**Spring框架**

版本：V 1.0

# 第1章 Spring概述

## 1.1 Spring概述

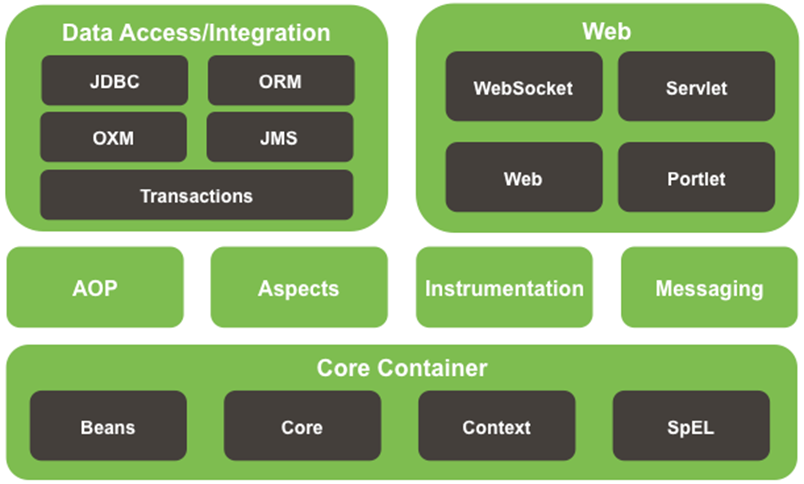
1. Spring是一个开源框架
2. Spring为简化企业级开发而生，使用Spring，JavaBean就可以实现很多以前要靠EJB才能实现的功能。同样的功能，在EJB中要通过繁琐的配置和复杂的代码才能够实现，而在Spring中却非常的优雅和简洁。
3. Spring是一个**IOC**(DI)和**AOP**容器框架。
4. Spring的优良特性

**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）。

1. 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> |

1. 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

## 1.3HelloWorld

1. 目标：使用Spring创建对象，为属性赋值
2. 创建HelloWorld类

|  |
| --- |
| **publicclass** HelloWorld {  **private** String name;    **public**HelloWorld() {  System.***out***.println("HelloWorld对象被创建了");  }    **publicvoid**setName(String name) {  System.***out***.println("name属性被赋值了");  **this**.name = name;  }    **publicvoid**sayHello() {  System.***out***.println("Hello: "+name);  }  } |

1. 创建Spring配置文件

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <beansxmlns=*"*[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)*"*      xmlns:xsi=*"*[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)*"*      xsi:schemaLocation=*"*[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)[*http://www.springframework.org/schema/beans/spring-beans.xsd*](http://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      -->      <beanid=*"helloWorld"*class=*"com.atguigu.spring.helloworld.HelloWorld"*>          <!--通过property标签给bean的属性赋值 -->          <propertyname=*"name"*value=*"Spring"*></property>      </bean>  </beans> |

1. 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  ApplicationContextioc = **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**); |

1. 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

|  |
| --- |
| <beanid=*"employee"*class=*"com.atguigu.spring.beans.Employee"*>  <propertyname=*"id"*value=*"1001"*></property>  <propertyname=*"lastName"*value=*"大海哥"*></property>  <propertyname=*"email"*value=*"hg@atguigu.com"*></property>  <propertyname=*"gender"*value=*"1"*></property>  </bean> |

或者使用<value>子标签

|  |
| --- |
| <propertyname=*"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 -->  <beanid=*"department"*class=*"com.atguigu.spring.beans.Department"*>  <propertyname=*"id"*value=*"101"*></property>  <propertyname=*"deptName"*value=*"影视部"*></property>  </bean>    <!--再将配置好的Department注入到Employee中 -->  <beanid=*"employee"*class=*"com.atguigu.spring.beans.Employee"*>  <propertyname=*"dept"*ref=*"department"*></property>  </bean>  或者  <!—-  <beanid=*"employee"*class=*"com.atguigu.spring.beans.Employee"*>  <propertyname=*"dept"*>  <refbean=*"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

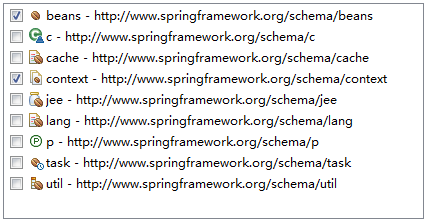
|  |
| --- |
| <!--直接配置 -->  <beanid=*"dataSource"*class=*"com.alibaba.druid.pool.DruidDataSource"*>  <propertyname=*"user"*value=*"root"*/>  <propertyname=*"password"*value=*"root"*/>  <propertyname=*"jdbcUrl"*value=*"jdbc:mysql:///test"*/>  <propertyname=*"driverClass"*value=*"com.mysql.jdbc.Driver"*/>  </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-placeholderlocation=*"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 -->  <beanid=*"dataSource"*class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>  <propertyname=*"user"*value=*"****${prop.userName}****"*/>  <propertyname=*"password"*value=*"****${prop.password}****"*/>  <propertyname=*"jdbcUrl"*value=*"****${prop.url}****"*/>  <propertyname=*"driverClass"*value=*"****${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

1. 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

1. 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

1. 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

1. 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"*/> |

1. 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。 |

1. 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。

1. 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。

1. @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。

1. @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。

1. @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

1. 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

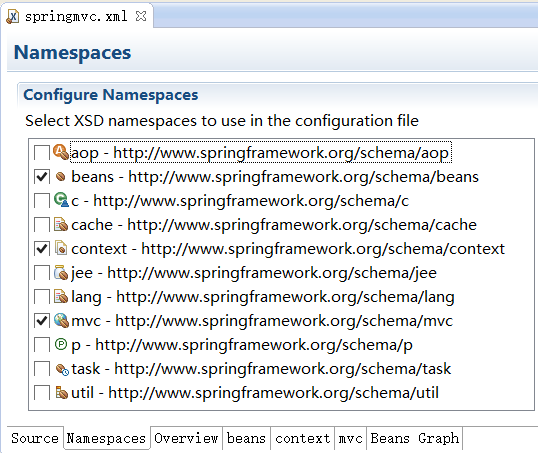
了spring-web, Maven会自动维护此依赖，因此只需加入对spring-webmvc的依

赖.

1. 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.DispatcherServlet</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> |

1. 加入SpringMVC的配置文件springmvc.xml
2. 增加名称空间



1. 配置组件扫描和视图解析器

|  |
| --- |
| <!--组件扫描 -->  <context:component-scanbase-package=*"com.atguigu.springmvc"*></context:component-scan>    <!--视图解析器 -->  <beanclass=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <propertyname=*"prefix"*value=*"/WEB-INF/views/"*></property>  <propertyname=*"suffix"*value=*".jsp"*></property>  </bean> |

1. 创建一个入口页面，index.jsp

在页面中编写超链接:<a href="${pageContext.request.contextPath }/hello">Hello Springmvc</a>

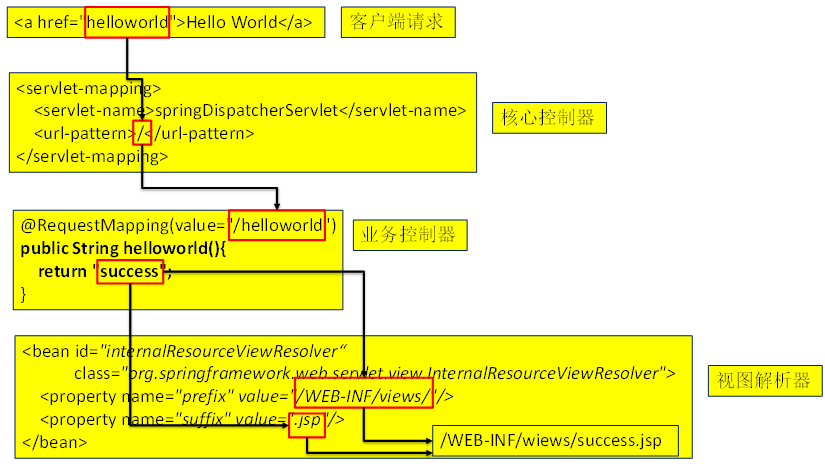
1. 编写请求处理器

|  |
| --- |
| /\*\*  \* 请求处理器  \*/  @Controller  **publicclass**HelloWorldHandler {    /\*\*  \* 请求处理方法  \* 浏览器端: http://localhost:8080/Springmvc01/hello  \*/  @RequestMapping(value="/hello")  **public** StringhandleHello() {  System.***out***.println("Hello Springmvc .");  **return**"success";  }      } |

1. 编写视图页面

在/WEB-INF/views下新建success.jsp页面

1. 部署测试
2. 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
2. HttpServletRequest
3. HttpServletResponse
4. HttpSession
5. **java.security.Principal**
6. **Locale**
7. **InputStream**
8. **OutputStream**
9. **Reader**

9) **Write**

## 4.5处理响应数据

### 4.5.1SpringMVC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:

MoelAndViewaddObject(String attributeName, Object attributeValue)

ModelAndViewaddAllObject(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> |

1. 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> |

1. 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