BeanFactory 和 ApplicationContext 有什么区别

> BeanFactory 可以理解为含有 bean 集合的工厂类。BeanFactory 包含了种 bean 的定义，以便在接收到客户端请求时将对应的 bean 实例化。
> BeanFactory 还能在实例化对象的时生成协作类之间的关系。此举将 bean 自身与 bean 客户端的配置中解放出来。BeanFactory 还包含了 bean 生命周期的控制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。
> 从表面上看，application context 如同 bean factory 一样具有 bean 定义、bean 关联关系的设置，根据请求分发 bean 的功能。但 application context 在此基础上还提供了其他的功能。
> 提供了支持国际化的文本消息
> 统一的资源文件读取方式
> 已在监听器中注册的 bean 的事件

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

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。ApplicationContex 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。

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。

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。

动态代理（cglib 与 JDK）

> JDK 动态代理类和委托类需要都实现同一个接口。也就是说只有实现了某个接口的类可以使用 Java 动态代理机制。但是，事实上使用中并不是遇到的所有类都会给你实现一个接口。因此，对于没有实现接口的类，就不能使用该机制。而 CGLIB 则可以实现对类的动态代理。

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

Spring 事务底层原理

> a、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）。
> b、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。
> c、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。
> d、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。

如何自定义注解实现功能

> 创建自定义注解和创建一个接口相似，但是注解的 interface 关键字需要以@符号开头。
> 注解方法不能带有参数；
> 注解方法返回值类型限定为：基本类型、String、Enums、Annotation 或者是这些类型的数组；
> 注解方法可以有默认值；
> 注解本身能够包含元注解，元注解被用来注解其它注解。

Spring MVC 运行流程

> 1.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。
> 2.DispatcherServlet 查询一个或多个 HandlerMapping,找到处理请求的 Controller.
> 3.DispatcherServlet 请请求提交到目标 Controller
> 4.Controller 进行业务逻辑处理后，会返回一个 ModelAndView
> 5.Dispathcher 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
> 6.视图对象负责渲染返回给客户端。

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。

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。

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。

Netty

为什么选择 Netty

> 1）API 使用简单，开发门槛低；
> 2）功能强大，预置了多种编解码功能，支持多种主流协议；
> 3）定制能力强，可以通过 ChannelHandler 对通信框架进行灵活的扩展；
> 4）性能高，通过与其它业界主流的 NIO 框架对比，Netty 的综合性能最优；
> 5）成熟、稳定，Netty 修复了已经发现的所有 JDK NIO BUG，业务开发人员不需要再为 NIO 的 BUG 而烦恼；
> 6）社区活跃，版本迭代周期短，发现的 BUG 可以被及时修复，同时，更多的新功能会被加入；
> 7）经历了大规模的商业应用考验，质量已经得到验证。在互联网、大数据、网络游戏、企业应用、电信软件等众多行业得到成功商用，证明了它可以完全满足不同行业的商业应用。
> 正是因为这些优点，Netty 逐渐成为 Java NIO 编程的首选框架。

说说业务中，Netty 的使用场景

> 构建高性能、低时延的各种 Java 中间件，例如 MQ、分布式服务框架、ESB 消息总线等，Netty 主要作为基础通信框架提供高性能、低时延的通信服务；
> 公有或者私有协议栈的基础通信框架，例如可以基于 Netty 构建异步、高性能的 WebSocket 协议栈；
> 各领域应用，例如大数据、游戏等，Netty 作为高性能的通信框架用于内部各模块的数据分发、传输和汇总等，实现模块之间高性能通信。

原生的 NIO 在 JDK 1.7 版本存在 epoll bug

> 它会导致 Selector 空轮询，最终导致 CPU 100%。官方声称在 JDK 1.6 版本的 update18 修复了该问题，但是直到 JDK 1.7 版本该问题仍旧存在，只不过该 BUG 发生概率降低了一些而已，它并没有得到根本性解决。

什么是 TCP 粘包/拆包

> 1、要发送的数据大于 TCP 发送缓冲区剩余空间大小，将会发生拆包。
> 2、待发送数据大于 MSS（最大报文长度），TCP 在传输前将进行拆包。
> 3、要发送的数据小于 TCP 发送缓冲区的大小，TCP 将多次写入缓冲区的数据一次发送出去，将会发生粘包。
> 4、接收数据端的应用层没有及时读取接收缓冲区中的数据，将发生粘包。

TCP 粘包/拆包的解决办法

> 1、发送端给每个数据包添加包首部，首部中应该至少包含数据包的长度，这样接收端在接收到数据后，通过读取包首部的长度字段，便知道每一个数据包的实际长度了。
> 2、发送端将每个数据包封装为固定长度（不够的可以通过补 0 填充），这样接收端每次从接收缓冲区中读取固定长度的数据就自然而然的把每个数据包拆分开来。
> 3、可以在数据包之间设置边界，如添加特殊符号，这样，接收端通过这个边界就可以将不同的数据包拆分开。

Netty 线程模型

> 首先，Netty 使用 EventLoop 来处理连接上的读写事件，而一个连接上的所有请求都保证在一个 EventLoop 中被处理，一个 EventLoop 中只有一个 Thread，所以也就实现了一个连接上的所有事件只会在一个线程中被执行。一个 EventLoopGroup 包含多个 EventLoop，可以把一个 EventLoop 当做是 Reactor 线程模型中的一个线程，而一个 EventLoopGroup 类似于一个 ExecutorService

说说 Netty 的零拷贝

> “零拷贝”是指计算机操作的过程中，CPU 不需要为数据在内存之间的拷贝消耗资源。而它通常是指计算机在网络上发送文件时，不需要将文件内容拷贝到用户空间（User Space）而直接在内核空间（Kernel Space）中传输到网络的方式。

Netty 内部执行流程

> 1. Netty 的接收和发送 ByteBuffer 采用 DIRECT BUFFERS，使用堆外直接内存进行 Socket 读写，不需要进行字节缓冲区的二次拷贝。如果使用传统的堆内存（HEAP BUFFERS）进行 Socket 读写，JVM 会将堆内存 Buffer 拷贝一份到直接内存中，然后才写入 Socket 中。相比于堆外直接内存，消息在发送过程中多了一次缓冲区的内存拷贝。

> 2. Netty 提供了组合 Buffer 对象，可以聚合多个 ByteBuffer 对象，用户可以像操作一个 Buffer 那样方便的对组合 Buffer 进行操作，避免了传统通过内存拷贝的方式将几个小 Buffer 合并成一个大的 Buffer。
> 3. Netty 的文件传输采用了 transferTo 方法，它可以直接将文件缓冲区的数据发送到