面试

Java基础

1. ### 集合

   - #### Collection

     - List
       ArrayList：底层结构是数组，线程不安全；
       LinkedList：底层结构是双向链表；
       Vector：底层结构是数组，线层安全；
     - Set
       HashSet：底层结构是HashMap，元素不可重复，可以用来去重；
       LinkedHashSet：继承HashSet；
       TreeSet：底层结构是红黑树；

   - #### Map

     - HashMap：JDK1.8之前底层结构是由数组+链表组成的;JDK1.8以后数据存储结构是一个 Node<K,V> 数组，底层结构是由数组+链表+红黑树（防止二叉树的在极限情况下变成链表）组成的，默认初始容量为16，负载因子为0.75，扩容为原来的2倍，线程不安全；
     - LinkedHashMap：LinkedHashMap 继承自 HashMap，所以它的底层仍然是基于拉链式散列结构即由数组和链表或红黑树组成；
     - HashTable： 底层结构是由数组+链表组成的，线程安全；
     - TreeMap： 底层结构是红黑树（自平衡的排序二叉树）；

Java中接口和抽象类的异同?

参数	抽象类	接口
声明	抽象类使用abstract关键字声明	接口使用interface关键字声明
实现	子类使用extends关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现	子类使用implements关键字来实现接口。它需要提供接口中所有声明的方法的实现
构造器	抽象类可以有构造器	接口不能有构造器
访问修饰符	抽象类中的方法可以是任意访问修饰符	接口方法默认修饰符是public。并且不允许定义为private 或者 protected
多继承	一个类最多只能继承一个抽象类	一个类可以实现多个接口
字段声明	抽象类的字段声明可以是任意的	接口的字段默认都是 static 和 final 的

this与super的区别

	super	this
代表的事物不同	super代表的是父类空间的引用	this代表的是所属函数的调用者对象
使用前提不同	super必须要有继承关系才能使用	this不需要继承关系也能使用
调用的构造函数不同	super：调用父类的构造函数	this：调用所属类的构造函数

IO 流

- InputStream/Reader: 所有的输入流的基类，前者是字节输入流，后者是字符输入流。
- OutputStream/Writer: 所有输出流的基类，前者是字节输出流，后者是字符输出流。

BIO,NIO,AIO 有什么区别?

- BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。
- NIO：Non IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。
- AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。

反射

什么是反射机制?

- JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

Java获取反射的三种方法

- 1.通过new对象实现反射机制

```
//方式一(通过建立对象)
Student stu = new Student();
Class classobj1 = stu.getClass();
System.out.println(classobj1.getName());
```

- 2.通过路径实现反射机制

```
 //方式二（所在通过路径-相对路径）
 Class classobj2 = Class.forName("fanshe.Student");
 System.out.println(classobj2.getName());
```

- 3.通过类名实现反射机制

```
//方式三（通过类名）
Class classobj3 = Student.class;
System.out.println(classobj3.getName());
```

Throwable

- **Error（错误）**

- Error 类及其子类。程序中无法处理的错误，表示运行应用程序中出现了严重的错误。通常有 Virtual MachineError（虚拟机运行错误）、NoClassDefFoundError（类定义错误）等。比如 OutOfMemoryError：内存不足错误；StackOverflowError：栈溢出错误。

- **Exception（异常）**
- 程序本身可以捕获并且可以处理的异常。Exception 这种异常又分为两类：运行时异常和编译时异常。
- **运行时异常**
- RuntimeException 类及其子类，表示 JVM 在运行期间可能出现的异常。比如NullPointerException空指针异常、ArrayIndexOutBoundException数组下标越界异常、ClassCastException类型转换异常、ArithmeticExecption算术异常。
- **编译时异常**
- Exception 中除 RuntimeException 及其子类之外的异常。Java 编译器会检查它。如果程序中出现此类异常，比如 ClassNotFoundException（没有找到指定的类异常），IOException（IO流异常），要么通过throws进行声明抛出，要么通过try-catch进行捕获处理，否则不能通过编译。在程序中，通常不会自定义该类异常，而是直接使用系统提供的异常类。**该异常我们必须手动在代码里添加捕获语句来处理该异常。**

Servlet

- **Servlet生命周期**

  - 加载阶段:加载并实例化(创建Servlet实例)
  - 初始化阶段:调用init()方法
  - 响应客户请求阶段：调用service()方法,doGet、doPost
  - 终止阶段：调用destroy()方法

Spring

Spring是**一个轻量级Java开发框架**，目的是为了**降低耦合度**，**解决企业级应用开发的复杂性，即简化Java开发**。

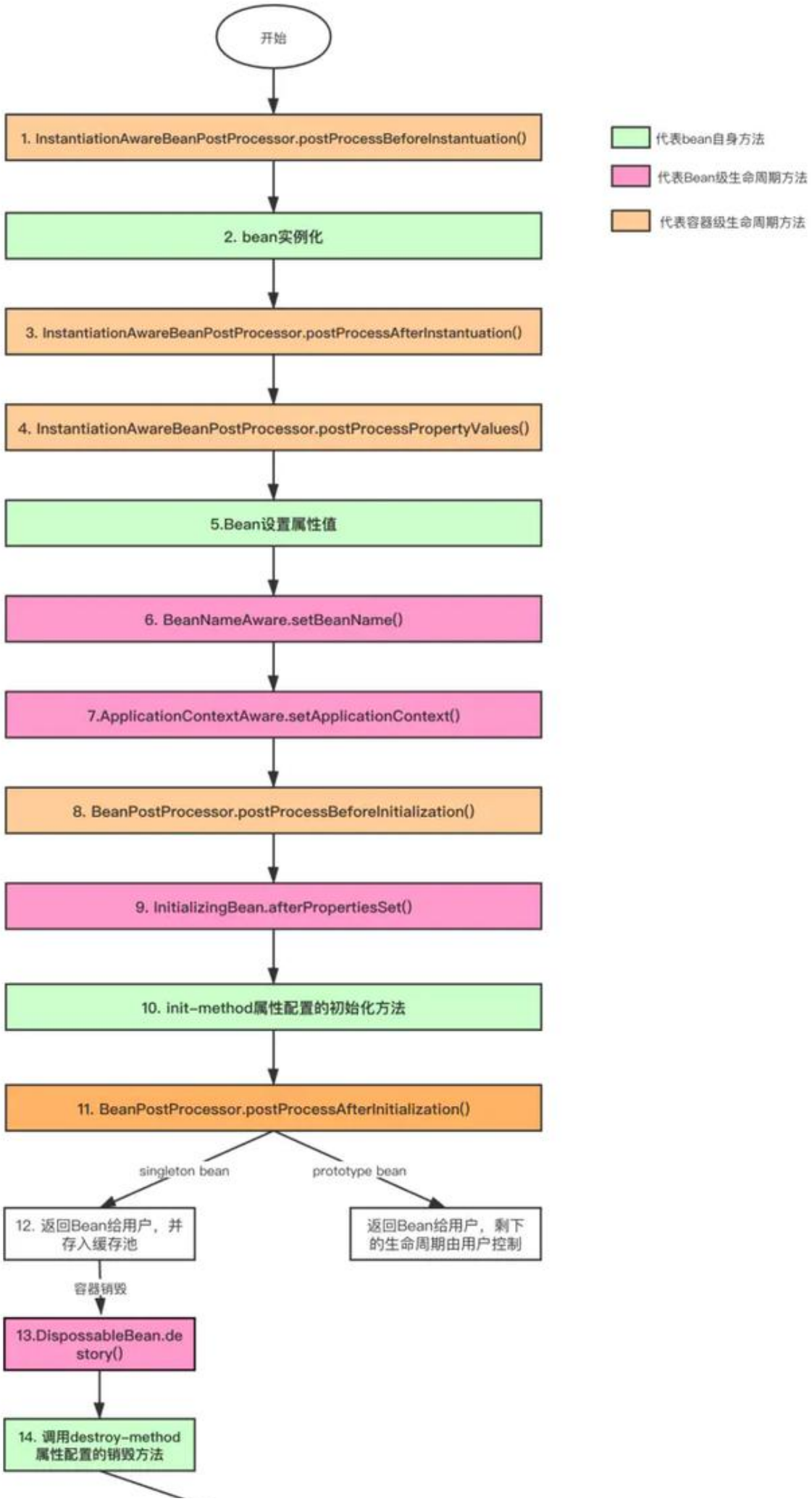
两个核心特性：**依赖注入（dependency injection，DI）和面向切面编程（aspect-oriented programming，AOP）**。

Spring框架的核心：IoC容器和AOP模块。通过IoC容器管理POJO对象以及他们之间的耦合关系；通过AOP以动态非侵入的方式增强服务。

Spring IOC 容器：

- 负责创建对象，管理对象（通过依赖注入（DI），装配对象，配置对象，并且管理这些对象的整个生命周期。
- 实现原理就是工厂模式加反射机制。

Bean的生命周期


开始
1. InstantiationAwareBeanPostProcessor.postProcessBeforeInstantuation()
2. bean实例化
3. InstantiationAwareBeanPostProcessor.postProcessAfterInstantuation()
4. InstantiationAwareBeanPostProcessor.postProcessPropertyValues()
5.Bean设置属性值
6. BeanNameAware.setBeanName()
7.ApplicationContextAware.setApplicationContext()
8. BeanPostProcessor.postProcessBeforeInitialization()
9. InitializingBean.afterPropertiesSet()
10. init-method属性配置的初始化方法
11. BeanPostProcessor.postProcessAfterInitialization()
singleton bean
prototype bean
12. 返回Bean给用户，并存入缓存池
返回Bean给用户，剩下的生命周期由用户控制
容器销毁
13.DispossableBean.destory()
14. 调用destroy-method属性配置的销毁方法
代表bean自身方法
代表Bean级生命周期方法
代表容器级生命周期方法

1.实例化

2.属性赋值

1. 如果这个Bean已经实现了BeanNameAware接口，会调用它实现的**setBeanName**(String)方法，传递的参数就是Spring配置文件中Bean的id值;
2. 如果这个Bean已经实现了BeanFactoryAware接口，会调用它实现的**setBeanFactory**(BeanFactory)，传递的是Spring工厂自身;
3. 如果这个Bean已经实现了ApplicationContextAware接口，会调用**setApplicationContext**(ApplicationContext)方法，传入Spring上下文;
4. 如果这个Bean关联了BeanPostProcessor接口，将会调用**postProcessBeforeInitialization**(Object obj, String s)方法，BeanPostProcessor经常被用作是Bean内容的更改，并且由于这个是在Bean初始化结束时调用那个的方法，也可以被应用于内存或缓存技术;

3.初始化(如果bean实现了InitializingBean接口，Spring将调用它们的after-PropertiesSet()方法。如果Bean在Spring配置文件中配置了init-method属性会自动调用其配置的初始化方法。)

1. 如果这个Bean关联了BeanPostProcessor接口，将会调用**postProcessAfterInitialization**(Object obj, String s)方法

4.销毁(当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean这个接口，会调用那个其实现的destroy()方法；)

Spring支持的事务管理类型

编程式事务管理：这意味你通过编程的方式管理事务，给你带来极大的灵活性，但是难维护。
声明式事务管理：这意味着你可以将业务代码和事务管理分离，你只需用注解和XML配置来管理事务。

Spring的事务传播行为

- ① **PROPAGATION_REQUIRED**：如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。
- ② **PROPAGATION_SUPPORTS**：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。
- ③ **PROPAGATION_MANDATORY**：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。
- ④ **PROPAGATION_REQUIRES_NEW**：创建新事务，无论当前存不存在事务，都创建新事务。
- ⑤ **PROPAGATION_NOT_SUPPORTED**：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
- ⑥ **PROPAGATION_NEVER**：以非事务方式执行，如果当前存在事务，则抛出异常。
- ⑦ **PROPAGATION_NESTED**：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则按REQUIRED属性执行。

Spring 的事务隔离

1. **ISOLATION_DEFAULT**：用底层数据库的设置隔离级别，数据库设置的是什么我就用什么；
2. **ISOLATION_READ_UNCOMMITTED**：未提交读，最低隔离级别、事务未提交前，就可被其他事务读取（会出现幻读、脏读、不可重复读）；
3. **ISOLATION_READ_COMMITTED**：提交读，一个事务提交后才能被其他事务读取到（会造成幻读、不可重复读），SQL server 的默认级别；

4. **ISOLATION_REPEATABLE_READ**：可重复读，保证多次读取同一个数据时，其值都和事务开始时候的内容是一致，禁止读取到别的事务未提交的数据（会造成幻读），MySQL 的默认级别；
5. **ISOLATION_SERIALIZABLE**：序列化，代价最高最可靠的隔离级别，该隔离级别能防止脏读、不可重复读、幻读。

Spring面向切面编程(AOP)

- AOP(Aspect-Oriented Programming)，一般称为面向切面编程，作为面向对象的一种补充，用于将那些与**业务无关**，但却对多个对象产生影响的**公共行为和逻辑**，抽取并封装为一个**可重用的模块**，这个模块被命名为“切面”（Aspect），减少系统中的**重复代码**，降低了模块间的**耦合度**，同时提高了系统的**可维护性**。可用于权限认证、日志、事务处理等。

JDK动态代理和CGLIB动态代理的区别

- JDK动态代理只提供接口的代理，不支持类的代理。核心InvocationHandler接口和Proxy类，InvocationHandler 通过invoke()方法反射来调用目标类中的代码，动态地将横切逻辑和业务编织在一起；接着，Proxy利用 InvocationHandler动态创建一个符合某一接口的的实例, 生成目标类的代理对象。
- 如果代理类没有实现 InvocationHandler 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**的生成指定类的一个子类对象，并覆盖其中特定方法并添加增强代码，从而实现AOP。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。

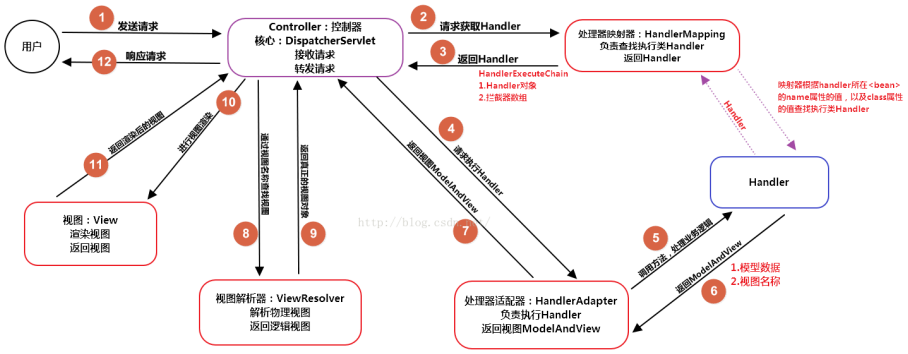
Spring MVC

- Spring MVC是一个基于Java的实现了MVC设计模式的请求驱动类型的轻量级Web框架
- 前端控制器(dispatcherServlet) , 请求到处理器映射（handlerMapping), 处理器适配器（HandlerAdapter), 视图解析器（ViewResolver）。

DispatcherServlet

- **工作流程**

  (1）用户发送请求至前端控制器DispatcherServlet；
  （2） DispatcherServlet收到请求后，调用HandlerMapping处理器映射器，请求获取Handle；
  （3）处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet；
  （4）DispatcherServlet 调用 HandlerAdapter处理器适配器；
  （5）HandlerAdapter 经过适配调用 具体处理器(Handler，也叫后端控制器)；
  （6）Handler执行完成返回ModelAndView；
  （7）HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet；
  （8）DispatcherServlet将ModelAndView传给ViewResolver视图解析器进行解析；
  （9）ViewResolver解析后返回具体View；
  （10）DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）
  （11）DispatcherServlet响应用户。



Spring Boot

Spring Boot 是 Spring 开源组织下的子项目，是 Spring 组件**一站式解决方案**，主要是**简化了使用 Spring 的难度**，**简省了繁重的配置**，提供了各种启动器，开发者能快速上手。

Spring Boot 的核心注解

- 启动类上面的注解是@**SpringBootApplication**，它也是 Spring Boot 的核心注解，主要组合包含了以下 3 个注解：

1. @SpringBootConfiguration：组合了 @Configuration 注解，实现配置文件的功能。
2. @EnableAutoConfiguration：打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能： @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。
3. @ComponentScan：Spring组件扫描。

Spring Cloud

五大组件Nocas

- 注册中心（EureKa/Nocas）
- 网关（Zuul/Gateway）
- 负载均衡(Ribbon)
- 服务调用(Ferign/OpenFeign)
- 熔断器(Hystix)
- 配置管理(Config/Nacos)

SpringBoot和SpringCloud的区别

- SpringBoot专注于快速方便的开发单个个体微服务。
- SpringCloud是关注全局的微服务协调整理治理框架，它将SpringBoot开发的一个个**单体微服务整合并管理**起来，为各个微服务之间提供，配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等集成服务
- SpringBoot可以离开SpringCloud独立使用开发项目， 但是SpringCloud离不开SpringBoot ，属于依赖的关系
- SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架。

Mybatis如何执行批量操作

foreach的主要用在构建in条件中，它可以在SQL语句中进行迭代一个集合。foreach标签的属性主要有item，index，collection，open，separator，close。

item　　表示集合中每一个元素进行迭代时的别名，随便起的变量名；

index　　指定一个名字，用于表示在迭代过程中，每次迭代到的位置，不常用；

open　　表示该语句以什么开始，常用“(”；

separator表示在每次进行迭代之间以什么符号作为分隔符，常用“,”；

close　　表示以什么结束，常用“)”。

mybatis的一级缓存和二级缓存

一级缓存

MyBatis的一缓存其实就是一个sqlsession级别的，意思就是sqlsession只能访问自己的一级缓存的数据。一级缓存查询存在于每一个的sqlsession类的实例对象中，当第一次查询某一个数据时候，sqlsession类的实例对象会将该数据存入到一级缓存。这样可以在没有收到改变该数据的请求之前，我们所查询数据都是从缓存中去获取的，而不是从数据库中去取数据，这样就大大减少数据库的频繁查询，导致效率降低的原因。

一级缓存原理

首先用户第一次查询sql时候，sql的查询结果就会被写入sqlsession一级缓存中的，这样用户第二次查询时，直接从一级缓存取出数据，而不是数据库。

如果用户出现commit操作时，比如增删改查，这时sqlsession中一级缓存区域就会全部清空。清空之后再次去一级缓存查找不到，就会走数据库进行查找，然后再次存到缓存中。注意：缓存使用的数据结构也是map的。

二级缓存

二级缓存的范围就是mapper级别，也就是mapper以命名空间为单位创建缓存数据结构，也是map结构。二级缓存和 一级缓存一样的是，二级缓存的多个sqlsession去操作同一个mapper映射的sql语句，然后多个sqlsession可以共用二级缓存这样的一个思想，它是跨sqlsession的。

怎么判断某两次查询是完全相同的查询?

mybatis认为，对于两次查询，如果以下条件都完全一样，那么就认为它们是完全相同的两次查询。

2.1 传入的**statementId**

2.2 查询时要求的**结果集中的结果范围**

2.3. 这次查询所产生的最终要传递给JDBC java.sql.Preparedstatement的**Sql语句字符串**（boundSql.getSql() ）

2.4 传递给java.sql.Statement要设置的**参数值**

MyBatis实现一对一，一对多有几种方式，怎么操作的?

有联合查询和嵌套查询。联合查询是几个表联合查询，只查询一次，通过在resultMap里面的association，collection节点配置一对一，一对多的类就可以完成

嵌套查询是先查一个表，根据这个表里面的结果的外键id，去再另外一个表里面查询数据，也是通过配置association，collection，但另外一个表的查询通过select节点配置。

为什么要用 Redis /为什么要用缓存

避免频繁访问数据库，从而浪费资源，降低性能

持久化

持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。

Redis 提供两种持久化机制 RDB（默认） 和 AOF 机制:

RDB：持久化

RDB是Redis默认的持久化方式。按照一定的时间将内存的数据以快照的形式保存到硬盘中，对应产生的数据文件为dump.rdb。通过配置文件中的save参数来定义快照的周期。

优点：

1、只有一个文件 dump.rdb，方便持久化。

2、容灾性好，一个文件可以保存到安全的磁盘。

3、性能最大化，fork 子进程来完成写操作，让主进程继续处理命令，所以是 IO 最大化。使用单独子进程来进行持久化，主进程不会进行任何 IO 操作，保证了 redis 的高性能

4.相对于数据集大时，比 AOF 的启动效率更高。

缺点：

1、数据安全性低。RDB 是间隔一段时间进行持久化，如果持久化之间 redis 发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候)

AOF：持久化

AOF持久化(即Append Only File持久化)， 是指所有的命令行记录以 redis 命令请 求协议的格式完全持久化存储)保存为 aof 文件，当重启Redis会重新将持久化的日志中文件恢复数据。

优点：

1、数据安全，aof 持久化可以配置 appendfsync 属性，有 always，每进行一次 命令操作就记录到 aof 文件中一次。

2、通过 append 模式写文件，即使中途服务器宕机，可以通过 redis-check-aof 工具解决数据一致性问题。

3、AOF 机制的 rewrite 模式。AOF 文件没被 rewrite 之前（文件过大时会对命令 进行合并重写），可以删除其中的某些命令（比如误操作的 flushall））

缺点：

1、AOF 文件比 RDB 文件大，且恢复速度慢。

2、数据集大的时候，比 rdb 启动效率低。

优缺点是什么?

AOF文件比RDB更新频率高，优先使用AOF还原数据。

AOF比RDB更安全也更大

RDB性能比AOF好

如果两个都配了优先加载AOF

Redis的过期键的删除策略

定时过期：每个设置过期时间的key都需要创建一个定时器，到过期时间就会立即清除。

惰性过期：只有当访问一个key时，才会判断该key是否已过期，过期则清除。

定期过期：每隔一定的时间，会扫描一定数量的数据库的expires字典中一定数量的key，并清除其中已过期的key。

Redis key的过期时间和永久有效分别怎么设置?

EXPIRE和PERSIST命令。

事务管理（ACID）概述

原子性（Atomicity）

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

一致性（Consistency）

事务前后数据的完整性必须保持一致。

隔离性（Isolation）

多个事务并发执行时，一个事务的执行不应影响其他事务的执行

持久性（Durability）
持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响
Redis的事务总是具有ACID中的一致性和隔离性，其他特性是不支持的。当服务器运行在AOF持久化模式下，并且appendfsync选项的值为always时，事务也具有耐久性。

Redis集群最大节点个数是多少?

16384个

集群方案

哨兵模式

主要有以下功能:

- 集群监控：负责监控 redis master 和 slave 进程是否正常工作。
- 消息通知：如果某个 redis 实例有故障，那么哨兵负责发送消息作为报警通知给管理员。
- 故障转移：如果 master node 挂掉了，会自动转移到 slave node 上。
- 配置中心：如果故障转移发生了，通知 client 客户端新的 master 地址。

缓存异常

缓存雪崩

缓存雪崩是指缓存同一时间大面积的失效，所以，后面的请求都会落到数据库上，造成数据库短时间内承受大量请求而崩掉。
解决方案

- 为热点数据设定超时时间时 采用随机数. 10秒+随机数 保障数据不会同时失效.
- 设定多级缓存

缓存穿透

缓存穿透是指缓存和数据库中都没有的数据，导致所有的请求都落到数据库上，造成数据库短时间内承受大量请求而崩掉。
解决方案

- 接口层增加校验
- 采用布隆过滤器

缓存击穿

说明: 在高并发条件下 用户频繁访问**一个热点数据**. 但是该热点数据在缓存中失效.导致用户直接访问数据库.
解决方案

- 设置热点数据永远不过期。
- 加互斥锁

缓存预热

缓存预热就是系统上线后，将相关的缓存数据直接加载到缓存系统。

缓存降级

当访问量剧增、服务出现问题，**缓存降级**的最终目的是保证核心服务可用，即使是有损的。而且有些服务是无法降级的（如加入购物车、结算）。

为什么使用MQ？MQ的优点

- 异步处理 - 相比于传统的串行、并行方式，提高了系统吞吐量。
- 应用解耦 - 系统间通过消息通信，不用关心其他系统的处理。
- 流量削锋 - 可以通过消息队列长度控制请求量；可以缓解短时间内的高并发请求。
- 日志处理 - 解决大量日志传输。
- 消息通讯 - 消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等。

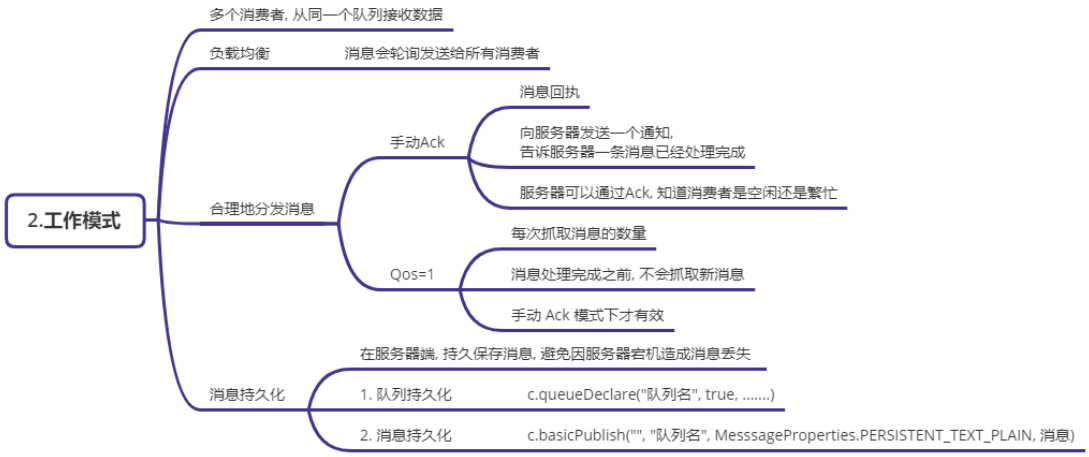
rabbitmq六种工作模式

```
/*
 * 声明队列,会在rabbitmq中创建一个队列
 * 如果已经创建过该队列，就不能再使用其他参数来创建
 *
 * 参数含义:
 *   -queue: 队列名称
 *   -durable: 队列持久化,true表示RabbitMQ重启后队列仍存在
 *   -exclusive: 排他,true表示限制仅当前连接可用
 *   -autoDelete: 当最后一个消费者断开后,是否删除队列
 *   -arguments: 其他参数
 */
ch.queueDeclare("helloworld", false,false,false,null);
```

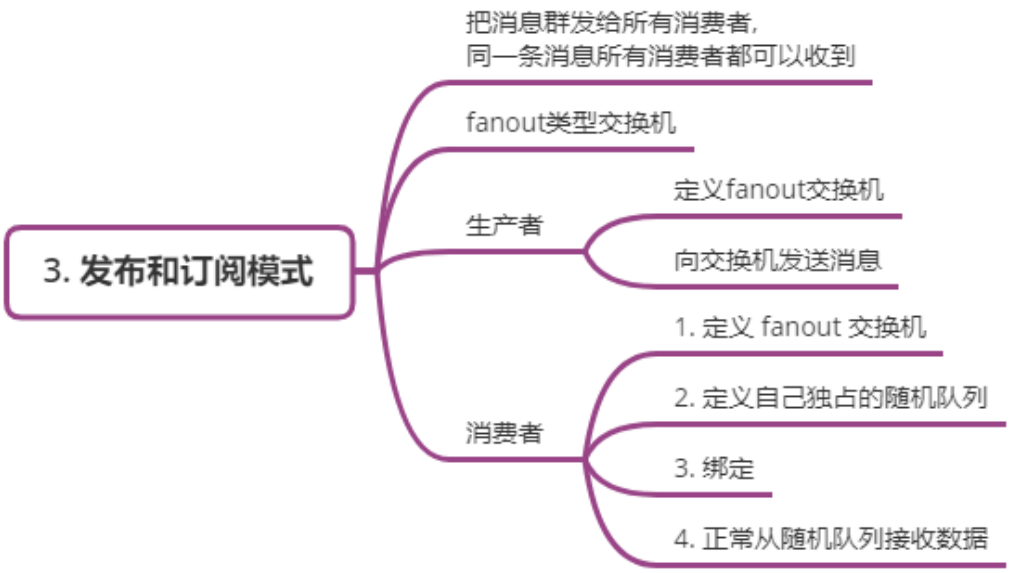
简单模式

- 发送消息的程序是生产者
- 队列就代表一个邮箱。虽然消息会流经RbbitMQ和你的应用程序，但消息只能被存储在队列里。队列存储空间只受服务器内存和磁盘限制，它本质上是一个大的消息缓冲区。多个生产者可以向同一个队列发送消息，多个消费者也可以从同一个队列接收消息.
- 消费者等待从队列接收消息

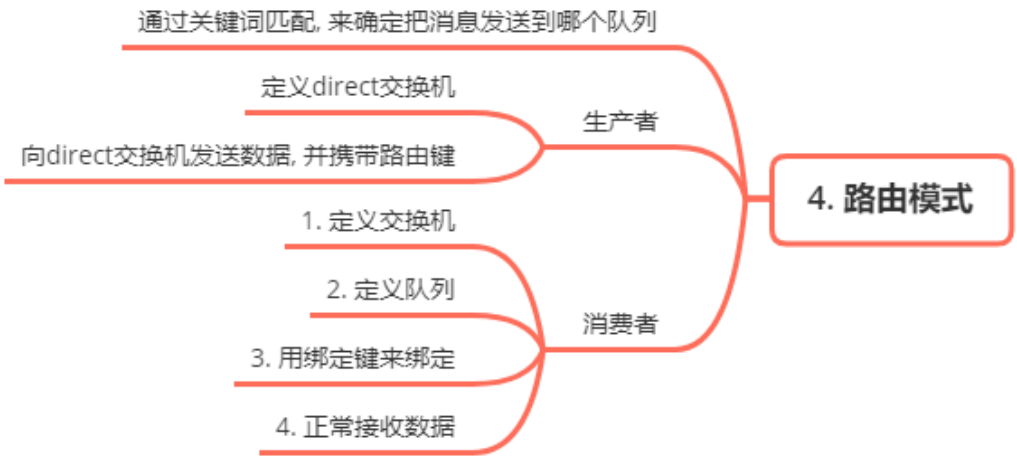
工作模式



发布订阅模式



路由模式



主题模式

SQL 优化的常见方法有哪些

- 避免 SELECT *
- 永远为每张表设置一个 ID
- 尽可能的使用 NOT NULL
- 查询条件减少使用函数，避免全表扫描
- 减少不必要的表连接
- 有些数据操作的业务逻辑可以放到应用层进行实现
- 可以使用with as
- 尽量避免使用游标，因为游标的效率较差
- 不要把SQL语句写得太复杂
- 不能循环执行查询
- 用 exists 代替 in
- 表关联关系不要太纠结
- 查询多用索引列取查，用charindex或者like[0-9]来代替%%
- inner关联的表可以先查出来，再去关联leftjoin的表
- 可以进行表关联数据拆分，即先查出核心数据，再通过核心数据查其他数据，这样会快得多
- 参考SQL执行顺序进行优化
- 表关联时取别名，也能提高效率
- 使用视图，给视图建立索引进行优化

- 表关联的数据集中抽取存入一张表中，查询时单表查询，提高了查询效率

索引

索引**是一个排序的列表**，在这个列表中存储着索引的值和包含这个值的数据所在行的物理地址
提高检索效率，但事后增加内存资源的消耗

分布式事务

Seata AT基本原理

Seata AT 事务分两个阶段来管理全局事务：
第一阶段： 执行各分支事务
第二阶段： 控制全局事务最终提交或回滚

TCC 基本原理

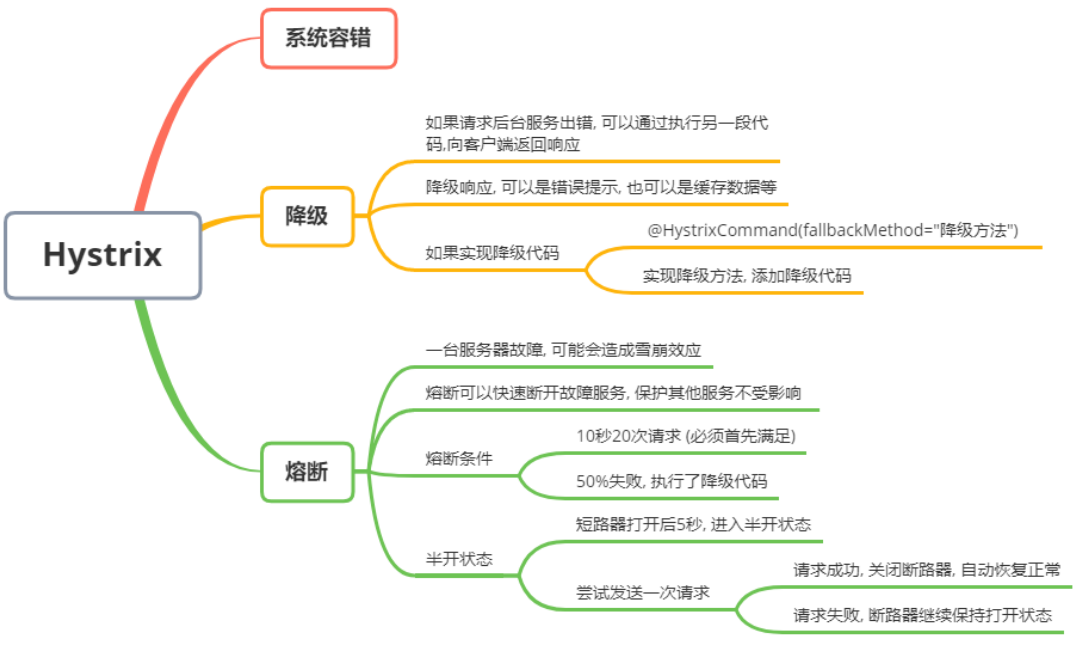
TCC 与 Seata AT 事务一样都是**两阶段事务**，它与 AT 事务的主要区别为：

- **TCC 对业务代码侵入严重**
  每个阶段的数据操作都要自己进行编码来实现，事务框架无法自动处理。
- **TCC 效率更高**
  不必对数据加**全局锁**，允许多个事务同时操作数据。

产生死锁的四个必要条件

- 互斥
- 不可剥夺
- 请求与保持vue
- 循环等待

Hystrix 断路器



我做了TIS的信用卡项目，主要完成了资金管理，征信，会员等功能，数据库是Oracle，前台用到jsp等技术，后台用到Java,maven,,

我做了人人商城，主要完成了商品服务，存储服务，会员等功能，数据库是MySQL，前台用到vue等技术，后台用到Java的SSM,springboot,springcloud,，
我主要是做了开发，但在开发前，我在项目经理的带领下参与了业务调研，数据库设计等工作，后期我参与了测试和部署工作。