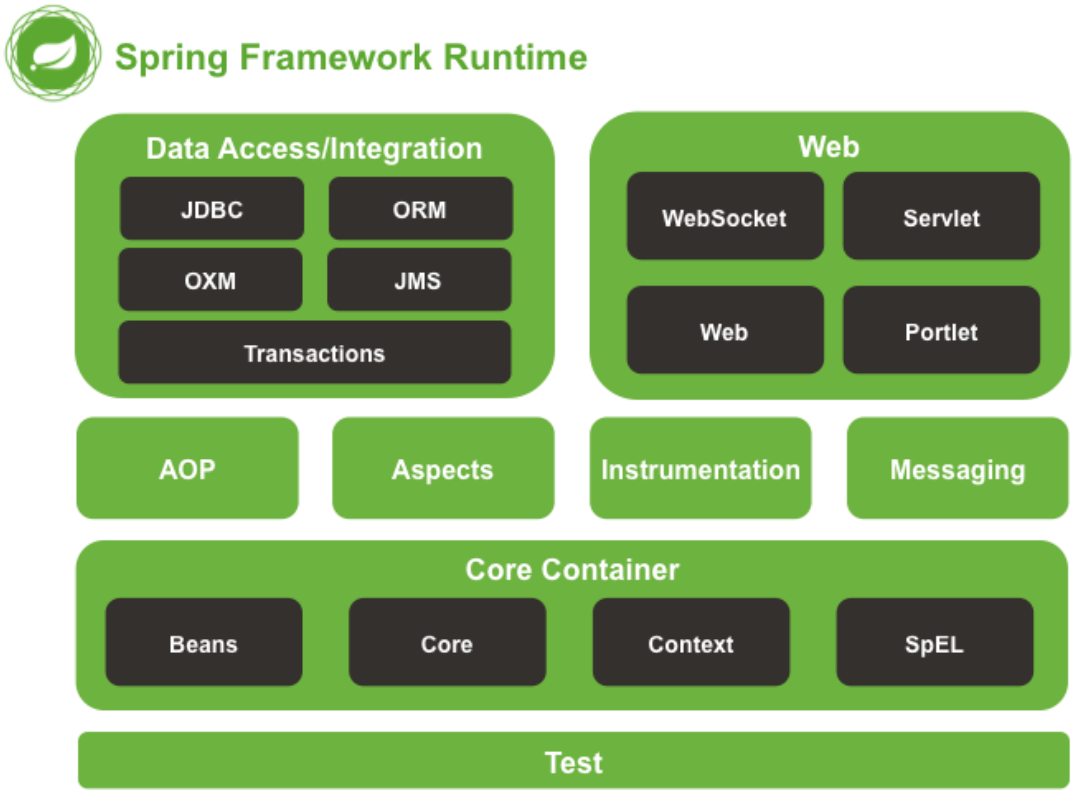
Spring面试题总结

1、什么是Spring框架?

Spring是一种开源的轻量级java框架，旨在提高开发人员的开发效率以及系统的可维护性。

2、列举一些重要的Spring模块?



- Spring Core：基础，可以说Spring其他所有的功能都依赖于该类库。主要提供IOC和DI功能。
- Spring Aspects：该模块为与AspectJ的集成提供支持。
- Spring AOP：提供面向方面的编程实现。
- Spring JDBC：Java数据库连接。
- Spring JMS：Java消息服务。
- Spring ORM：用于支持Hibernate等ORM工具。
- Spring Web：为创建Web应用程序提供支持。
- Spring Test：提供了对JUnit和TestNG测试的支持。

3、什么是IOC? 如何实现的?

IOC（Inversion Of Controll，控制反转）是一种设计思想，就是将原本在程序中手动创建对象的控制权，交给IOC容器来管理，并由IOC容器完成对象的注入。这样可以很大程度上简化应用的开发，把应用从复杂的依赖关系中解放出来。IOC容器就像是一个工厂一样，当我们需要创建一个对象的时候，只需要配置好配置文件/注解即可，完全不用考虑对象是如何被创建出来的。

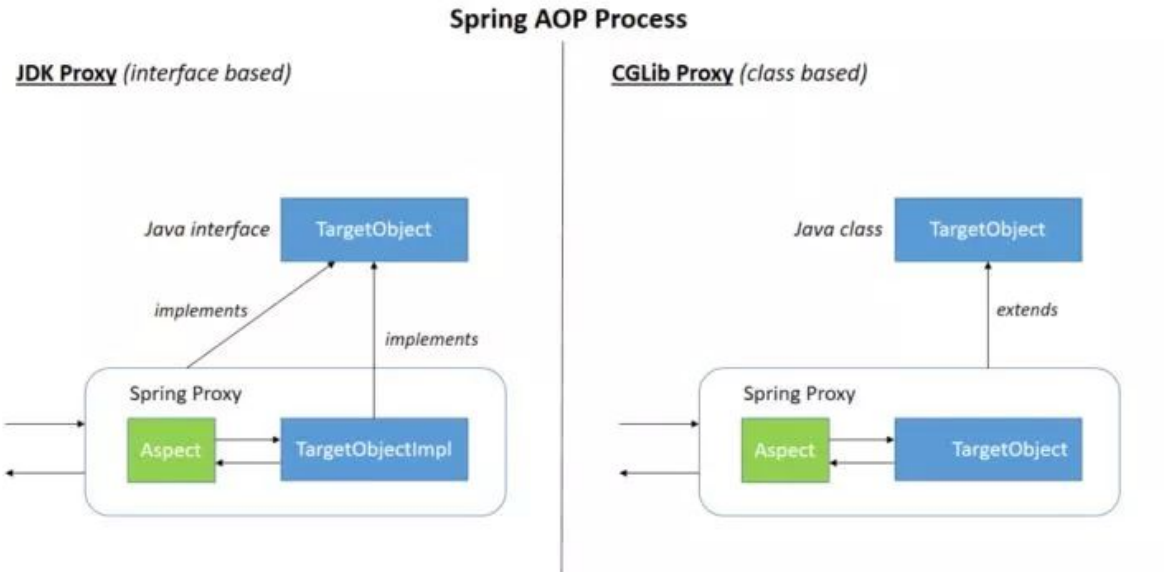
Spring 中的 IoC 的实现原理就是工厂模式加反射机制。

```
interface Fruit {
     public abstract void eat();
}
class Apple implements Fruit {
    public void eat(){
        System.out.println("Apple");
    }
}

class Orange implements Fruit {
    public void eat(){
        System.out.println("Orange");
    }
}

class Factory {
    public static Fruit getInstance(String ClassName) {
        Fruit f=null;
        try {
            f=(Fruit)Class.forName(ClassName).newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return f;
    }
}

class Client {
    public static void main(String[] a) {
        Fruit f=Factory.getInstance("io.github.dunwu.spring.Apple");
        if(f!=null){
            f.eat();
        }
    }
}
```

4、什么是AOP? 有哪些AOP的概念?

AOP（Aspect-Oriented Programming，面向切面编程）能够将那些与业务无关，却为业务模块所共同调用的逻辑或责任（例如事务处理、日志管理、权限控制等）封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可扩展性和可维护性。

Spring AOP是基于动态代理的，如果要代理的对象实现了某个接口，那么Spring AOP就会使用JDK动态代理去创建代理对象；而对于没有实现接口的对象，就无法使用JDK动态代理，转而使用CGlib动态代理生成一个被代理对象的子类来作为代理。



当然也可以使用AspectJ，Spring AOP中已经集成了AspectJ，AspectJ应该算得上是Java生态系统中最完整的AOP框架了。使用AOP之后我们可以把一些通用功能抽象出来，在需要用到的地方直接使用即可，这样可以大大简化代码量。我们需要增加新功能也方便，提高了系统的扩展性。日志功能、事务管理和权限管理等场景都用到了AOP。

5、解释AOP中的几个概念

1. Jointpoint（连接点）：具体的切面点点抽象概念，可以是在字段、方法上，Spring中具体表现形式是PointCut（切入点），仅作用在方法上。
2. Advice（通知）: 在连接点进行的具体操作，如何进行增强处理的，分为前置、后置、异常、最终、环绕五种情况。
3. 目标对象：被AOP框架进行增强处理的对象，也被称为被增强的对象。
4. AOP代理：AOP框架创建的对象，简单的说，代理就是对目标对象的加强。Spring中的AOP代理可以是JDK动态代理，也可以是CGLIB代理。
5. Weaving（织入）：将增强处理添加到目标对象中，创建一个被增强的对象的过程

总结为一句话就是：在目标对象（target object）的某些方法（jointpoint）添加不同种类的操作（通知、增强操处理），最后通过某些方法（weaving、织入操作）实现一个新的代理目标对象。

6、AOP 有哪些应用场景?

举几个例子：

- 记录日志(调用方法后记录日志)
- 监控性能(统计方法运行时间)
- 权限控制(调用方法前校验是否有权限)
- 事务管理(调用方法前开启事务，调用方法后提交关闭事务)
- 缓存优化(第一次调用查询数据库，将查询结果放入内存对象， 第二次调用，直接从内存对象返回，不需要查询数据库)

7、有哪些AOP Advice通知的类型?

特定 JoinPoint 处的 Aspect 所采取的动作称为 Advice。Spring AOP 使用一个 Advice 作为拦截器，在 JoinPoint “周围”维护一系列的拦截器。

- **前置通知**（Before advice）： 这些类型的 Advice 在 joinpoint 方法之前执行，并使用 @Before 注解标记进行配置。

- **后置通知**（After advice）：这些类型的 Advice 在连接点方法之后执行，无论方法退出是正常还是异常返回，并使用 @After 注解标记进行配置。
- **返回后通知**（After return advice）：这些类型的 Advice 在连接点方法正常执行后执行，并使用 @AfterReturning 注解标记进行配置。
- **环绕通知**（Around advice）：这些类型的 Advice 在连接点之前和之后执行，并使用 @Around 注解标记进行配置。
- **抛出异常后通知**（After throwing advice）：仅在 joinpoint 方法通过抛出异常退出并使用 @AfterThrowing 注解标记配置时执行。

8、AOP 有哪些实现方式?

实现 AOP 的技术，主要分为两大类：

- 静态代理

  - 指使用 AOP 框架提供的命令进行编译，从而在编译阶段就可生成 AOP 代理类，因此也称为编译时增强；

    - 编译时编织（特殊编译器实现）
    - 类加载时编织（特殊的类加载器实现）。
- 动态代理

  - 在运行时在内存中“临时”生成 AOP 动态代理类，因此也被称为运行时增强。

    - JDK 动态代理
      - JDK Proxy 是 Java 语言自带的功能，无需通过加载第三方类实现；
      - Java 对 JDK Proxy 提供了稳定的支持，并且会持续的升级和更新，Java 8 版本中的 JDK Proxy 性能相比于之前版本提升了很多；
      - JDK Proxy 是通过拦截器加反射的方式实现的；
      - JDK Proxy 只能代理实现接口的类；
      - JDK Proxy 实现和调用起来比较简单；
    - CGLIB
      - CGLib 是第三方提供的工具，基于 ASM 实现的，性能比较高；
      - CGLib 无需通过接口来实现，它是针对类实现代理，主要是对指定的类生成一个子类，它是通过实现子类的方式来完成调用的。

9、谈谈你对CGLib的理解?

JDK 动态代理机制只能代理实现接口的类，一般没有实现接口的类不能进行代理。使用 CGLib 实现动态代理，完全不受代理类必须实现接口的限制。

CGLib 的原理是对指定目标类生成一个子类，并覆盖其中方法实现增强，但因为采用的是继承，所以不能对 final 修饰的类进行代理。

```
public class CGLibDemo {

    // 需要动态代理的实际对象
    static class Sister  {
        public void sing() {
            System.out.println("I am Jinsha, a little sister.");
        }
    }

    static class CGLibProxy implements MethodInterceptor {

        private Object target;
```

```
        public Object getInstance(Object target){
            this.target = target;
            Enhancer enhancer = new Enhancer();
            // 设置父类为实例类
            enhancer.setSuperclass(this.target.getClass());
            // 回调方法
            enhancer.setCallback(this);
            // 创建代理对象
            return enhancer.create();
        }

        @Override
        public Object intercept(Object o, Method method, Object[] objects,
MethodProxy methodProxy) throws Throwable {
            System.out.println("introduce yourself...");
            Object result = methodProxy.invokeSuper(o,objects);
            System.out.println("score...");
            return result;
        }
    }

    public static void main(String[] args) {
        CGLibProxy cgLibProxy = new CGLibProxy();
        //获取动态代理类实例
        Sister proxySister = (Sister) cgLibProxy.getInstance(new Sister());
        System.out.println("CGLib Dynamic object name: " +
proxySister.getClass().getName());
        proxySister.sing();
    }
}
```

CGLib 的调用流程就是通过调用拦截器的 intercept 方法来实现对被代理类的调用。而拦截逻辑可以写在 intercept 方法的 invokeSuper(o, objects);的前后实现拦截。

10、Spring AOP和AspectJ AOP有什么区别?

Spring AOP是属于运行时增强，而AspectJ是编译时增强。Spring AOP基于代理（Proxying），而AspectJ基于字节码操作（Bytecode Manipulation）。

Spring AOP已经集成了AspectJ，AspectJ应该算得上是Java生态系统中最完整的AOP框架了。AspectJ相比于Spring AOP功能更加强大，但是Spring AOP相对来说更简单。

如果我们的切面比较少，那么两者性能差异不大。但是，当切面太多的话，最好选择AspectJ，它比SpringAOP快很多。

11、Spring中的bean的作用域有哪些?

1. singleton：唯一bean实例，Spring中的bean默认都是单例的。
2. prototype：每次请求都会创建一个新的bean实例。
3. request：每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP request内有效。
4. session：每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP session内有效。
5. global-session：全局session作用域，仅仅在基于Portlet的Web应用中才有意义，Spring5中已经没有了。Portlet是能够生成语义代码（例如HTML）片段的小型Java Web插件。它们基于Portlet容器，可以像Servlet一样处理HTTP请求。但是与Servlet不同，每个Portlet都有不同的会话。

12、Spring中的单例bean的线程安全问题了解吗?

大部分时候我们并没有在系统中使用多线程，所以很少有人会关注这个问题。单例bean存在线程问题，主要是因为当多个线程操作同一个对象的时候，对这个对象的非静态成员变量的写操作会存在线程安全问题。

有两种常见的解决方案：

1.在bean对象中尽量避免定义可变的成员变量（不太现实）。

2.在类中定义一个ThreadLocal成员变量，将需要的可变成员变量保存在ThreadLocal中（推荐的一种方式）。

13、Spring中的bean生命周期?

1.Bean容器找到配置文件中Spring Bean的定义。

2.Bean容器利用Java Reflection API创建一个Bean的实例。

3.如果涉及到一些属性值，利用set()方法设置一些属性值。

4.如果Bean实现了BeanNameAware接口，调用setBeanName()方法，传入Bean的名字。

5.如果Bean实现了BeanClassLoaderAware接口，调用setBeanClassLoader()方法，传入ClassLoader对象的实例。

6.如果Bean实现了BeanFactoryAware接口，调用setBeanClassFacotory()方法，传入ClassLoader对象的实例。

7.与上面的类似，如果实现了其他*Aware接口，就调用相应的方法。

8.如果有和加载这个Bean的Spring容器相关的BeanPostProcessor对象，执行postProcessBeforeInitialization()方法。

9.如果Bean实现了InitializingBean接口，执行afeterPropertiesSet()方法。

10.如果Bean在配置文件中的定义包含init-method属性，执行指定的方法。

11.如果有和加载这个Bean的Spring容器相关的BeanPostProcess对象，执行postProcessAfterInitialization()方法。

12.当要销毁Bean的时候，如果Bean实现了DisposableBean接口，执行destroy()方法。

13.当要销毁Bean的时候，如果Bean在配置文件中的定义包含destroy-method属性，执行指定的方法。

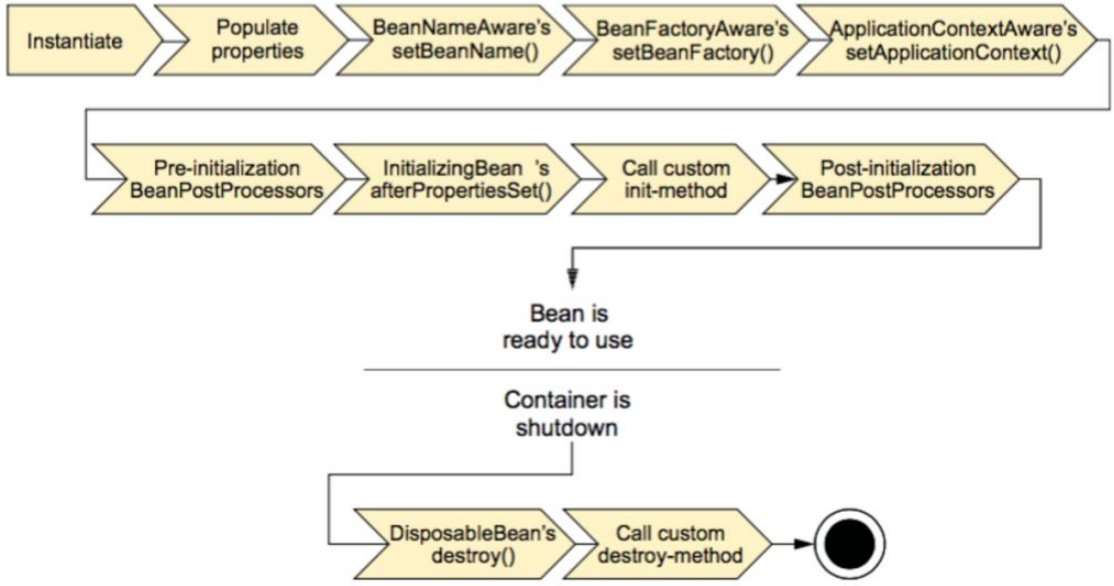
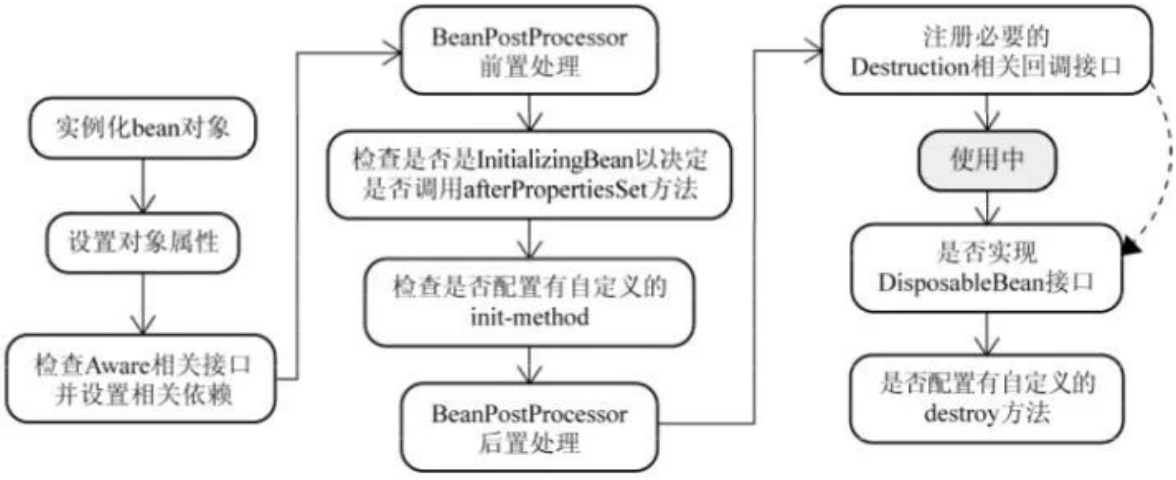


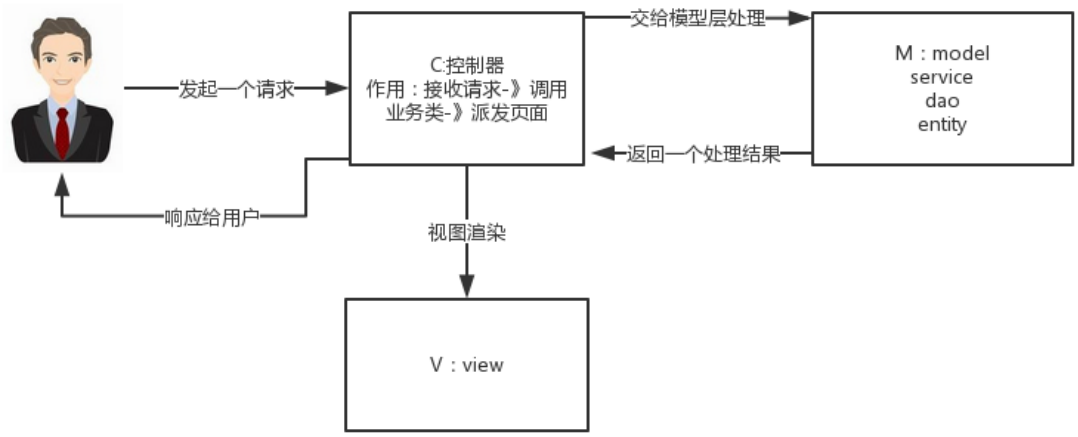
Figure 1.5 A bean goes through several steps between creation and destruction in the Spring container. Each step is an opportunity to customize how the bean is managed in Spring.



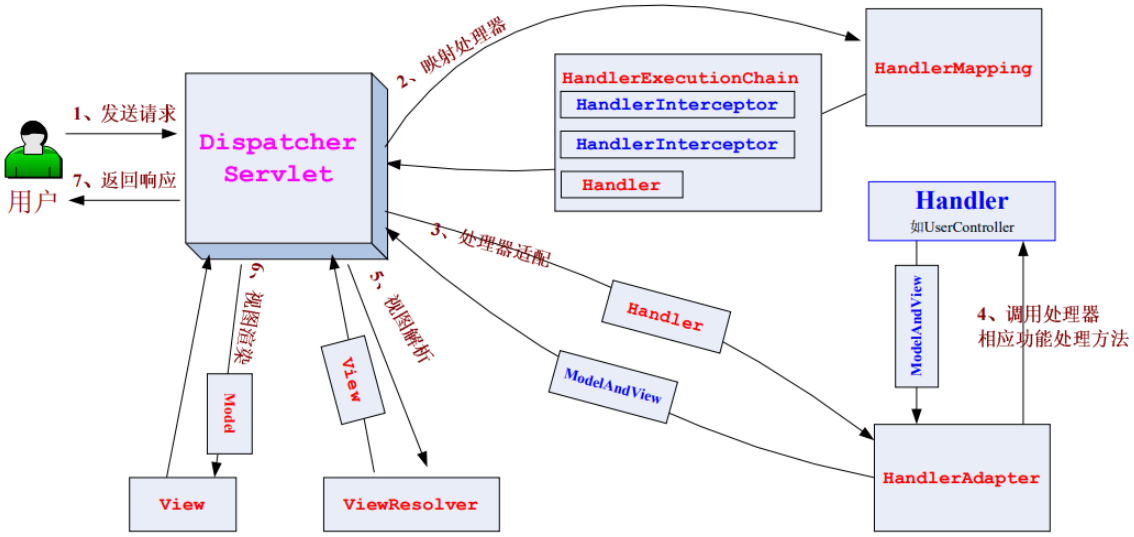
14、说说自己对于Spring MVC的了解?

MVC是一种设计模式，Spring MVC是一款很优秀的MVC框架。Spring MVC可以帮助我们进行更简洁的Web层的开发，并且它天生与Spring框架集成。Spring MVC下我们一般把后端项目分为Service层（处理业务）、Dao层（数据库操作）、Entity层（实体类）、Controller层（控制层，返回数据给前台页面）。

Spring MVC的简单原理图如下：



15、Spring MVC的工作原理了解嘛?



流程说明:

1.客户端（浏览器）发送请求，直接请求到DispatcherServlet。

2.DispatcherServlet根据请求信息调用HandlerMapping，解析请求对应的Handler。

3.解析到对应的Handler（也就是我们平常说的Controller控制器）。

4.HandlerAdapter会根据Handler来调用真正的处理器来处理请求和执行相对应的业务逻辑。

5.处理器处理完业务后，会返回一个ModelAndView对象，Model是返回的数据对象，View是逻辑上的View。

6.ViewResolver会根据逻辑View去查找实际的View。

7.DispatcherServlet把返回的Model传给View（视图渲染）。

8.把View返回给请求者（浏览器）。

16、Spring框架中用到了哪些设计模式?

举几个例子

1.工厂设计模式：Spring使用工厂模式通过BeanFactory和ApplicationContext创建bean对象。

2.代理设计模式：Spring AOP功能的实现。

3.单例设计模式：Spring中的bean默认都是单例的。

4.模板方法模式：Spring中的jdbcTemplate、hibernateTemplate等以Template结尾的对数据库操作的类，它们就使用到了模板模式。

5.包装器设计模式：我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。

6.观察者模式：Spring事件驱动模型就是观察者模式很经典的一个应用。

7.适配器模式：Spring AOP的增强或通知（Advice）使用到了适配器模式、Spring MVC中也是用到了适配器模式适配Controller。

17、@Component和@Bean的区别是什么?

1.作用对象不同。@Component注解作用于类，而@Bean注解作用于方法。

2.@Component注解通常是通过类路径扫描来自动侦测以及自动装配到Spring容器中（我们可以使用@ComponentScan注解定义要扫描的路径）。@Bean注解通常是在标有该注解的方法中定义产生这个bean，告诉Spring这是某个类的实例，当我需要用它的时候还给我。

3.@Bean注解比@Component注解的自定义性更强，而且很多地方只能通过@Bean注解来注册bean。比如当引用第三方库的类需要装配到Spring容器的时候，就只能通过@Bean注解来实现。

```
@Configuration
public class AppConfig {
    @Bean
    public TransferService transferService() {
        return new TransferServiceImpl();
    }
}
```

```
<beans>
    <bean id="transferService" class="com.yanggb.TransferServiceImpl"/>
</beans>
```

```
@Bean
public OneService getService(status) {
    case (status)  {
        when 1:
                return new serviceImpl1();
        when 2:
                return new serviceImpl2();
        when 3:
                return new serviceImpl3();
    }
}
```

18、将一个类声明为Spring的bean的注解有哪些?

我们一般使用@Autowired注解去自动装配bean。而想要把一个类标识为可以用@Autowired注解自动装配的bean，可以采用以下的注解实现：

1.@Component注解。通用的注解，可标注任意类为Spring组件。如果一个Bean不知道属于哪一个层，可以使用@Component注解标注。

2.@Repository注解。对应持久层，即Dao层，主要用于数据库相关操作。

3.@Service注解。对应服务层，即Service层，主要涉及一些复杂的逻辑，需要用到Dao层（注入）。

4.@Controller注解。对应Spring MVC的控制层，即Controller层，主要用于接受用户请求并调用Service层的方法返回数据给前端页面。

19、Spring事务管理的方式有几种?

1.编程式事务：在代码中硬编码（不推荐使用）。

2.声明式事务：在配置文件中配置（推荐使用），分为基于XML的声明式事务和基于注解的声明式事务。

20、Spring事务中的隔离级别有哪几种?

在TransactionDefinition接口中定义了五个表示隔离级别的常量：

ISOLATION_DEFAULT：使用后端数据库默认的隔离级别，Mysql默认采用的REPEATABLE_READ隔离级别；Oracle默认采用的READ_COMMITTED隔离级别。

ISOLATION_READ_UNCOMMITTED：最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读。

ISOLATION_READ_COMMITTED：允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生

ISOLATION_REPEATABLE_READ：对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。

ISOLATION_SERIALIZABLE：最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读。但是这将严重影响程序的性能。通常情况下也不会用到该级别。

21、Spring事务中有哪几种事务传播行为?

在TransactionDefinition接口中定义了八个表示事务传播行为的常量。

支持当前事务的情况：

PROPAGATION_REQUIRED：如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。

PROPAGATION_SUPPORTS： 如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。

PROPAGATION_MANDATORY： 如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。（mandatory：强制性）。

不支持当前事务的情况：

PROPAGATION_REQUIRES_NEW： 创建一个新的事务，如果当前存在事务，则把当前事务挂起。

PROPAGATION_NOT_SUPPORTED： 以非事务方式运行，如果当前存在事务，则把当前事务挂起。

PROPAGATION_NEVER：以非事务方式运行，如果当前存在事务，则抛出异常。

其他情况：

PROPAGATION_NESTED：如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于PROPAGATION_REQUIRED。

22、Bean Factory和ApplicationContext有什么区别?

ApplicationContex提供了一种解析文本消息的方法，一种加载文件资源（如图像）的通用方法，它们可以将事件发布到注册为侦听器的bean。此外，可以在应用程序上下文中以声明方式处理容器中的容器或容器上的操作，这些操作必须以编程方式与Bean Factory一起处理。ApplicationContex实现MessageSource，一个用于获取本地化消息的接口，实际的实现是可插入的。

23、如何定义bean的范围?

在Spring中定义一个时，我们也可以为bean声明一个范围。它可以通过bean定义中的scope属性定义。例如，当Spring每次需要生成一个新的bean实例时，bean'sscope属性就是原型。另一方面，当每次需要Spring都必须返回相同的bean实例时，bean scope属性必须设置为singleton。

24、可以通过多少种方式完成依赖注入?

通常，依赖注入可以通过三种方式完成，即：

- 构造函数注入
- setter 注入
- 接口注入

###