Java面试题-架构/框架相关

一.MVC 设计思想

MVC 是三个单词的首字母缩写，它们是 Model（模型）、View（视图）和 Controller（控制）。
这个模式认为，程序不论简单或复杂，从结构上看，都可以分成三层：

- 最上面的一层，是直接面向最终用户的"视图层"（View）。它是提供给用户的操作界面，是程序的外壳。
- 最底下的一层，是核心的"数据层"（Model），也就是程序需要操作的数据或信息。
- 中间的一层，就是"控制层"（Controller），它负责根据用户从"视图层"输入的指令，选取"数据层"中的数据，然后对其进行相应的操作，产生最终结果。

二.说说 Spring AOP?

面向切面编程，在我们的应用中，经常需要做一些事情，但是这些事情与核心业务无关，比如，要记录所有 update 方法的执行时间时间，操作人等等信息，记录到日志，
通过 Spring 的 AOP 技术，就可以在不修改 update 的代码的情况下完成该需求。

三.Spring AOP 实现原理

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。

四.Spring Bean 的生命周期

- Spring Bean 的生命周期简单易懂。在一个 bean 实例被初始化时，需要执行一系列的初始化操作以达到可用的状态。同样的，当一个 bean 不在被调用时需要进行相关的析构操作，并从 bean 容器中移除。
- Spring bean factory 负责管理在 spring 容器中被创建的 bean 的生命周期。Bean 的生命周期由两组回调（call back）方法组成。
    - 初始化之后调用的回调方法。
    - 销毁之前调用的回调方法。
- Spring 框架提供了以下四种方式来管理 bean 的生命周期事件：
    - InitializingBean 和 DisposableBean 回调接口
    - 针对特殊行为的其他 Aware 接口
    - Bean 配置文件中的 Custom init() 方法和 destroy() 方法
    - @PostConstruct 和 @PreDestroy 注解方式

五.Spring IOC 如何实现

- Spring 中的 `org.springframework.beans` 包和 `org.springframework.context` 包构成了 Spring 框架 IoC 容器的基础。
- BeanFactory 接口提供了一个先进的配置机制，使得任何类型的对象的配置成为可能。ApplicationContext 接口对 BeanFactory（是一个子接口）进行了扩展，在 BeanFactory 的基础上添加了其他功能，比如与 Spring 的 AOP 更容易集成，也提供了处理 message resource 的机制（用于国际化）、事件传播以及应用层的特别配置，比如针对 Web 应用的 WebApplicationContext。
- `org.springframework.beans.factory.BeanFactory` 是 Spring IoC 容器的具体实现，用来包装和管理前面提到的各种 bean。BeanFactory 接口是 Spring IoC 容器的核心接口。

六.Spring 框架中用到了哪些设计模式

- 代理模式：在 AOP 和 Remoting 中被用的比较多。
- 单例模式：在 Spring 配置文件中定义的 Bean 默认为单例模式。
- 模板方法：用来解决代码重复的问题。比如. RestTemplate, JmsTemplate, JpaTemplate。
- 前端控制器：Spring 提供了 DispatcherServlet 来对请求进行分发。
- 视图帮助(View Helper)：Spring 提供了一系列的 JSP 标签，高效宏来辅助将分散的代码整合在视图里。
- 依赖注入：贯穿于 BeanFactory / ApplicationContext 接口的核心理念。
- 工厂模式：BeanFactory 用来创建对象的实例。

七.Spring 的单例实现原理

Spring 对 Bean 实例的创建是采用单例注册表的方式进行实现的，而这个注册表的缓存是 ConcurrentHashMap 对象。

八.Spring 事务实现方式

1.编码方式

所谓编程式事务指的是通过编码方式实现事务，即类似于 JDBC 编程实现事务管理。

2.声明式事务管理方式

声明式事务管理又有两种实现方式：
- 基于 xml 配置文件的方式；
- 另一个实在业务方法上进行 `@Transaction` 注解，将事务规则应用到业务逻辑中；

九.Spring 事务底层原理

划分处理单元 IOC

由于 Spring 解决的问题是对单个数据库进行局部事务处理的，具体的实现首相用 Spring 中的 IOC 划分了事务处理单元。并且将对事务的各种配置放到了 IOC 容器中（设置事务管理器，设置事务的传播特性及隔离机制）。

AOP 拦截需要进行事务处理的类

Spring 事务处理模块是通过 AOP 功能来实现声明式事务处理的，具体操作（比如事务实行的配置和读取，事务对象的抽象），用 `TransactionProxyFactoryBean` 接口来使用 AOP 功能，生成 proxy 代理对象，通过 `TransactionInterceptor` 完成对代理方法的拦截，将事务处理的功能编织到拦截的方法中。读取 IOC 容器事务配置属性，转化为 Spring 事务处理需要的内部数据结构（`TransactionAttributeSourceAdvisor`），转化为 `TransactionAttribute` 表示的数据对象。

对事物处理实现（事务的生成、提交、回滚、挂起）

Spring 委托给具体的事务处理器实现。实现了一个抽象和适配。适配的具体事务处理器：DataSource 数据源支持、Hibernate 数据源事务处理支持、JDO 数据源事务处理支持，JPA、JTA 数据源事务处理支持。这些支持都是通过设计 `PlatformTransactionManager`、`AbstractPlatforTransaction` 一系列事务处理的支持。 为常用数据源支持提供了一系列的 `TransactionManager`。

结合

`PlatformTransactionManager` 实现了 `TransactionInterception` 接口，让其与 `TransactionProxyFactoryBean` 结合起来，形成一个 Spring 声明式事务处理的设计体系。

十.Spring MVC 启动流程

在 `web.xml` 文件中给 Spring MVC 的 Servlet 配置了 `load-on-startup`，所以程序启动的时候会初始化 Spring MVC，在 `HttpServletBean` 中将配置的 == `contextConfigLocation` ==属性设置到 Servlet 中，然后在 `FrameworkServlet` 中创建了 `WebApplicationContext`，`DispatcherServlet` 根据 `contextConfigLocation` 配置的 `classpath` 下的 xml 文件初始化了 Spring MVC 总的组件。

十一.Spring MVC 运行流程

- Spring MVC 将所有的请求都提交给 `DispatcherServlet`，它会委托应用系统的其他模块负责对请求进行真正的处理工作。
- `DispatcherServlet` 查询一个或多个 `HandlerMapping`，找到处理请求的 Controller.
- `DispatcherServlet` 请求提交到目标 Controller
- Controller 进行业务逻辑处理后，会返回一个 `ModelAndView`
- Dispatcher 查询一个或多个 `ViewResolver` 视图解析器,找到 `ModelAndView` 对象指定的视图对象
- 视图对象负责渲染返回给客户端。

十二.你怎么理解 RPC 框架

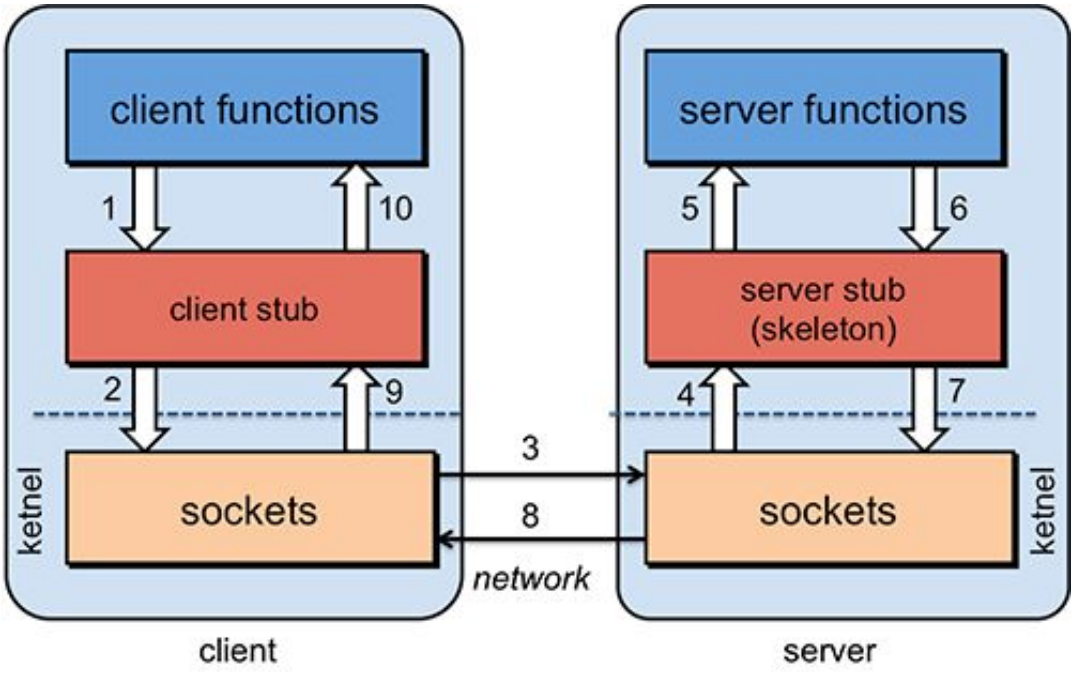
什么是 RPC?

RPC 是指远程过程调用，也就是说两台服务器 A，B 一个应用部署在 A 服务器上，想要调用 B 服务器上应用提供的函数或方法，由于不在一个内存空间，不能直接调用，需要通过网络来表达调用的语义和传达调用的数据。

RPC 的实现原理

首先需要有处理网络连接通讯的模块，负责连接建立、管理和消息的传输。其次需要有编解码的模块，因为网络通讯都是传输的字节码，需要将我们使用的对象序列化和反序列化。剩下的就是客户端和服务器端的部分，服务器端暴露要开放的服务接口，客户调用服务接口的一个代理实现，这个代理实现负责收集数据、编码并传输给服务器然后等待结果返回。

RPC 是如何通讯的?



1. 要解决通讯的问题，主要是通过在客户端和服务器之间建立 TCP 连接，远程过程调用的所有交换的数据都在这个连接里传输。连接可以是按需连接，调用结束后就断掉，也可以是长连接，多个远程过程调用共享同一个连接。
2. 要解决寻址的问题，也就是说，A 服务器上的应用怎么告诉底层的 RPC 框架，如何连接到 B 服务器（如主机或 IP 地址）以及特定的端口，方法的名称是什么，这样才能完成调用。比如基于 Web 服务协议栈的 RPC，就要提供一个 endpoint URI，或者是从 UDDI 服务上查找。如果是 RMI 调用的话，还需要一个 RMI Registry 来注册服务的地址。
3. 当 A 服务器上的应用发起远程过程调用时，方法的参数需要通过底层的网络协议如 TCP 传递到 B 服务器，由于网络协议是基于二进制的，内存中的参数的值要序列化成二进制的形式，也就是序列化（Serialize）或编组（marshal），通过寻址和传输将序列化的二进制发送给 B 服务器。
4. B 服务器收到请求后，需要对参数进行反序列化（序列化的逆操作），恢复为内存中的表达方式，然后找到对应的方法（寻址的一部分）进行本地调用，然后得到返回值。
5. 返回值还要发送回服务器 A 上的应用，也要经过序列化的方式发送，服务器 A 接到后，再反序列化，恢复为内存中的表达方式，交给 A 服务器上的应用。

为什么要用 RPC?

就是无法在一个进程内，甚至一个计算机内通过本地调用的方式完成的需求，比如比如不同的系统间的通讯，甚至不同的组织间的通讯。由于计算能力需要横向扩展，需要在多台机器组成的集群上部署应用，