四十一、springboot中bean的生命周期？

Bean 容器找到配置文件中 Spring Bean 的定义。（@Component等）
Bean 容器利用 Java Reflection API 创建一个Bean的实例。（无参、有参构造方法（实例化））
如果涉及到一些属性值 利用 set() 方法设置一些属性值。（依赖注入）
如果 Bean 实现了 BeanNameAware 接口，调用 setBeanName() 方法，传入Bean的名字。 如果 Bean 实现了 BeanClassLoaderAware 接口，调用 setBeanClassLoader() 方法， 传入 ClassLoader 对象的实例。
与上面的类似，如果实现了其他 *.Aware 接口，就调用相应的方法。
如果有和加载这个 Bean 的 Spring 容器相关的 BeanPostProcessor 对象，执 行 postProcessBeforeInitialization() 方法（初始化前置方法）
如果Bean实现了 InitializingBean 接口，执行 afterPropertiesSet() 方法。（初始化-自定义逻辑）
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。（初始化-自定义逻辑）
如果有和加载这个 Bean的 Spring 容器相关的 BeanPostProcessor 对象，执 行 postProcessAfterInitialization() 方法（初始化后置方法）
当要销毁 Bean 的时候，如果 Bean 实现了 DisposableBean 接口，执行 destroy() 方法。
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。

四十二、spring中Aware接口和BeanPostProcessor和BeanFactoryPostProcessor的区别？

（1）从调用时间点上看，Aware接口的方法（都是setXXX方法）是在Bean的属性被设置之后，
初始化方法初始化方法（InitializingBean接口的方法，或@PostContruct等）执行之前被调用（这个时候Bean的整个初始化过程还没有完成）
BeanPostProcessor接口由两个回调方法组成，初始化方法（InitializingBean接口的方法，或@PostContruct等）执行之前调用BeanPostProcessor的postProcessBeforeInitialization回调，
并在初始化方法执行之后调用BeanPostProcessor的postProcessAfterInitialization回调；
BeanFactoryPostProcessor是在容器初始化之前被调用。
（2）从功能上来看，BeanPostProcessor主要是用来对实例化后的Bean做操作；
BeanFactoryPostProcessor可以取得BeanFactory的引用，对实例化前的Bean定义做操作。
Aware接口分的比较细，可以取得各种引用，例如：ApplicationContext，BeanFactory，MessageSource等等。
在ApplicationContextAware接口里，也可以可以通过ApplicationContext取得BeanFactory，但要注意Aware接口方法的被调用时间点和BeanFactoryPostProcessor不一样。（也可以使用BeanFactoryAware接口取得BeanFactory）
（3）从实现上看，Aware接口的回调其实是通过BeanPostProcessor接口实现的。
可以看一下ApplicationContextAwareProcessor这个类，这个类继承了BeanPostProcessor接口。
这个类的处理逻辑就是，看Bean是否是几个Aware接口的实例，如果是的话就调用接口提供的回调。

四十二、InitializingBean的作用？和init-method有什么区别？

InitializingBean 接口为 bean 提供了 初始化方法的方式，接口只包括一个无返回值的 afterPropertiesSet 方法，凡是继承该接口的类，在初始化 bean 的时候都会执行该方法

1：spring为bean提供了两种初始化bean的方式，实现InitializingBean接口，实现afterPropertiesSet方法，或者在配置文件中通过init-method指定，两种方式可以同时使用
2：实现InitializingBean接口是直接调用afterPropertiesSet方法，比通过反射调用init-method指定的方法效率相对来说要高点。但是init-method方式消除了对spring的依赖
3：调用afterPropertiesSet方法在前。如果调用afterPropertiesSet方法时出错，则不调用init-method指定的方法。

四十三、Dubbo集群提供的负载均衡策略？介绍下一致性hash策略？

Random LoadBalance: 随机选取提供者策略，有利于动态调整提供者权重。截面碰撞率高，调用次数越多，分布越均匀；（权重概率）
RoundRobin LoadBalance: 轮循选取提供者策略，平均分布，但是存在请求累积的问题；（轮询）
LeastActive LoadBalance: 最少活跃调用策略，解决慢提供者接收更少的请求；（找最快的）
ConstantHash LoadBalance: 一致性Hash策略，使相同参数请求总是发到同一提供者，一台机器宕机，可以基于虚拟节点，分摊至其他提供者，避免引起提供者的剧烈变动；

普通的hash算法： hash值 % 服务器数 -> 服务器数量变化 -> 缓存大面积失效 -> 缓存雪崩
一致性hash算法： 根据后端节点的某个固定属性计算 hash 值，然后把所有节点计算出来的 hash 值放在一个2的32次方节点数的 hash 圆环。
请求过来的时候根据请求的特征计算 hash 值，然后顺时针查找 hash 环上的 hash 值，第一个比请求特征的 hash 值大的 hash 值所对应的节点即为被选中的节点。
当服务器数量发生变化，可以使得只有少部分缓存失效
缺点：hash偏斜 -> 部分节点上承受太多的请求 -> 引入虚拟节点 -> 每个实际节点重复 n 次，放入hash圆环上

四十四、缓存雪崩、穿透、击穿

雪崩：同一时间缓存大面积失效。
1、redis存数据的时候加随机过期时间expire
2、设置热点数据永不过期，有更新操作更新缓存（电商首页）
穿透：用户不断发起请求，缓存和数据库都没有的数据。（id为负数或者很大值的用户）
1、在接口层增加参数校验，不合法参数直接返回。
2、将不存在的key对应的value写为null、稍后重试等，并将过期时间设置短点
3、网关层nginx有配置项，可以对单ip每秒访问次数超过阈值的ip拉黑
4、布隆过滤器：利用搞笑的数据结构和算法快速判断是不是在数据库中存在。（布谷鸟过滤器）
击穿：某个key非常热点，扛着大并发，突然这个key失效，直接请求数据库
1、设置热点数据永不过期

四十五、解释分库分表？

垂直分库：（免费表、会员可用表）
按表的业务归属不同，不同表拆入不同的库，专库专表
分析：公用的配置表拆到单独的库中，可以服务化

垂直分表：（详情表、展示表）
以字段活跃性为依据，将表中字段拆到不同的表（主表和扩展表）中。
问题：查询时读磁盘产生大量随机读io，产生io瓶颈。
修改热点数据会影响读热点数据。
分析：每个表字段至少有一列交集，一般是主键，用于关联数据。且查询关联数据应在Service层做文章
，因为join增加cpu负担且必须在一个数据库实例上。

```
水平分库：以字段为依据，将一个库中的表拆到多个数据库中。
    分析：系统绝对并发量上来了，分表不能解决根本问题
        库多了，io和cpu的压力就降下来了

水平分表：以字段为依据，将一个表拆为多个表
    分析：系统的绝对并发量没有上来，只是因为单表的数据量太多，影响sql效率，加重cpu负担
        表数据量少了，单词sql知性效率高，自然减轻cpu压力

1数据库在设计时就要考虑垂直分库，垂直分表
2优先考虑缓存处理，读写分离，使用索引等方式，不行再做水平分库和水平分表。
3工具：sharding-sphere、Mycat
```

四十六、分库分表带来什么问题？

1.事务一致性问题

当更新内容同时分布在不同库中，就会有跨库事务问题。一般采用“XA协议”和“两阶段提交处理”。

分布式事务在提交事务时需要协调多个节点，推后了提交事务的时间点，延长事务 执行时间。导致事务在访问共享资源时发生冲突或死锁的概率提高。所以对于 性能要求很高，但对一致性要求不高的系统，可以在允许的时间达到最终一致 性即可。

2.跨节点关联查询join问题。

数据可能分布在不同节点上，考虑性能，尽量避免使用join查询。

1.建立全局表，即字典表，把系统中所有模块都可能依赖的表，这些数据通常会很少修改，在每个数据库中都保存一份。

2.字段冗余，空间换时间避免join。适用场景有限，会出现冗余字段数据一致性问题。

3.数据组装，在服务层多次查询，将数据拼装

4.ER分片，如果表之间关联关系明显，可以放到同一个分片上。

3.跨节点分页，排序，函数问题。

Limit分页，order by排序，Max、Sum函数。可能都需要在不同分片排序返回，再 将所有结果汇总再次排序。

4.主键重复问题。

UUID、SnowFlake分布式自增ID算法

四十七、beanfactory和applicationcontext的区别？

applicationcontext是BeanFactory的子接口，功能更完整，一般情况都用它。

BeanFactory：

1、延时加载，节约内存，但是程序启动时不一定容易发现配置问题

2、以编程方式被创建

3、支持BeanPostProcessor，BeanFactoryPostProcessor，需要手动注册

ApplicationContext：

1、容器启动时全部加载，占用内存，容易启动时发现配置错误，运行快

2、以编程、生命方式被创建

3、支持BeanPostProcessor，BeanFactoryPostProcessor，需要手动注册

四十八、解释MVCC？

多版本并发控制：读取时通过一种快照的方式将数据保存，使得读锁和写锁不冲突。只工作在读已提交和可重复读两个隔离级别下。

在InnoDB引擎表中，它的聚簇索引记录中有两个必要的隐藏列：

trx_id：

这个id用来存储的每次对某条聚簇索引记录进行修改的时候的事务id。

roll_pointer：

每次对哪条聚簇索引记录有修改的时候，都会把老版本写入undo日志中。这个roll_pointer就是存了一个指针，它指向这条聚簇索引记录的上一个版本的位置，通过它来获得上一个版本的记录信息。(注意插入操作的undo日志没有这个属性，因为它没有老版本)

```
开始事务创建ReadView，ReadView维护当前活动事务的id，即未提交事务的id，为一个排序数组。
访问数据时，获取数据中的最大事务id和ReadView对比，在左边意味该事务已提交，可以访问。
在右边或者在ReadView中，意味还未提交。不可访问，通过roll_pointer获取上一版本号重新对比。

读已提交隔离级别下的事务在每次查询的开始都会生成一个独立的ReadView,而可重复读隔离级别则在第一次读的时候生成一个ReadView，之后的读都复用之前的ReadView。
```

四十九、为什么Java中不支持多重继承？

1.第一个原因是围绕钻石💎形继承问题产生的歧义。

2.多重继承确实使设计复杂化并在转换、构造函数链接等过程中产生问题。假设你需要多重继承的情况并不多，简单起见，明智的决定是省略它。

此外，Java 可以通过使用接口支持单继承来避免这种歧义。由于接口只有方法声明而且没有提供任何实现，因此只有一个特定方法的实现，因此不会有任何歧义。

五十、MyBatis 的工作原理？

1）读取 MyBatis 配置文件：mybatis-config.xml 为 MyBatis 的全局配置文件，配置了 MyBatis 的运行环境等信息，例如数据库连接信息。

```
2）加载映射文件。映射文件即 SQL 映射文件，该文件中配置了操作数据库的 SQL 语句，需要在 MyBatis 配置文件 mybatis-config.xml 中加载。mybatis-config.xml 文件可以加载多个映射文件，每个文件对应数据库中的一张表。

3）构造会话工厂：通过 MyBatis 的环境等配置信息构建会话工厂 SqlSessionFactory。

4）创建会话对象：由会话工厂创建 SqlSession 对象，该对象中包含了执行 SQL 语句的所有方法。

5）Executor 执行器：MyBatis 底层定义了一个 Executor 接口来操作数据库，它将根据 SqlSession 传递的参数动态地生成需要执行的 SQL 语句，同时负责查询缓存的维护。

6）MappedStatement 对象：在 Executor 接口的执行方法中有一个 MappedStatement 类型的参数，该参数是对映射信息的封装，用于存储要映射的 SQL 语句的 id、参数等信息。

7）输入参数映射：输入参数类型可以是 Map、List 等集合类型，也可以是基本数据类型和 POJO 类型。输入参数映射过程类似于 JDBC 对 preparedStatement 对象设置参数的过程。

8）输出结果映射：输出结果类型可以是 Map、 List 等集合类型，也可以是基本数据类型和 POJO 类型。输出结果映射过程类似于 JDBC 对结果集的解析过程。
```

五十一、为什么建议innodb表必须建主键，并且推荐使用整形的自增主键？

如果设置了主键，那么InnoDB会选择主键作为聚集索引、如果没有显式定义主键，则InnoDB会选择第一个不包含有NULL值的唯一索引作为主键索引、

如果也没有这样的唯一索引，则InnoDB会选择内置6字节长的ROWID作为隐含的聚集索引(ROWID随着行记录的写入而主键递增)。

我们为了节约数据库资源，为了性能高，节约开销，尽量去要建立主键。

> 如果表使用自增主键，那么每次插入新的记录，记录就会顺序添加到当前索引节点的后续位置，主键的顺序按照数据记录的插入顺序排列，自动有序。当一页写满，就会自动开辟一个新的页
> 如果使用非自增主键（如果身份证号或学号等），由于每次插入主键的值近似于随机，因此每次新纪录都要被插到现有索引页得中间某个位置，此时MySQL不得不为了将新记录插到合适位置而移动数据，甚至目标页面可能已经被回写到磁盘上而从缓存中清掉，此时又要从磁盘上读回来，
>
> 这增加了很多开销，同时频繁的移动、分页操作造成了大量的碎片，得到了不够紧凑的索引结构，后续不得不通过OPTIMIZE TABLE来重建表并优化填充页面。
> 使用整形的自增主键，维护、查找B+树时，对索引比较及排序会更快，同时也能节约内存空间。

五十二、mysql最左前缀优化原则？

索引的底层是一颗B+树，那么联合索引的底层也就是一颗B+树，只不过联合索引的B+树节点中存储的是键值。由于构建一棵B+树只能根据一个值来确定索引关系，

所以MySQL创建联合索引的规则是首先会对联合索引的最左边第一个字段排序，在第一个字段的排序基础上，然后在对第二个字段进行排序。如果不通过匹配左边的字段时，相当于是在无序的表中进行全表查询。

五十三、MYSQL相关tips？

1、本身innodb可以没有主键。
2、innodb的页默认16kb。
3、表放在安装目录下的data文件夹中

五十四、Spring两种代理方式

若目标对象实现了接口，spring默认使用JDK的动态代理。
优点：因为有接口，所以使系统更加松耦合；
缺点：为每一个目标类创建接口；

若目标对象没有实现任何接口，spring使用CGLib进行动态代理。
优点：因为代理类与目标类是继承关系，所以不需要有接口的存在。
缺点：因为没有使用接口，所以系统的耦合性没有使用JDK的动态代理好。

五十五、简述Spring AOP的实现原理？

以CGLib为例，在行postProcessAfterInitialization阶段，通过ProxyFactoryBean获取动态代理对象。

其中代理类有一个叫做target的属性，值是父类对象（原对象）。

当我们调用代理方法时，触发CglibAopProxy.intercept()方法，获得一个拦截器链，按这个拦截器链来执行增强方法，其中业务逻辑主要是调用target.方法

五十六、BeanFactory 和 FactoryBean的区别？

BeanFactory是个Factory，也就是IOC容器或对象工厂，在Spring中，所有的Bean都是由BeanFactory(也就是IOC容器)来进行管理的，提供了实例化对象和拿对象的功能。

FactoryBean是个Bean，这个Bean不是简单的Bean，而是一个能生产或者修饰对象生成的工厂Bean,它的实现与设计模式中的工厂模式和修饰器模式类似。

五十七、Spring 是如何管理事务的，事务管理机制？

Spring的事务机制包括声明式事务和编程式事务。

编程式事务管理：Spring推荐使用TransactionTemplate，实际开发中使用声明式事务较多。

声明式事务管理：将我们从复杂的事务处理中解脱出来，获取连接，关闭连接、事务提交、回滚、异常处理等这些操作都不用我们处理了，Spring都会帮我们处理。

声明式事务管理使用了AOP面向切面编程实现的，本质就是在目标方法执行前后进行拦截。在目标方法执行前加入或创建一个事务，在执行方法执行后，根据实际情况选择提交或是回滚事务。

```
如何管理的：
Spring事务管理主要包括3个接口，Spring的事务主要是由它们(PlatformTransactionManager，
TransactionDefinition，TransactionStatus)三个共同完成的。
1. PlatformTransactionManager：事务管理器--主要用于平台相关事务的管理
    主要有三个方法：
        commit 事务提交；
        rollback 事务回滚；
        getTransaction 获取事务状态。

2. TransactionDefinition：事务定义信息--用来定义事务相关的属性，给事务管理器
PlatformTransactionManager使用
    这个接口有下面四个主要方法：
        getIsolationLevel：获取隔离级别；
        getPropagationBehavior：获取传播行为；
        getTimeout：获取超时时间；
        isReadOnly：是否只读（保存、更新、删除时属性变为false--可读写，查询时为true--只
读）
    事务管理器能够根据这个返回值进行优化，这些事务的配置信息，都可以通过配置文件进行配置。

3. TransactionStatus：事务具体运行状态--事务管理过程中，每个时间点事务的状态信息。

    例如它的几个方法：
        hasSavepoint()：返回这个事务内部是否包含一个保存点，
        isCompleted()：返回该事务是否已完成，也就是说，是否已经提交或回滚
        isNewTransaction()：判断当前事务是否是一个新事务

声明式事务的优缺点：
优点：不需要在业务逻辑代码中编写事务相关代码，只需要在配置文件配置或使用注解
（@Transaction），这种方式没有侵入性。
缺点：声明式事务的最细粒度作用于方法上，如果像代码块也有事务需求，只能变通下，将代码块变为方
法。
```

五十八、Spring 中用到了那些设计模式？

Spring框架中使用到了大量的设计模式，下面列举了比较有代表性的：

代理模式—在AOP和remoting中被用的比较多。

Spring AOP 就是基于动态代理的，如果要代理的对象，实现了某个接口，那么Spring AOP会使用JDK Proxy，去创建代理对象，而对于没有实现接口的对象，这时候Spring AOP会使用Cglib 生成一个被代理对象的子类来作为代理

单例模式—在spring配置文件中定义的bean默认为单例模式。

Spring默认将所有的Bean设置成 单例模式，即对所有的相同id的Bean的请求，都将返回同一个共享的Bean实例。这样就可以大大降低Java创建对象和销毁时的系统开销。

模板方法—用来解决代码重复的问题。比如. RestTemplate, jdbcTemplate, redisTemplate。

模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤的实现方式。Spring 中 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。

工厂模式—BeanFactory用来创建对象的实例。

Spring使用工厂模式可以通过 BeanFactory 或 ApplicationContext 创建 bean 对象。

适配器--spring mvc

每一个Controller都有一个适配器与之对应，而各个适配器Adapter又都是适配器接口HandlerAdapter的实现类，我们就可以统一通过适配器的hanle()方法来调用Controller中的用于处理请求的方法。

装饰器--spring data hashmapper

```
观察者-- spring 时间驱动模型
    定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。 spring中Observer模式常用的地方是listener的实现。如ApplicationListener。
回调--Spring ResourceLoaderAware回调接口
```

五十九、springboot自动配置的原理

```
在spring程序main方法中 添加@SpringBootApplication或者@EnableAutoConfiguration
会自动去maven中读取每个starter中的META-INF/spring.factories文件 该文件里配置了所有需要被创建spring容器中的bean
```

六十、什么情况下设置了索引但无法使用

（1）以“%”开头的 LIKE 语句，模糊匹配

（2）OR 语句前后没有同时使用索引

（3）存在索引列的数据类型隐形转换，则用不上索引，比如列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引

（4）where 子句里对索引列上有数学运算，用不上索引

（5）最左前缀原则

（6）如果mysql估计使用全表扫描要比使用索引快,则不使用索引，比如数据量极少的表

六十一、什么情况下不推荐使用索引？

1) 数据唯一性差（一个字段的取值只有几种时）的字段不要使用索引

比如性别，只有两种可能数据。意味着索引的二叉树级别少，多是平级。这样的二叉树查找无异于全表扫描。

```
2) 频繁更新的字段不要使用索引
    比如logincount登录次数，频繁变化导致索引也频繁变化，增大数据库工作量，降低效率。

3) 字段不在where语句出现时不要添加索引,如果where后含IS NULL /IS NOT NULL/ like ‘%输入符%’等条件，不建议使用索引
    只有在where语句出现，mysql才会去使用索引

4） where 子句里对索引列使用不等于（<>），使用索引效果一般
```