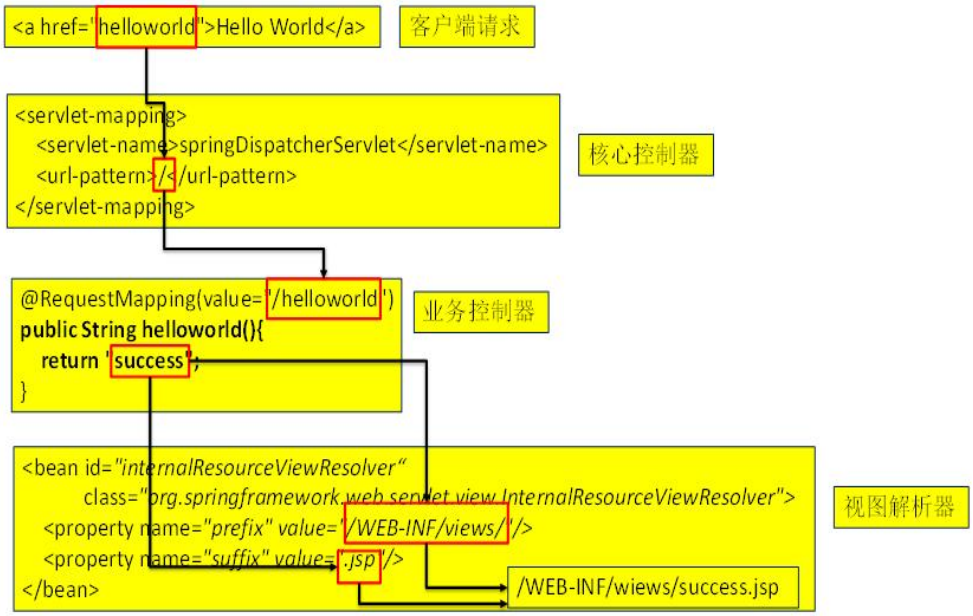
```java
public class HelloWorldHandler {

    /**
     * 请求处理方法
     * 浏览器端: http://localhost:8080/Springmvc01/hello
     */
    @RequestMapping(value="/hello")
    public  String   handleHello() {
        System.out.println("Hello Springmvc .");
        return "success";
    }


}
```

7) 编写视图页面

   在/WEB-INF/views 下新建 success.jsp 页面

8) 部署测试

9) 流程解析



4.3 @RequestMapping 映射请求注解

4.3.1 @RequestMapping 概念

1） SpringMVC 使用@RequestMapping 注解为控制器指定可以处理哪些 URL 请求

2） 作用：DispatcherServlet 截获请求后，就通过控制器上 @RequestMapping 提供的映射信息确定请求所对应的处理方法。

4.3.2 @RequestMapping 可标注的位置

4.3.3 @RequestMapping 映射请求 URL 与 请求方式

4.4 处理请求数据

4.4.1 请求处理方法签名

1） Spring MVC 通过分析处理方法的签名，将 HTTP 请求信息绑定到处理方法的相应入参中。
2） Spring MVC 对控制器处理方法签名的限制是很宽松的，几乎可以按喜欢的任何方式对方法进行签名。

4.4.2 @RequestParam 注解

1）在处理方法入参处使用 @RequestParam 可以把请求参数传递给请求方法
2）value：参数名
3）required：是否必须。默认为 true，表示请求参数中必须包含对应的参数，若不存在，将抛出异常
4）defaultValue: 默认值，当没有传递参数时使用该值

4.4.3 使用 POJO 作为参数

1） 使用 POJO 对象绑定请求参数值

2） Spring MVC 会按请求参数名和 POJO 属性名进行自动匹配，自动为该对象填充属性值。支持级联属性。如：dept.deptId、dept.address.tel 等

4.4.4 使用 Servlet 原生 API 作为参数

1） MVC 的 Handler 方法可以接受哪些 ServletAPI 类型的参数

1) HttpServletRequest
2) HttpServletResponse
3) HttpSession
4) java.security.Principal
5) Locale
6) InputStream
7) OutputStream
8) Reader
9) Write

4.5 处理响应数据

4.5.1 SpringMVC 处理响应数据概述

1） ModelAndView: 处理方法返回值类型为 ModelAndView 时，方法体即可通过该对象添加模型数据

2） Map 及 Model: 入参为 org.springframework.ui.Model、org.springframework.ui.ModelMap 或 java.uti.Map 时，处理方法返回时，Map 中的数据会自动添加到模型中。

4.5.2 处理响应数据之 ModelAndView

1） 控制器处理方法的返回值如果为 ModelAndView，则其既包含视图信息，也包含模型数据信息。

2）添加模型数据:

MoelAndView addObject(String attributeName, Object attributeValue)

ModelAndView addAllObject(Map<String, ?> modelMap)

3）设置视图:

void setView(View view)

void setViewName(String viewName)

4.5.3 处理响应数据之 Map 、Model

1）Spring MVC 在内部使用了一个 org.springframework.ui.Model 接口存储模型数据
具体使用步骤
2）Spring MVC 在调用方法前会创建一个隐含的模型对象作为模型数据的存储容器。
3）如果方法的入参为 Map 或 Model 类型，Spring MVC 会将隐含模型的引用传递给这些入参。
4）在方法体内，开发者可以通过这个入参对象访问到模型中的所有数据，也可以向模型中添加**新的属性数据**

4.5.4 @ResponseBody 注解

在 Handler 方法上添加该注解之后，方法的返回值将以字符串的形式直接响应给浏览器。

4.5.5 重定向

1) 一般情况下，控制器方法返回字符串类型的值会被当成逻辑视图名处理
2) 如果返回的字符串中带 **forward: 或 redirect:** 前缀时，SpringMVC 会对他们进行特殊处理：将 forward: 和 redirect: 当成指示符，其后的字符串作为 URL 来处理
3) redirect:success.jsp：会完成一个到 success.jsp 的重定向的操作
4) forward:success.jsp：会完成一个到 success.jsp 的转发操作

4.6 处理静态资源[了解]

1) 在 springmvc 的配置文件中加入如下两个配置:
```
<mvc:default-servlet-handler/>
<mvc:annotation-driven/>
```

第 5 章 Spring 与 Springmvc 的整合

1.1 Spring 与 SpringMVC 的整合问题：

1） 需要进行 Spring 整合 SpringMVC 吗 ？
2） 还是否需要再加入 Spring 的 IOC 容器 ？
3） 是否需要在 web.xml 文件中配置启动 Spring IOC 容器的 ContextLoaderListener ？

需要：通常情况下，类似于数据源，事务，整合其他框架都是放在 Spring 的配置文件中(而不是放在 SpringMVC 的配置文件中).
实际上放入 Spring 配置文件对应的 IOC 容器中的还有 Service 和 Dao.

不需要：都放在 SpringMVC 的配置文件中. 也可以分多个 Spring 的配置文件，然后使用 import 节点导入其他的配置文件

1.2 Spring 整合 SpringMVC_解决方案配置监听器

1） 监听器配置

```
<!-- 配置启动 Spring IOC 容器的 Listener -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:beans.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

2） 创建 Spring 的 bean 的配置文件：beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="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-4.0.xsd">

<!-- 设置扫描组件的包 -->
<context:component-scan base-package="com.atguigu.springmvc"/>

<!-- 配置数据源，整合其他框架，事务等. -->

</beans>
```

3） springmvc 配置文件：springmvc.xml

在 HelloWorldHandler、UserService 类中增加构造方法，启动服务器，查看构造器执行情况。
问题：若 Spring 的 IOC 容器和 SpringMVC 的 IOC 容器扫描的包有重合的部分，就会导致有的 bean 会被创建 2 次.
解决:

使 Spring 的 IOC 容器扫描的包和 SpringMVC 的 IOC 容器扫描的包没有重合的部分.
使用 exclude-filter 和 include-filter 子节点来规定只能扫描的注解

springmvc.xml

```
<context:component-scan base-package="com.atguigu.springmvc" use-default-filters="false">
        <context:include-filter type="annotation"
                expression="org.springframework.stereotype.Controller"/>
</context:component-scan>
```

beans.xml

```
<context:component-scan base-package="com.atguigu.springmvc">
    <context:exclude-filter type="annotation"
            expression="org.springframework.stereotype.Controller"/>
</context:component-scan>
<!-- 配置数据源, 整合其他框架, 事务等. -->
```

1.3 SpringIOC 容器和 SpringMVC IOC 容器的关系

SpringMVC 的 IOC 容器中的 bean 可以来引用 Spring IOC 容器中的 bean.
返回来呢 ? 反之则不行. Spring IOC 容器中的 bean 却不能来引用 SpringMVC IOC 容器中的 bean