**尚硅谷SSM答辩面试题**

**版本：V1.0**

**尚硅谷Java技术中心**

## Maven项目依赖中作用范围scope

当一个Maven工程添加了对某个jar包的依赖后，这个被依赖的jar包可以对应下面几个可选的范围，默认 是compile。

| **依赖范围** | **编译(main)** | **测试(test)** | **部署** | **示例** |
| --- | --- | --- | --- | --- |
| **compile** | **√** | **√** | **√** | **log4j** |
| **provided** | **√** | **√** | **×** | **servlet-api lombok** |
| **test** | **×** | **√** | **×** | **junit** |
| runtime | × | √ | √ | JDBC驱动类 |
| system | √ | √ | × | 和provided相似，很少用 |
| import | × | × | × | 只能与dependencyManagement配合使用；学习SpringBoot时会使用 |

## Maven项目之间的三种关系及其特征

* 依赖关系（兄弟关系、朋友关系）：
  + 项目A依赖项目B，项目A可以访问项目B中某些依赖，避免重复提供
  + 缺点：但是只有范围为compile时可以访问
* 继承关系（父子关系，并列关系）：
  + 使用继承时，需要定义父项目和子项目
  + 继承关系是单向关系：子项目指定父项目，父项目不用指定子项目
  + 父项目的所有的依赖（compile、test、provided等）子项目都可以自动使用
  + 父项目中可以通过dependencyManagement来管理依赖，子项目如果需要必须手动声明
  + 缺点：彼此间是并列关系，父项目、子项目需要分别逐个手动进行clean、compile操作
* 聚合关系（父子关系，包含关系）：
  + 首先是继承关系，并且比继承关系更进一步
  + 在聚合关系中, 子项目明确父项目, 但是父项目明确子项目，是双向关系
  + 其实是一个大项目包含多个子项目，对父项目进行clear、compile等命令，是对所有子项目进行clear、compile命令。但是如果对一个子项目进行maven操作，不影响其他子项目

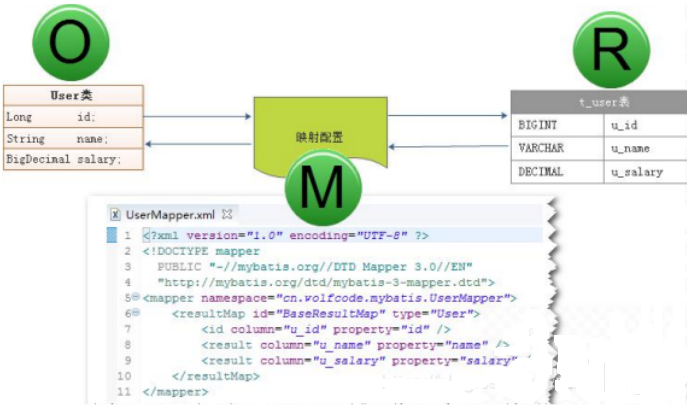
## 如何理解框架framework

* 生活案例：不需要自己盖房，直接购买毛坯房即可，自己来装修，质量户型有保证，还节省了建房的时间。在开发过程中使用的框架就好比毛坯房。
* 框架=jar（大量最佳实践基础上的对特定问题的固定解决方案进行封装并提供相应的API）+配置文件（个性化定制，配置变化的内容，比如数据库连接参数、端口号、接口的具体实现类等）
* 作用1：可以保证减少开发时间、降低开发难度，并且还保证设计质量。好比和世界上最优秀的软件工程师是一个项目的，并且他们完成的还是基础、全局的工作。想想是不是很嗨的一件事情。
* 作用2：框架还有一个作用是约束，统一了代码流程和风格。同样的技术解决同样的问题会产生不同流程和风格的解决方案，而采用一种框架其实就是限制用户必须使用其规定的方案来实现，可降低程序员之间沟通以及日后维护的成本。
* 常用的基于JavaEE的三大开源框架，已经从SSH、SSH2过渡到了SSM：SpringMVC、Spring、MyBatis。
* 总之，框架是一个半成品，已经对基础的代码进行了封装并提供相应的API，开发者在使用框架是直接调用封装好的API可以省去很多代码编写，从而提高工作效率和开发速度，并统一了风格（这一点不要忽略掉）。



## 如何理解ORM

* JDBC的缺点：需要手动的完成面向对象的Java语言、面向关系的数据库之间数据的转换，代码繁琐无技术含量，影响了开发效率（查询时需要手动的将结果集ResultSet的列数据转换为Java对象的属性；而添加操作时需要手动将Java对象的属性转换为数据库表的列字段）。
* 关于面向对象的Java语言、面向关系的数据库之间数据的转换必须要做，问题在于这个转换是否可以不由开发者来做。可以的。ORM框架就是专门来做这个问题的，相当于在面向对象语言和关系数据库之间搭建一个桥梁。
* ORM，Object-Relationl Mapping，对象关系映射，它的作用是在关系型数据库和对象之间作一个映射，这样我们在具体的操作数据库的时候，只要像平时操作对象一样操作它就可以了，ORM框架会根据映射完成对数据库的操作，就不需要再去和复杂的SQL语句打交道了。



* Hibernate是一个全自动的ORM框架。因为Hibernate创建了Java对象和数据库表之间的完整映射，可以完全以面向对象的思想来操作数据库，程序员不需要手写SQL语句。MyBatis中还需要手写SQL语句，所以是半自动化的。但是最终MyBatis却战胜了Hibernate，主要还是因为MyBatis可以更加精确的定义SQL，更加灵活，也便于优化性能。

## MyBatis中#{}和${}的区别是什么

* #{}是预编译处理，${}是字符串替换。
* Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值。
* Mybatis在处理${}时，就是把${}替换成变量的值。
* 使用#{}可以有效的防止SQL注入，提高系统安全性。

## MyBatis中当实体类中的属性名和表中的字段名不一样，怎么办

* 方法1：启用驼峰命名。前提，属性名和字段名虽然不同，但是满足驼峰命名规则。比如字段名是emp\_name、house\_type\_id，但是属性名是empName、HouseTypeId。

|  |
| --- |
| <settings>  <!-- 将xxx\_xxx这样的列名自动映射到xxXxx这样驼峰式命名的属性名 -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  </settings> |

* 方法2：起别名。数据库表的字段的别名就起为实体类中对应属性名。

|  |
| --- |
| select emp\_id as empId,emp\_name as empName,emp\_sal as salary from t\_emp |

* 方法3：进行ResultMap映射。一次性定义，重复使用，避免重复起别名。

|  |
| --- |
| <resultMap id="employeeMap" type="com.atguigu.mybatis.entity.Employee">  <!-- 使用id标签设置主键列和主键属性之间的对应关系 -->  <!-- column属性用于指定字段名；property属性用于指定Java实体类属性名 -->  <id column="emp\_id" property="empId"/>  <!-- 使用result标签设置普通字段和Java实体类属性之间的关系 -->  <result column="emp\_name" property="empName"/>  <result column="emp\_salary" property="empSalary"/>  </resultMap>  <select id="selectEmployeeByRM" resultMap="employeeMap">  select emp\_id,emp\_name,emp\_salary from t\_emp where emp\_id=#{empId}  </select> |

## MyBatis如何获取自动生成的(主)键值？如何完成MySQL的批量操作

在<insert>标签中使用 useGeneratedKeys 和 keyProperty 两个属性来获取自动生成的主键值。

例如:

|  |
| --- |
| <insert id="insertname"  usegeneratedkeys="true" keyproperty="id">      insert into names (name) values (#{name})  </insert> |

MyBatis完成MySQL的批量操作主要是通过<foreache>标签来拼装相应的SQL语句。

例如:

|  |
| --- |
| <insert id="insertBatch" >      insert into tbl\_employee(last\_name,email,gender,d\_id) values      <foreach collection="emps" item="curr\_emp" separator=",">          (#{curr\_emp.lastName},#{curr\_emp.email},#{curr\_emp.gender},#{curr\_emp.dept.id})      </foreach>  </insert> |

## 讲述下Mybatis的一级、二级缓存

1. 缓存作用

使用缓存可以减少对硬盘（数据库）的访问次数，提高访问速度。

1. 缓存分类

* 一级缓存：自动开启，默认开启。针对同一个SqlSession（不同的SqlSession有不同的一级缓存）。
* 二级缓存：手动开启，默认关闭。针对同一个SqlSessionFactory（由这一个SqlSessionFactory创建的所有的SqlSession可以共享二级缓存中的数据）。

1. 缓存访问步骤

* 先查询二级缓存，因为二级缓存中可能会有其他程序已经查出来的数据，可以拿来直接使用。
* 如果二级缓存没有命中，再查询一级缓存。
* 如果一级缓存也没有命中，则查询数据库。
* 查询数据库的结果，会立刻放入一级缓存（不会立刻放入二级缓存）。
* SqlSession关闭之前，一级缓存中的数据会写入二级缓存。

## 简述Mybatis的动态SQL，列出常用的6个标签及作用

动态SQL是MyBatis的强大特性之一 基于功能强大的OGNL表达式。

动态SQL主要是来解决查询条件不确定的情况，在程序运行期间，根据提交的条件动态的完成查询。

常用的标签：

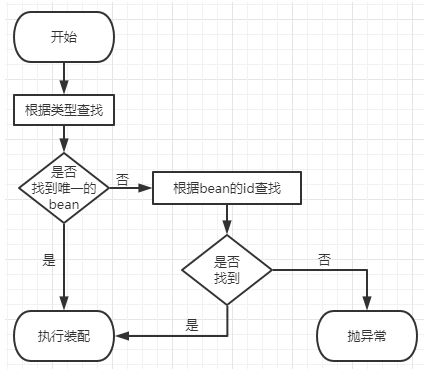
* <if> : 进行条件的判断
* <where>：在<if>判断后的SQL语句前面添加WHERE关键字，并处理SQL语句开始位置的AND 或者OR的问题
* <trim>：可以在SQL语句前后进行添加指定字符 或者去掉指定字符.
* <set>: 主要用于修改操作时出现的逗号问题
* <choose> <when> <otherwise>：类似于java中的switch语句.在所有的条件中选择其一
* <foreach>：迭代操作

## 描述下IoC依赖注入，有哪些方式

|  |
| --- |
| **控制反转（IOC）：**  拆开来解释。控制指的是对对象的生命周期的控制权。反转指的是reverse。  在IOC之前，每一层的实例对象的生命周期由程序员自己来控制。在哪里定义，在哪里初始化，何时销毁。在初始化前后需要有哪些操作，对象是否是单例。这些都由程序员自己决定。那么，现在我们使用了spring。所有的bean实例全部交由spring的IOC容器来管理。对象的生命周期，是否单例，初始化操作，后置处理器等等。  也就是，对象的控制权从程序员的手中反转给了容器去维护。这样程序员的精力更集中于业务功能的实现。  **依赖注入（DI）：**  拆开来解释。  依赖：dependency。层与层之间存在耦合，上层调用下层的方法。我们也称之为存在依赖。在DI之前，我们自己去降低耦合，通过各种手段，比如工厂模式+配置文件的方式。现在spring给我们提供了成熟的解决方案，DI。我们只需要在配置文件中(注解以及组件扫描是后来的版本中才出现的特色)配置说明实例之间的依赖关系，那么我们在运行时，spring容器会按照我们的配置文件帮我们把依赖关系逐个设置，我们称之为依赖注入（Injection）。  **注入方式**   * 通过setter方法注入 * 通过构造方法注入 |

## 解释Spring自动装配的各种模式

1. byType：首先根据所需要的组件类型到IoC容器中查找，能够找到唯一的bean，直接执行装配。如果完全找不到匹配这个类型的bean，装配失败。
2. byName：如果和所需类型匹配的bean不止一个，没有@Qualifier注解：根据@Autowired标记位置成员变量的变量名作为bean的id进行匹配，能够找到则执行装配。找不到则装配失败。
3. 使用@Qualifier注解：根据@Qualifier注解中指定的名称作为bean的id进行匹配，能够找到则执行装配，找不到则装配失败。



## 说一下Spring中支持的bean作用域

在Spring中可以通过配置bean标签的scope属性来指定bean的作用域范围，各取值含义参加下表：

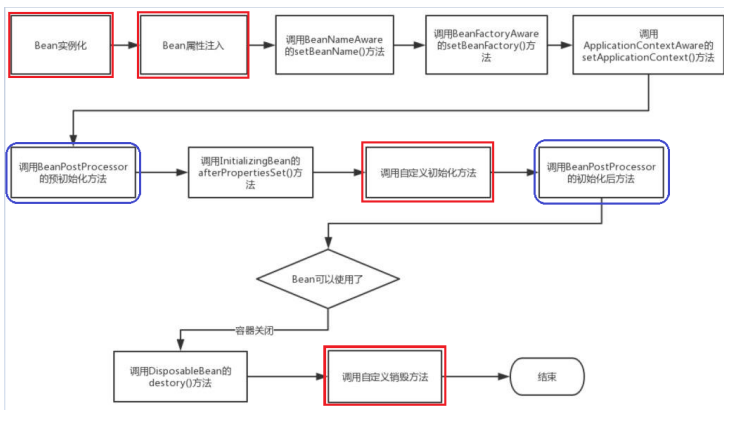
| **取值** | **含义** | **创建对象的时机** |
| --- | --- | --- |
| singleton | 在IOC容器中，这个bean的对象始终为单实例 | IOC容器初始化时 |
| prototype | 这个bean在IOC容器中有多个实例 | 获取bean时 |

 如果是在WebApplicationContext环境下还会有另外两个作用域（但不常用）：

| **取值** | **含义** |
| --- | --- |
| request | 在一个请求范围内有效 |
| session | 在一个会话范围内有效 |

## 解释Spring框架中bean的生命周期

在传统Java应用中，bean生命周期很简单，使用Java关键字 new 进行Bean 的实例化，然后该Bean 就能够使用了。一旦bean不再被使用，则由Java自动进行垃圾回收。相比之下，Spring管理Bean的生命周期就复杂多了，正确理解Bean 的生命周期非常重要，因为Spring对Bean的管理可扩展性非常强。



其中红框标记四个阶段的是基本阶段。

蓝色标记的是针对所有Bean的扩展阶段，需定义BeanPostProcessor实现类并配置。

未进行标记的是只针对当前Bean的阶段，需要当前Bean实现相应的接口。

从上图可以看出，Bean生命周期的主要阶段包括：

* bean对象创建（调用无参构造器）
* 给bean对象的属性注入值
* 给bean对象的属性注入值
* 执行Bean后置处理器中的 postProcessBeforeInitialization
* 调用初始化方法，进行初始化， 初始化方法是通过init-method来指定的
* 执行Bean的后置处理器中 postProcessAfterInitialization
* 使用
* IOC容器关闭时， 销毁Bean对象

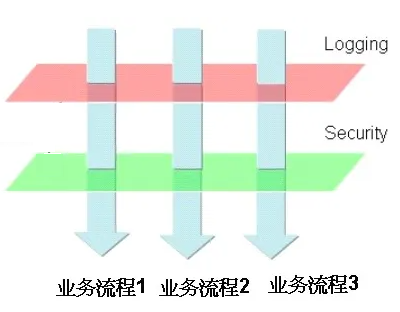
只需要回答出红色和蓝色标记节点即可，未进行标记节点也回答可适当加分。

## Spring中常用的设计模式

* 代理模式——Spring 中两种代理方式，若目标对象实现了若干接口，Spring 使用JDK的java.lang.reflect.Proxy类代理。若目标没有实现任何接口，Spring 使用 CGLIB 库生成目标类的子类。
* 单例模式——在 Spring 的配置文件中设置 bean 默认为单例模式。
* 模板方式模式——用来解决代码重复的问题。比如：RestTemplate、JmsTemplate、JpaTemplate。
* 工厂模式——在工厂模式中，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用同一个接口来指向新创建的对象。Spring 中使用 beanFactory 来创建对象的实例。

## 说下对Spring面向切面编程(AOP)的理解

* 面向切面编程AOP可以说是OOP（Object Oriented Programming，面向对象编程）的补充和完善。可以和过滤器进行类比来理解AOP，有很多相似点。
* OOP解决纵向的业务问题，AOP解决横向的问题，比如日志、安全验证、事务、异常等。相同的代码重复的出现在项目的不同位置，不利于维护。可以将这些功能分别提取出来，由多份变成一份，然后在编译或者运行时织入到指定的多个位置。
* Spring底层使用了动态代理模式实现AOP。若目标对象实现了若干接口，Spring 使用JDK的动态代理。若目标没有实现任何接口，Spring 使用 CGLIB 库生成目标类的子类。还可以配置不管是否实现接口，都是要CGLIB。



## 说下Spring面向切面编程(AOP)的常用术语

* 横切关注点

纵向关注点（核心关注点 业务关注点）：add()、sub()、findAll()、insert()，OOP解决纵向关注点。

横切关注点：日志、安全验证、事务、异常等非业务功能，代码相似却分散才项目的不同位置。AOP 解决横切关注点。

* 通知Advice（称为增强更容易理解）

就是AOP的要做的事情，比如写日志、进行事务处理。在Spring中，通知表现为一个方法。

通知的分类：前置通知，返回通知，异常通知，后置通知，环绕通知。

* 连接点 Joinpoint

在什么地方加入通知呢？比如目标类的add()、sub()、findAll()方法，在Spring中，连接点是一个方法。

* 切入点 Pointcut

一个表达式，能涵盖所有的连接点。比如：execution(\* com.atguigu.service.CalculatorImpl.\*(..))。

* 切面 Aspect

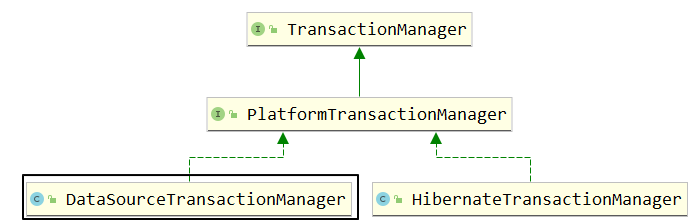
切面是一个类，包括两部分：通知（干什么）+切入点（在哪里干）。

类比过滤器：定义一个类(EncodingFilter)+过滤路径(<url-pattern>\*.html</url-pattern>)。

* 目标 target：被代理的目标对象。
* 代理 proxy：向目标对象应用通知之后创建的代理对象。
* 织入 weave：指把通知应用到目标上，生成代理对象的过程。

## 请描述一下Spring 的声明式事务管理

* 编程式事务管理：指事务功能的相关操作全部通过开发者编写代码来实现，事务代码的结构基本是确定的，暴露细节，代码重复，不利于后期维护。
* 声明式事务管理：既然事务控制的代码有规律可循，代码的结构基本是确定的，所以框架就可以将固定模式的代码抽取出来，进行相关的封装。封装起来后，只需要在配置文件中进行简单的配置即可完成操作。好处在于，可以消除冗余的代码，提高开发效率，并且框架会综合考虑相关领域中在实际开发环境下有可能遇到的各种问题，进行了健壮性、性能等各个方面的优化。
* Spring5.2后声明式事务的顶级接口是TransactionManager，它是一个空接口，真正的接口是PlatformTransactionManager。整合JDBCTemplate和MyBatis使用实现类DataSourceTransactionManager。



* 声明式事务管理包括基于XML配置和基于注解的事务管理两种，目前推荐基于注解方式。实现步骤：
  + 添加依赖：spring-orm.jar，依赖于spring-tx.jar和spring-jdbc.jar
  + 配置事务管理器：DataSourceTransactionManager
  + 启用事务的注解驱动：<tx:annotation-driven></tx:annotation-driven>
  + 使用@Transactional注解标注方法或者类

## SpringMVC中如何实现请求域request中传递数据

* 使用原生API
* 使用Model
* 使用Map
* 使用ModelMap
* 使用ModelAndView

注意：传入的Model、ModelMap、Map类型的数的三种方式其实本质上都是 BindingAwareModelMap 类型的，而四种非原生方式底层都调用了原生的request.setAttribute(name,value)。

## SpringMVC怎么样设定重定向和转发的

* 方式1：默认使用转发，并添加前缀和后缀，将逻辑路径转换为物理路径。

|  |
| --- |
| return "admin/index"; // 转发到WEB-INF/templates/index.html |

* 方式2：显式指定forward来使用转发，不再添加前缀后缀，后面的路径就是要跳转的物理路径。

|  |
| --- |
| return "forward:/add.html";  return "forward:/admin/findAll"; //跳转到Controller的方法  return "forward:/WEB-INF/templates/index.html"; |

* 方式3：显式指定redirect来使用重定向，可以重定向到互联网任意可以访问的位置。

|  |
| --- |
| return "redirect:/add.html";  return "redirect:/admin/findAll"; //跳转到Controller的方法  return "redirect:http://www.atguigu.com"; |

## 说出Spring/SpringMVC常用的6个注解，并简单描述作用

|  |  |
| --- | --- |
| @Component | 基本注解，标识一个受Spring管理的组件 |
| @Controller | 标识为一个表示层的组件 |
| @Service | 标识为一个业务层的组件 |
| @Repository | 标识为一个持久层的组件 |
| @Configuration | 标识为一个配置类 |
| @Bean | 标记一个方法，将方法返回的对象交给Spring容器管理，bean id为方法名 |
| @Autowired | 自动装配 |
| @Qualifier("xxx" ) | 具体指定要装配的组件的id值 |
| @RequestMapping() | 完成请求映射 |
| @PathVariable | Restful风格映射请求URL中占位符到请求处理方法的形参 |
| @RequestBody | 标识从请求体中获取请求数据 |
| @ResponseBody | 标识返回结果直接写入HTTP响应体中，一般在Ajax异步获取数据时使用 |

## SpringMVC中如何实现Restful风格的数据传输和接收

1. web.xml中添加过滤器org.springframework.web.filter.HiddenHttpMethodFilter，负责根据表单项的取值情况将POST请求转换为PUT请求或者DELETE。
2. 视图中页面上使用Restful的地址提交请求（不管是Ajax还是非Ajax请求均可）。
3. Controller中方法使用@GetMapping/@PostMapping/@PutMapping/@DeleteMapping指定映射路径，如果包含路径变量使用{}来表示，方法参数前使用@PathVariable注解将路径变量和参数绑定。

|  |
| --- |
| <filter>  <filter-name>hiddenHttpMethodFilter</filter-name>  <filter-class>org.springframework.web.filter.**HiddenHttpMethodFilter**</filter-class> </filter> <filter-mapping>  <filter-name>hiddenHttpMethodFilter</filter-name>  <url-pattern>/\*</url-pattern> </filter-mapping> |
| <a th:href="@{**/emp/zh/300}**">按照姓名和工资查询员工1</a><br>  <form th:action="@{/emp}" method="post">   **<input type="hidden" name="\_method" value="put">**  员工姓名:<input type="text" name="empId" value="2" readonly><br>  员工姓名:<input type="text" name="empName" value="张三"><br>  员工薪水:<input type="text" name="empSalary" value="123.45"><br>  <input type="submit" value="提交"> </form> |
| @Controller @RequestMapping("/emp") @Slf4j public class EmployeeController {  **@GetMapping("/{empName}/{salary}")**  public String findEmp(**@PathVariable("empName")** String ename,**@PathVariable** Double salary){  *log*.debug("--EmployeeController findEmp empName="+ename+",salary="+salary+"--");  return "result";  }   **@PutMapping**  public String updateEmp(Employee employee, HttpServletRequest request){  *log*.debug("method:"+request.getMethod()); //PUT  *log*.debug("-----EmployeeController updateEmp:"+employee+"-------");  return "result";  } } |

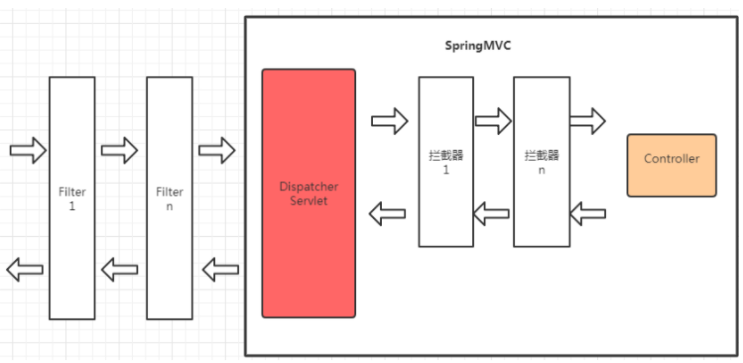
## 简述SpringMVC中如何通过Ajax请求返回JSON数据

1. 在项目中加入JSON转换的依赖，例如Jackson，Fastjson，Gson等。默认使用Jackson。如何使用其他JSON转换组件，需要在配置文件中说明，比如：

|  |
| --- |
| <mvc:annotation-driven>  <mvc:message-converters register-defaults="true">  *<!-- 配置Fastjson支持 -->* <bean class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter">  <property name="supportedMediaTypes">  <list>  <value>text/html;charset=UTF-8</value>  <value>application/json</value>  </list>  </property>  </bean>  </mvc:message-converters> </mvc:annotation-driven> |

1. 在请求处理方法中将返回值改为具体返回的数据的类型， 例如数据的集合类List<Employee>等。
2. 在请求处理方法上使用@ResponseBody注解。

## SpringMVC中拦截器和JavaEE中Filter的异同



**相似点：三要素相同**

* 拦住：必须先把请求拦住，才能执行后续操作。
* 处理：拦截器或过滤器存在的意义就是对请求进行统一处理。
* 放行：对请求执行了必要操作后，放请求过去，让它访问原本想要访问的资源；也可能不放行，比如安全验证中没有登录。

**不同点**

* 工作平台不同

过滤器工作在 Servlet 容器中，在DispatcherServlet前执行（Filter执行顺序早于所有的Servlet）。

拦截器工作在 SpringMVC 的基础上，在DispatcherServlet后执行，只有经过了DispatcherServlet的请求才有可能过拦截器。

* 拦截的范围不同

过滤器：能够拦截到的最大范围是整个 Web 应用。

拦截器：能够拦截到的最大范围是整个 SpringMVC 负责的请求（看DispatcherServlet的<url-pattern>的设置是/还是\*.action）。

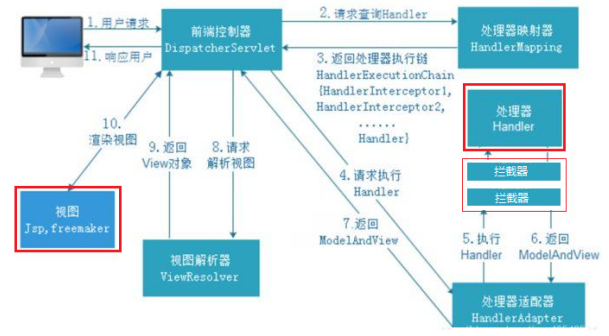
* IOC 容器支持不同

过滤器：想得到 IOC 容器需要调用专门的工具方法，是间接的。

拦截器：它自己就在 IOC 容器中，所以可以直接从 IOC 容器中装配组件。

## 简单的谈一下SpringMVC的工作流程

**执行流程图**



**执行流程**

1. 浏览器发送请求——>DispatcherServlet，前端控制器收到请求后自己不进行处理，而是委托给其他组件进行处理，作为统一访问点，进行全局的流程控制。
2. DispatcherServlet——>HandlerMapping，处理器映射器将会把请求映射为HandlerExecutionChain对象（包含一个Handler处理器对象、多个HandlerInterceptor拦截器）对象。
3. DispatcherServlet——>HandlerAdapter，处理器适配器将会把处理器包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器。
4. HandlerAdapter——>调用处理器相应功能处理方法，并返回一个ModelAndView对象（Model部分是业务对象返回的模型数据，View部分为逻辑视图名）。
5. DispatcherServlet——> ViewResolver，视图解析器将把逻辑视图名解析为物理视图，返回View对象。
6. DispatcherServlet——>View，View会根据传进来的Model模型数据进行渲染，此处的Model实际是一个Map数据结构。
7. DispatcherServlet——>响应，返回控制权给DispatcherServlet，由它响应用户，到此一个流程结束。

**总结强调**

1. 总控制器DispatcherServlet（不是撒手掌柜）每次调用其他组件要求返回结果，并根据结果调用下个组件。
2. 分控制器Handler的返回值：ModelAndView。Model中是数据，View是要跳转到的视图。
3. 以上流程针对非Ajax请求。如果是Ajax请求，就不返回ModelAndView了，而是直接返回数据片段。也就不涉及视图解析和渲染，但是涉及拦截器的执行。
4. SpringMVC流程固定，开发者的主要任务是：开发控制器类和视图文件，部分是否还要开发拦截器（图中红框标记部分）。

## 简单的谈一下SpringMVC的核心API

* DispatcherServlet：总控制器
* HandlerMapping：处理器映射器，建立了请求路径和分控制器方法之间的映射
* HandlerExecutionChain：总控制器调用HandlerMapping组件的返回值，是一个执行链，不仅有要执行的分控制器方法，还有相应的多个拦截器，组成一个执行链
* HandlerAdapter：处理器适配器，调用Handler，不是由总控制直接调用的，而是由HandlerAdapter来调用
* ViewResolver：逻辑视图（result）----->物理视图（/WEB-INF/templates/result.html）

## 说下ContextLoaderListener的作用

**常见使用场景：**

* 场景1：使用了Spring但是没有使用SpringMVC的web项目（比如Dubbo服务提供者），如何加载Spring配置文件？可以使用ContextLoaderListener来加载。
* 场景2：使用了SpringMVC但是项目规模大，有多个配置文件，除了使用DispatcherServlet一次性加载，是否有其他方法？可以使用ContextLoaderListener和DispatcherServlet分别加载。

**使用及其注意事项：**

* public class ContextLoaderListener extends ContextLoader implements ServletContextListener，该类是一个ServletContextListener，在项目启动的时候加载。注意：Servlet、Filter、Listener的加载顺序：Listener、Filter、Servlet。所以会先加载ContextLoaderListener ，再加载DispatcherServlet。
* 使用ContextLoaderListener加载非SpringMVC配置文件创建的IoC容器是父容器，DispatcherServlet加载SpringMVC的的配置文件创建的IoC容器是子容器。子容器优先使用自己的Bean，如果没有，可以使用父容器的Bean。
* 注意事项：同时使用了ContextLoaderListener和DispatcherServlet，如果<context:component-scan >的路径设置不合理，就会重复的创建Bean，甚至导致无法应用业务层事务的问题。

|  |
| --- |
| 错误的加载方式 |
| <!--ContextLoaderListener加载的配置文件：spring-tx.xml--> <context:component-scan base-package="**com.atguigu**"></context:component-scan>  <!-- DispatcherServlet加载的配置文件;springmvc.xml--> <context:component-scan base-package="**com.atguigu**"></context:component-scan> |
| 正确的加载方式 |
| <!--ContextLoaderListener加载的配置文件：spring-tx.xml--> <context:component-scan base-package="**com.atguigu.service,com.atguigu.dao**"></context:component-scan>  <!-- DispatcherServlet加载的配置文件;springmvc.xml--> <context:component-scan base-package="**com.atguigu.controller**"></context:component-scan> |

## VMWare三种工作模式

* bridged(桥接模式)：VMWare虚拟出来的操作系统就像是局域网中的一台独立的主机，它可以访问网内任何一台机器。需要手工为虚拟系统配置IP地址、子网掩码，而且还要和宿主机器处于同一网段，这样虚拟系统才能和宿主机器进行通信。
* NAT(网络地址转换模式)：让虚拟系统借助NAT(网络地址转换)功能，通过宿主机器所在的网络来访问公网。虚拟系统无法和本局域网中的其他真实主机进行通讯。
* host-only(主机模式)：所有的虚拟系统是可以相互通信的，但虚拟系统和真实的网络是被隔离开的。虚拟系统和宿主机器系统是可以相互通信的。如果想利用VMWare创建一个与网内其他机器相隔离的虚拟系统，进行某些特殊的网络调试工作，可选择host-only模式。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **和宿主机通信** | **通过宿主机联网** | **和宿主机同局域网的其他主机通信** | **其他** |
| 桥接模式（vmnet0） | 可以 | 可以 | 可以 | 和宿主的网卡地址属于一个网段 |
| NAT模式（vmnet8） | 可以 | 可以 | 不可以 | 和宿主的网卡地址不属一个网段 |
| 仅主机模式（vmnet1） | 可以 | 不可以 | 不可以 | 可以进行某些特殊的网络调试工作 |

## 写出Linux根目录下的6个常用目录及其作用

|  |  |
| --- | --- |
| /bin | 是Binary的缩写, 这个目录存放着最经常使用的命令 |
| /home（c:\users） | 存放普通用户的主目录，在Linux中每个用户都有一个自己的目录，一般该目录名是以用户的账号命名的。 |
| /root (c:\users\Administrator) | 该目录为系统管理员，也称作超级权限者的用户主目录。 |
| /lib | 系统开机所需要最基本的动态连接共享库，其作用类似于Windows里的DLL文件。几乎所有的应用程序都需要用到这些共享库。 |
| /etc(conf) | 所有的系统管理所需要的配置文件和子目录 |
| /usr(program files) | 这是一个非常重要的目录，unix software resource，用户的很多应用程序和文件都放在这个目录下， |
| /boot | 这里存放的是启动Linux时使用的一些核心文件，包括一些连接文件以及镜像文件，自己的安装别放这里 |
| /tmp | 这个目录是用来存放一些临时文件的。 |
| /dev (device) | 类似windows的设备管理器，把所有硬件用文件形式存储 |
| /opt (program files) | 这是给主机额外安装软件所摆放的目录。比如你安装一个ORACLE数据库则就可以放到这个目录下。默认是空的。 |
| /var | 这个目录中存放着在不断扩充着的东西，我们习惯将那些经常被修改的目录放在这个目录下。包括各种日志文件。 |

## Redis为什么那么快

* 完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)。
* 数据结构简单，对数据操作也简单，Redis中的数据结构是专门进行设计的。
* 采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗（Redis6.0后引入了多线程，但是为了重复利用CPU多核的优势，线程数量取决于CPU核数，而不是每个请求开启一个新线程）。
* 使用多路I/O复用模型，非阻塞IO。I/O多路复用就是通过一种机制，让一个进程可以监视多个描述符，一旦某个描述符就绪（一般是读就绪或者写就绪），能够通知程序进行相应的读写操作。常用的IO多路复用的实现有：select、poll、epoll。多路复用IO 技术最适用的是“高并发”场景。
* 使用底层模型不同，它们之间底层实现方式以及与客户端之间通信的应用协议不一样，Redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

## Redis的五大数据类型

|  |  |
| --- | --- |
| 字符串string | 字符串（一个字符串类型最大存储容量为512M） |
| 列表list | 可以重复的集合 |
| 集合set | 不可以重复的集合 |
| 哈希hash | 类似于Map<String,String> |
| 有序集合zset(sorted set） | 带分数的以分数为排序依据的有序的set |