9/1

1、Spring中自动装配的方式有哪些?

- no：不进行自动装配，手动设置Bean的依赖关系。 - byName：根据Bean的名字进行自动装配。 - byType：根据Bean的类型进行自动装配。 - constructor：类似于byType，不过是应用于构造器的参数，如果正好有一个Bean与构造器的参数类型相同则可以自动装配，否则会导致错误。 - autodetect：如果有默认的构造器，则通过constructor的方式进行自动装配，否则使用byType的方式进行自动装配。

自动装配没有自定义装配方式那么精确，而且不能自动装配简单属性（基本类型、字符串等），在使用时应注意。

2、什么是IoC和DI？并且简要说明一下DI是如何实现的?

IoC叫控制反转，是Inversion of Control的缩写，DI（Dependency Injection）叫依赖注入，是对IoC更简单的诠释。控制反转是把传统上由程序代码直接操控的对象的调用权交给容器，通过容器来实现对象组件的装配和管理。所谓的"控制反转"就是对组件对象控制权的转移，从程序代码本身转移到了外部容器，由容器来创建对象并管理对象之间的依赖关系。IoC体现了好莱坞原则 - "Don’t call me, we will call you"。依赖注入的基本原则是应用组件不应该负责查找资源或者其他依赖的协作对象。配置对象的工作应该由容器负责，查找资源的逻辑应该从应用组件的代码中抽取出来，交给容器来完成。DI是对IoC更准确的描述，即组件之间的依赖关系由容器在运行期决定，形象的来说，即由容器动态的将某种依赖关系注入到组件之中。

一个类A需要用到接口B中的方法，那么就需要为类A和接口B建立关联或依赖关系，最原始的方法是在类A中创建一个接口B的实现类C的实例，但这种方法需要开发人员自行维护二者的依赖关系，也就是说当依赖关系发生变动的时候需要修改代码并重新构建整个系统。如果通过一个容器来管理这些对象以及对象的依赖关系，则只需要在类A中定义好用于关联接口B的方法（构造器或setter方法），将类A和接口B的实现类C放入容器中，通过对容器的配置来实现二者的关联。 依赖注入可以通过setter方法注入（设值注入）、构造器注入和接口注入三种方式来实现，Spring支持setter注入和构造器注入，通常使用构造器注入来注入必须的依赖关系，对于可选的依赖关系，则setter注入是更好的选择，setter注入需要类提供无参构造器或者无参的静态工厂方法来创建对象。

3、springIOC原理是什么？如果你要实现IOC需要怎么做？请简单描述一下实现步骤?

①IoC（Inversion of Control，控制倒转）。这是spring的核心，贯穿始终。所谓IoC，对于spring框架来说，就是由spring来负责控制对象的生命周期和对象间的关系。

IoC的一个重点是在系统运行中，动态的向某个对象提供它所需要的其他对象。这一点是通过DI（Dependency Injection，依赖注入）来实现的。比如对象A需要操作数据库，以前我们总是要在A中自己编写代码来获得一个Connection对象，有了 spring我们就只需要告诉spring，A中需要一个Connection，至于这个Connection怎么构造，何时构造，A不需要知道。在系统运行时，spring会在适当的时候制造一个Connection，然后像打针一样，注射到A当中，这样就完成了对各个对象之间关系的控制。A需要依赖 Connection才能正常运行，而这个Connection是由spring注入到A中的，依赖注入的名字就这么来的。那么DI是如何实现的呢？ Java 1.3之后一个重要特征是反射（reflection），它允许程序在运行的时候动态的生成对象、执行对象的方法、改变对象的属性，spring就是通过反射来实现注入的。

举个简单的例子，我们找女朋友常见的情况是，我们到处去看哪里有长得漂亮身材又好的女孩子，然后打听她们的兴趣爱好、qq号、电话号、ip号、iq号.....，想办法认识她们，投其所好送其所要，这个过程是复杂深奥的，我们必须自己设计和面对每个环节。传统的程序开发也是如此，在一个对象中，如果要使用另外的对象，就必须得到它（自己new一个，或者从JNDI中查询一个），使用完之后还要将对象销毁（比如Connection等），对象始终会和其他的接口或类藕合起来。

②实现IOC的步骤

定义用来描述bean的配置的Java类

解析bean的配置，将bean的配置信息转换为上面的BeanDefinition对象保存在内存中，spring中采用HashMap进行对象存储，其中会用到一些xml解析技术

遍历存放BeanDefinition的HashMap对象，逐条取出BeanDefinition对象，获取bean的配置信息，利用Java的反射机制实例化对象，将实例化后的对象保存在另外一个Map中即可。

4、谈一下autowired 和resource区别是什么?

共同点

两者都可以写在字段和setter方法上。两者如果都写在字段上，那么就不需要再写setter方法。

不同点

（1）@Autowired

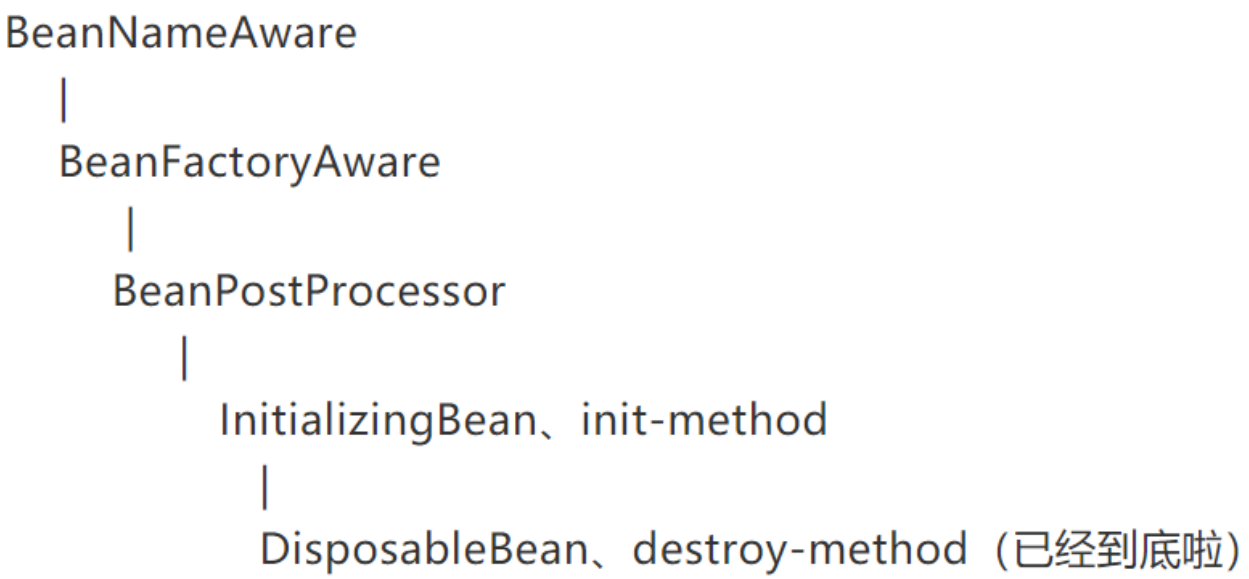
@Autowired为Spring提供的注解，需要导入包org.springframework.beans.factory.annotation.Autowired;只按照byType注入。

@Autowired注解是按照类型（byType）装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许null值，可以设置它的required属性为false。如果我们想使用按照名称（byName）来装配，可以结合@Qualifier注解一起使用。

（2）@Resource

@Resource默认按照ByName自动注入，由J2EE提供，需要导入包javax.annotation.Resource。@Resource有两个重要的属性：name和type，而Spring将@Resource注解的name属性解析为bean的名字，而type属性则解析为bean的类型。所以，如果使用name属性，则使用byName的自动注入策略，而使用type属性时则使用byType自动注入策略。如果既不制定name也不制定type属性，这时将通过反射机制使用byName自动注入策略。

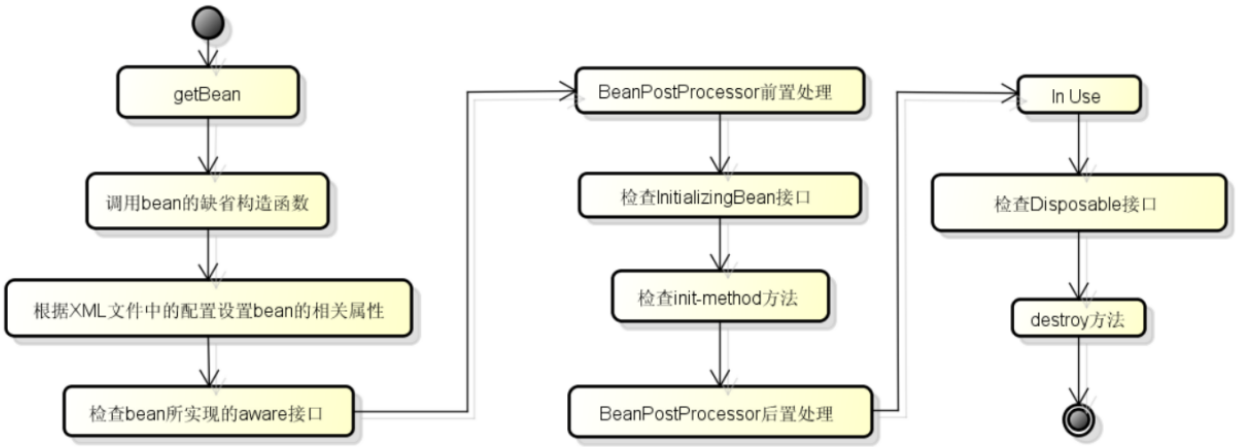
5、介绍一下bean的生命周期?



下面以BeanFactory为例，说明一个Bean的生命周期活动：

> - Bean的建立， 由BeanFactory读取Bean定义文件，并生成各个实例
> - Setter注入，执行Bean的属性依赖注入
> - BeanNameAware的setBeanName(), 如果实现该接口，则执行其setBeanName方法

> - BeanFactoryAware的setBeanFactory()，如果实现该接口，则执行其setBeanFactory方法
> - BeanPostProcessor的processBeforeInitialization()，如果有关联的processor，则在Bean初始化之前都会执行这个实例的processBeforeInitialization()方法
> - InitializingBean的afterPropertiesSet()，如果实现了该接口，则执行其afterPropertiesSet()方法
> - Bean定义文件中定义init-method
> - BeanPostProcessors的processAfterInitialization()，如果有关联的processor，则在Bean初始化之前都会执行这个实例的processAfterInitialization()方法
> - DisposableBean的destroy()，在容器关闭时，如果Bean类实现了该接口，则执行它的destroy()方法
> - Bean定义文件中定义destroy-method，在容器关闭时，可以在Bean定义文件中使用“destory-method”定义的方法



6、说明一下IOC和AOP是什么?

依赖注入的三种方式：（1）接口注入（2）Construct注入（3）Setter注入

控制反转（IoC）与依赖注入（DI）是同一个概念，引入IOC的目的：（1）脱开、降低类之间的耦合；（2）倡导面向接口编程、实施依赖倒换原则；（3）提高系统可插入、可测试、可修改等特性。

具体做法：（1）将bean之间的依赖关系尽可能地抓换为关联关系；

（2）将对具体类的关联尽可能地转换为对Java interface的关联，而不是与具体的服务对象相关联；

（3）Bean实例具体关联相关Java interface的哪个实现类的实例，在配置信息的元数据中描述；

（4）由IoC组件（或称容器）根据配置信息，实例化具体bean类、将bean之间的依赖关系注入进来。

AOP（Aspect Oriented Programming），即面向切面编程，可以说是OOP（Object Oriented Programming，面向对象编程）的补充和完善。OOP引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过OOP允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对象的核心功能毫无关系对于其他类型的代码，如安全性、异常处理和透明的持续性也都是如此，这种散布在各处的无关的代码被称为横切（cross cutting），在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。

AOP技术恰恰相反，它利用一种称为"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。

使用"横切"技术，AOP把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。

7、AOP的原理是什么?

AOP（Aspect Orient Programming），指面向方面（切面）编程，作为面向对象的一种补充，用于处理系统中分布于各个模块的横切关注点，比如事务管理、日志、缓存等等。AOP实现的关键在于AOP框架自动创建的AOP代理，AOP代理主要分为静态代理和动态代理，静态代理的代表为AspectJ；而动态代理则以Spring AOP为代表。通常使用AspectJ的编译时增强实现AOP，AspectJ是静态代理的增强，所谓的静态代理就是AOP框架会在编译阶段生成AOP代理类，因此也称为编译时增强。

Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理。JDK动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口。JDK动态代理的核心是InvocationHandler接口和Proxy类。

如果目标类没有实现接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成某个类的子类，注意，CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理的。

8、aop的应用场景有哪些?

Authentication 权限， Caching 缓存， Context passing 内容传递， Error handling 错误处理， Lazy loading 懒加载， Debugging 调试， logging, tracing, profiling and monitoring 记录跟踪　优化　校准，Performance optimization 性能优化， Persistence 持久化， Resource pooling 资源池， Synchronization同步，Transactions 事务。

9、Spring框架为企业级开发带来的好处有哪些?

- 非侵入式：支持基于POJO的编程模式，不强制性的要求实现Spring框架中的接口或继承Spring框架中的类。
- IoC容器：IoC容器帮助应用程序管理对象以及对象之间的依赖关系，对象之间的依赖关系如果发生了改变只需要修改配置文件而不是修改代码，因为代码的修改可能意味着项目的重新构建和完整的回归测试。有了IoC容器，程序员再也不需要自己编写工厂、单例，这一点特别符合Spring的精神"不要重复的发明轮子"。
- AOP（面向切面编程）：将所有的横切关注功能封装到切面（aspect）中，通过配置的方式将横切关注功能动态添加到目标代码上，进一步实现了业务逻辑和系统服务之间的分离。另一方面，有了AOP程序员可以省去很多自己写代理类的工作。
- MVC：Spring的MVC框架为Web表示层提供了更好的解决方案。
- 事务管理：Spring以宽广的胸怀接纳多种持久层技术，并且为其提供了声明式的事务管理，在不需要任何一行代码的情况下就能够完成事务管理。
- 其他：选择Spring框架的原因还远不止于此，Spring为Java企业级开发提供了一站式选择，你可以在需要的时候使用它的部分和全部，更重要的是，甚至可以在感觉不到Spring存在的情况下，在你的项目中使用Spring提供的各种优秀的功能

10、spring框架的优点都有哪些?

Spring是一个轻量级的DI和AOP容器框架，在项目的中的使用越来越广泛，它的优点主要有以下几点：

Spring是一个非侵入式框架，其目标是使应用程序代码对框架的依赖最小化，应用代码可以在没有Spring或者其他容器的情况运行。

Spring提供了一个一致的编程模型，使应用直接使用POJO开发，从而可以使运行环境隔离开来。

Spring推动应用的设计风格向面向对象及面向接口编程转变，提高了代码的重用性和可测试性。

Spring改进了结构体系的选择，虽然作为应用平台，Spring可以帮助我们选择不同的技术实现，比如从Hibernate切换到其他的ORM工具，从Struts切换到Spring MVC,尽管我们通常不会这么做，但是我们在技术方案上选择使用Spring作为应用平台，Spring至少为我们提供了这种可能性的选择，从而降低了平台锁定风险。

11、Struts拦截器和Spring AOP有什么区别?

拦截器是AOP的一种实现，struts2 拦截器采用xwork2的interceptor！而spring的AOP基于IoC基础,其底层采用动态代理与CGLIB代理两种方式结合的实现方式。

12、简单介绍一下spring?

Spring是一个轻量级框架，可以一站式构建你的企业级应用。

Spring的模块大概分为6个。分别是:

- Core Container（Spring的核心）【重要】
- AOP（面向切面变成）【重要】
- Messaging（消息发送的支持）
- Data Access/Integration（数据访问和集成）
- Web（主要是SpringWeb内容，包括MVC）【重要】
- Test（Spring测试支持，包含JUint等测试单元的支持） 7、Instrumentation（设备支持，比如Tomcat的支持）

13、持久层设计要考虑的问题有哪些？请谈一下你用过的持久层框架都有哪些?

所谓"持久"就是将数据保存到可掉电式存储设备中以便今后使用，简单的说，就是将内存中的数据保存到关系型数据库、文件系统、消息队列等提供持久化支持的设备中。持久层就是系统中专注于实现数据持久化的相对独立的层面。

持久层设计的目标包括： - 数据存储逻辑的分离，提供抽象化的数据访问接口。 - 数据访问底层实现的分离，可以在不修改代码的情况下切换底层实现。 - 资源管理和调度的分离，在数据访问层实现统一的资源调度（如缓存机制）。 - 数据抽象，提供更面向对象的数据操作。

持久层框架有： - Hibernate - MyBatis - TopLink - Guzz - jOOQ - Spring Data - ActiveJDBC

14、Mybatis和Hibernate的区别是什么?

简介

Hibernate：Hibernate是当前最流行的ORM框架之一，对JDBC提供了较为完整的封装。Hibernate的O/R Mapping实现了POJO 和数据库表之间的映射，以及SQL的自动生成和执行。

Mybatis：Mybatis同样也是非常流行的ORM框架，主要着力点在于 POJO 与 SQL 之间的映射关系。然后通过映射配置文件，将SQL所需的参数，以及返回的结果字段映射到指定 POJO 。相对Hibernate“O/R”而言，Mybatis 是一种“Sql Mapping”的ORM实现。

缓存机制对比

相同点

Hibernate和Mybatis的二级缓存除了采用系统默认的缓存机制外，都可以通过实现你自己的缓存或为其他第三方缓存方案，创建适配器来完全覆盖缓存行为。

不同点

Hibernate的二级缓存配置在SessionFactory生成的配置文件中进行详细配置，然后再在具体的表-对象映射中配置是那种缓存。

MyBatis的二级缓存配置都是在每个具体的表-对象映射中进行详细配置，这样针对不同的表可以自定义不同的缓存机制。并且Mybatis可以在命名空间中共享相同的缓存配置和实例，通过Cache-ref来实现。

两者比较

因为Hibernate对查询对象有着良好的管理机制，用户无需关心SQL。所以在使用二级缓存时如果出现脏数据，系统会报出错误并提示。而MyBatis在这一方面，使用二级缓存时需要特别小心。如果不能完全确定数据更新操作的波及范围，避免Cache的盲目使用。否则，脏数据的出现会给系统的正常运行带来很大的隐患。

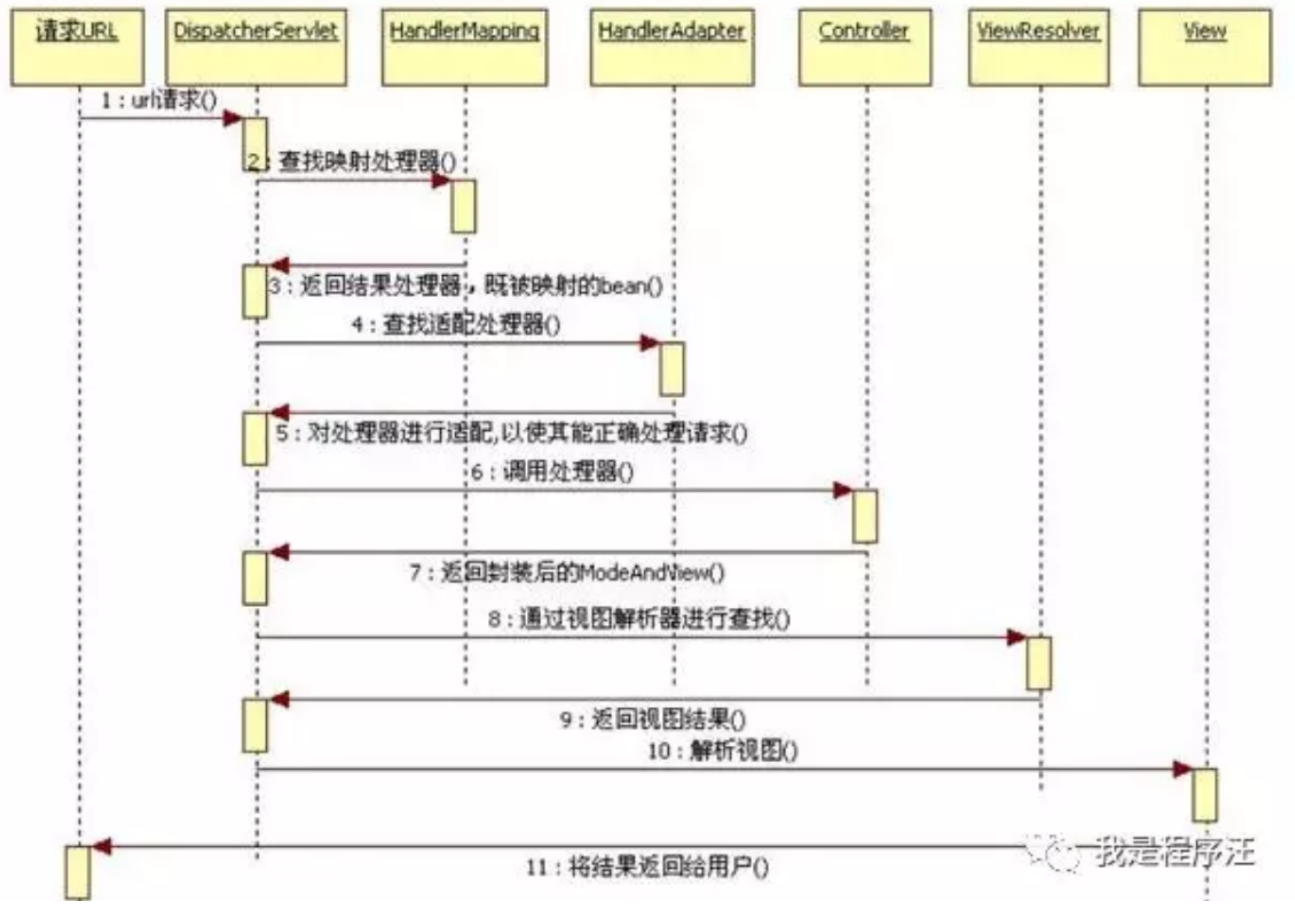
Mybatis：小巧、方便、高效、简单、直接、半自动化

Hibernate：强大、方便、高效、复杂、间接、全自动化

15、谈一下Spring MVC的工作原理是怎样的?

核心类与接口：

- DispatcherServlet 前置控制器
- HandlerMapping 请求映射（到Controller）
- HandlerAdapter 请求映射（到Controller类的方法上）
- Controller 控制器
- HandlerIntercepter 拦截器
- ViewResolver 视图映射
- View 视图处理



①客户端的所有请求都交给前端控制器DispatcherServlet来处理，它会负责调用系统的其他模块来真正处理用户的请求。 ② DispatcherServlet收到请求后，将根据请求的信息（包括URL、HTTP协议方法、请求头、请求参数、Cookie等）以及HandlerMapping的配置找到处理该请求的Handler（任何一个对象都可以作为请求的Handler）。 ③在这个地方Spring会通过HandlerAdapter对该处理器进行封装。 ④ HandlerAdapter是一个适配

器，它用统一的接口对各种Handler中的方法进行调用。 ⑤ Handler完成对用户请求的处理后，会返回一个ModelAndView对象给DispatcherServlet，ModelAndView顾名思义，包含了数据模型以及相应的视图的信息。 ⑥ ModelAndView的视图是逻辑视图，DispatcherServlet还要借助ViewResolver完成从逻辑视图到真实视图对象的解析工作。 ⑦ 当得到真正的视图对象后，DispatcherServlet会利用视图对象对模型数据进行渲染。 ⑧ 客户端得到响应，可能是一个普通的HTML页面，也可以是XML或JSON字符串，还可以是一张图片或者一个PDF文件。

16、简述一下SpringMVC的运行机制？以及运行机制的流程是什么？

- 用户发送请求时会先从DispathcherServler的doService方法开始，在该方法中会将ApplicationContext、localeResolver、themeResolver等对象添加到request中，紧接着就是调用doDispatch方法。
- 进入该方法后首先会检查该请求是否是文件上传的请求(校验的规则是是否是post并且contenttType是否为multipart/为前缀)即调用的是checkMultipart方法；如果是的将request包装成MultipartHttpServletRequest。
- 然后调用getHandler方法来匹配每个HandlerMapping对象，如果匹配成功会返回这个Handle的处理链HandlerExecutionChain对象，在获取该对象的内部其实也获取我们自定定义的拦截器，并执行了其中的方法。
- 执行拦截器的preHandle方法，如果返回false执行afterCompletion方法并理解返回
- 通过上述获取到了HandlerExecutionChain对象，通过该对象的getHandler()方法获得一个object通过HandlerAdapter进行封装得到HandlerAdapter对象。
- 该对象调用handle方法来执行Controller中的方法，该对象如果返回一个ModelAndView给DispatcherServlet。
- DispatcherServlet借助ViewResolver完成逻辑试图名到真实视图对象的解析，得到View后DispatcherServlet使用这个View对ModelAndView中的模型数据进行视图渲染。

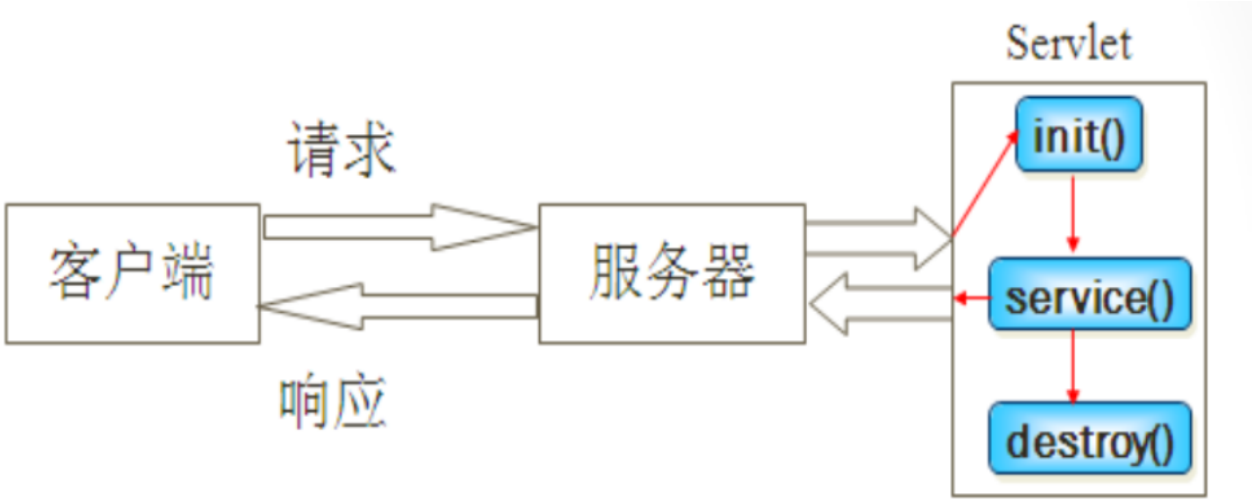
17、说明一下springmvc和spring-boot区别是什么？

总的来说，Spring 就像一个大家族，有众多衍生产品例如 Boot，Security，JPA等等。但他们的基础都是Spring 的 IOC 和 AOP，IOC提供了依赖注入的容器，而AOP解决了面向切面的编程，然后在此两者的基础上实现了其他衍生产品的高级功能；因为 Spring 的配置非常复杂，各种xml，properties处理起来比较繁琐。于是为了简化开发者的使用，Spring社区创造性地推出了Spring Boot，它遵循约定优于配置，极大降低了Spring使用门槛，但又不失Spring原本灵活强大的功能。

18、说明一下Spring MVC注解的优点是什么？

- XML配置起来有时候冗长，此时注解可能是更好的选择，如jpa的实体映射；注解在处理一些不变的元数据时有时候比XML方便的多，比如springmvc的数据绑定，如果用xml写的代码会多的多；
- 注解最大的好处就是简化了XML配置；其实大部分注解一定确定后很少会改变，所以在一些中小项目中使用注解反而提供了开发效率，所以没必要一头走到黑；
- 注解相对于XML的另一个好处是类型安全的，XML只能在运行期才能发现问题。

19、Servlet 的生命周期？



在一个Servlet的生命期中

- init()调用一次，一般第一次访问Servlet时调用
  - <load-on-startup>1</load-on-startup> 0 ,1,2,3
- service()调用多次，每次访问时调用
- destroy()调用一次，应用程序关闭时调用

- 构造方法（实例化）与 init(初始化)方法是一起执行的，要么第一次访问时执行，要么是 web 程序启动时执行。当加上1时，两个方法在 web 应用程序启动时调用，多个调用顺序 0,1,2,3。
- init()调用一次，一般第一次访问 Servlet 时调用。 service()调用多次，每次访问时调用。也有可能调用 0 次。
- destroy()调用一次， 应用程序关闭时调用（服务器关了，应用程序一定关；程序关闭，服务器不一定关）

- 实例化，或称创建， new，创建对象，分配内存空间。由容器完成。
- 初始化，指的是调用 init 方法，在对象实例化后执行。完成初始 servlet 要使用的资源，给变量赋初值等工作。
- 服务，调用 service 方法。
- 销毁， destroy,释放初始化占用的资源。
- 不可用，垃圾回收车收集内存。

20、请求转发 VS 重定向?

- 请求转发可以传递数据（数据不会丢失），重定向不能传递数据（数据会丢失）。
- 请求转发只能在一个 web 应用程序内部转发，重定向可以到其他应用程序。
- 请求转发处理完， 地址栏 url 路径为第一个（servlet）资源路径。 重定向为最后一个资源路径。 （利用重定向防止表单重复提交。）
- 请求转发不需要加 webApp 名称。重定向需要添加 webApp 名称，但如果是定向到一个程序内部，添加有些繁琐，这时可以不添加，但资源路径不能再写“/“
- 请求转发后面调用的方法类别与前面相同，而重定向全部会变为 get 请求，与之前的没关系。
- 请求转发是一次请求（只有一个 request 对象），重定向是多次请求（生成多个 request 对象）。