## java基础

### int和Integer的区别

### IO与NIO



NIO三大核心内容selector（选择器用于监听管道channel）、管道channel(通道)、buffer(缓冲区)

### java对象克隆

实现cloneable接口

1、Integer是int的包装类，int则是java的一种基本数据类型

2、Integer变量必须实例化后才能使用，而int变量不需要

3、Integer实际是对象的引用，当new一个Integer时，实际上是生成一个指针指向此对象；而int则是直接存储数据值 。

4、Integer的默认值是null，int的默认值是0

## 1.java集合

### List

List为接口，其实现类为ArrayList和LinkedList。

ArrayList底层结构为数组。LinkedList底层结构为链表。

#### 为什么不用数组

**java本身有数组，为什么还要用ArrayList呢？**

因为java数组在定义时必须要指定长度，相当于数组是定长的，如果初始化数组大小太大的话会浪费内存，太小了不够用。但ArrayList不同，它可以动态扩容。

#### 如何扩容

**ArrayList是如何实现动态扩容的？**

首先在new ArrayList()时，会默认生成一个长度为0的Object数组，调用add()方法都会去计算这个数组空间是否够用，第一次调用如果够用直接追加元素，如果不够就要扩容，扩容依靠的是ArrayList中的grow()方法，每次扩容为原来的1.5倍。第一次调用add()方法时，由于数组长度为0，会调用grow()将数组长度增加到10，之后每次扩容都为原来的1.5倍。

#### 为什么用的最多

**为什么ArrayList在日常开发用的最多？**

由于底层是数组，日常开发中，遍历比增删要多，即便是增删也是往往在尾部，时间复杂度也就O(1),另外ArrayList增删调用的copyOf()被优化过

#### Vector

**Vector也是List的实现类，理解吗？**

vector底层也是数组，现在用的比较少，相对于ArrayList,它是线程安全的，扩容的时候扩容2倍

#### 线程安全CopyOnWrite

**如果不用Vector，还有哪些线程安全的List？**

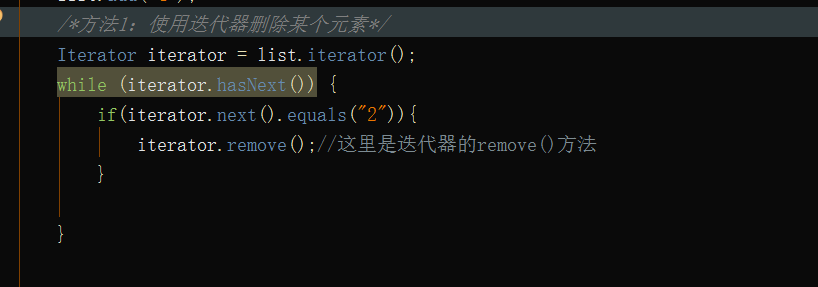
CopyOnWriteArrayList是线程安全的，CopyOnWrite容器很常见（通俗地讲，当我们往容器中添加一个元素的时候，不是直接添加，而是对当前容器copy，复制一个容器，在这个复制的容器中添加元素，添加完之后，再将引用指向这个新容器。）

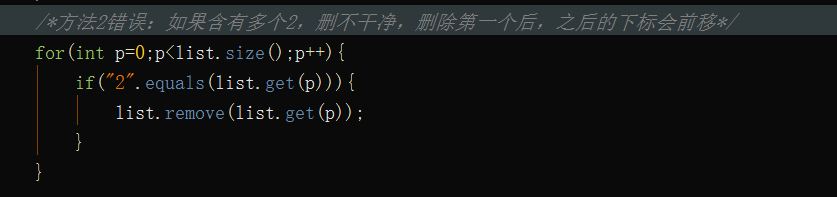
缺点：

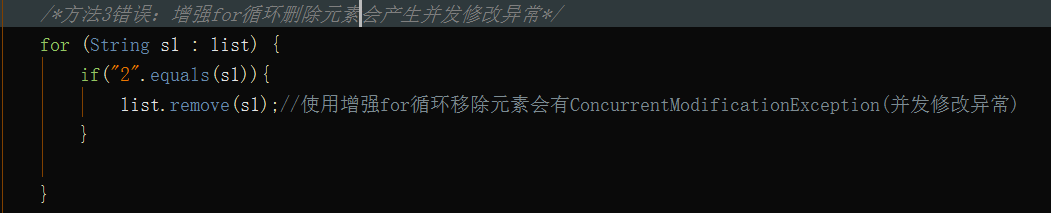
1.内存占用问题，产生了两个容器

2.只能保持数据的最终一致性，无法保持实时性

#### 移除元素Iterator

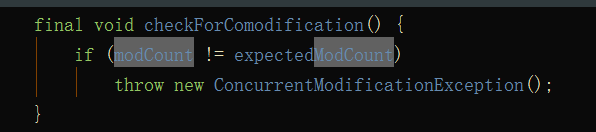






增强for修改元素报异常的原因：

在遍历时都会调用下面的checkForComodificaiton()方法，如果有元素被移除或者增加了就会导致modCount！=expectedMoCount，从而报异常。



### Map

Map是java中util包下的一个接口。常见的实现类有HashMap、LinkedHashMap、TreeMap、ConcurrentHashMap

#### 底层结构

哈希表是由数组+链表组成

HashMap、ConcurrentHashMap底层为数组+链表/红黑树

LinkedHashMap底层为数组+链表+双向链表

TreeMap底层结构为红黑树

#### HashMap初始值、负载因子

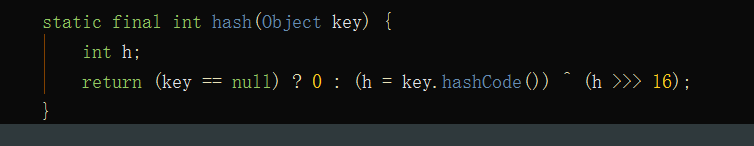
HashMap有几个构造方法，可以指定它的初始大小和负载因子。如果给参数，默认HashMap的大小为16，负载因子的大小为0.75；

public HashMap(int initialcapacity,float loadFactor){}为HashMap的有参构造器，参数分别为初始值大小和负载因子大小；如果初始值传的7，我们会将HashMap的初始大小定义为8（2次幂大小），是因为HashMap里采用的是**位运算**来代替取模，当我们想要把元素放进HashMap的时候，需要取出这个元素位置，这一步利用位运算效率更高。

负载因子：

它的大小决定的是哈希表的扩容和哈希冲突，比如我们默认的HaspMap的大小为16，这样最多能put12个元素，当put第13个元素时，就需要扩容，12/16=0.75；鉴于上面的位运算，HashMap的大小只能是2次幂，所以HashMap每次扩容倍数为原来的2倍

#### put元素时key如何计算哈希值



首先计算正常的hash值，然后与高16位做异或运算，产生最终的hash值，这样做可以减少hash冲突的概率。

#### HashMap什么时候用红黑树

当数组的大小大于64且链表大小大于8时，才会将链表改为红黑树；当红黑树大小为6时，会退化为链表。他们之间的转换主要是出于查询和插入的性能问题。

#### 线程安全的Map

juc包下的ConcurrentHashMap,它通过部分加锁和CAS算法来实现同步

### Set

## 2.java中抽象类和接口的区别

相同点：

1.都不能进行实例化

2.继承抽象类和实现接口都要对其中的抽象方法全部实现

区别：

接口都是公开抽象的方法，没有构造器；抽象类可以有普通方法，并且会有默认无参的构造器

## 3.redis

### linux配置

redis安装：

解压：tar –zxvf redis-5.0.7.tar.gz

将解压后的文件移动到/usr/local ： mv redis-5.0.7 /usr/local/

安装gcc环境：yum install gcc c++

安装redis：进入redis-5.0.7后执行make命令，之后执行make install

移动并修改redis配置文件：在/usr/local/ redis-5.0.7创建bin、etc：mkdir bin/mkdir etc

mv redis.conf /usr/local/ redis-5.0.7/etc

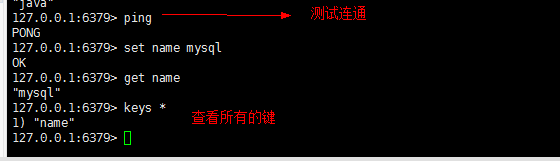
vim redis.conf后将daemonize调整为yes，代表后台运行

redis启动：

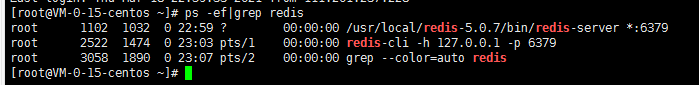
redis-server /usr/local/ edis-5.0.7/etc/redis.conf 通过指定的配置文件启动redis

redis-cli –h xxx -p 6379 启动客户端，并连接机器ip为xxx，端口号为6379的redis

连接测试：



查看redis进程：



注销redis服务：



### 分布式锁

### 高可用之Cluster集群部署模式

Redis支持主从同步，Cluster集群部署时，通过Sentinel哨兵来监控Redis主服务器的状态。当主节点Master挂掉时，在从节点slave中根据一定的策略选举出新主节点

## 4.spring

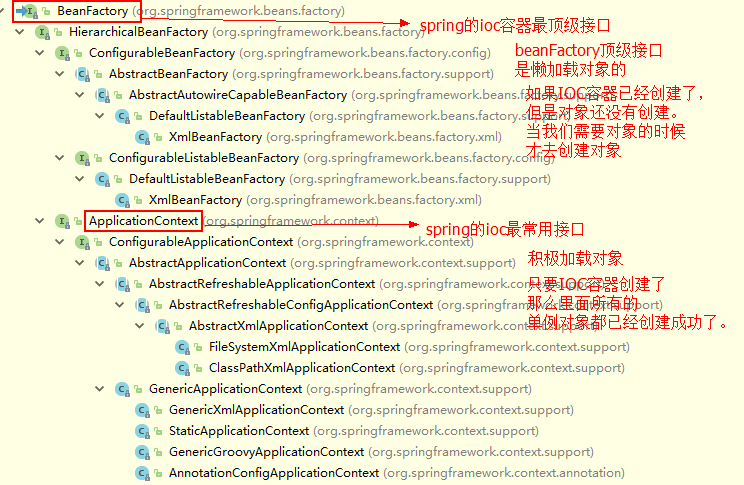
### IOC概念：

Inversion of Control控制反转,创建和管理对象的权利交给了IOC，当我们想要使用对象的时候就去IOC容器中取。

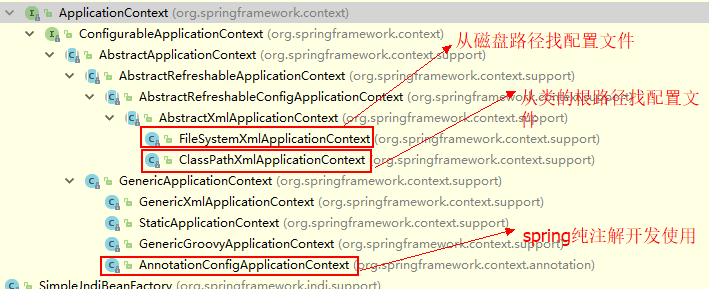
### Spring的四大核心包

beans、context、core、expression

### IOC常见接口



以及常用实现类



### 循环依赖

spring内部有三级缓存：

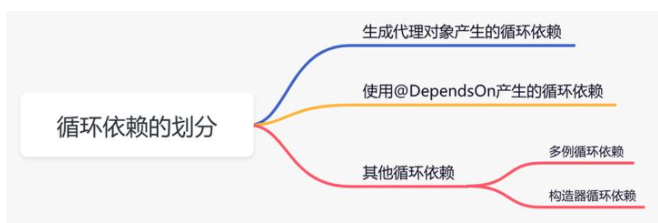
一级缓存，用于保存实例化、注入、初始化完成的bean实例

二级缓存，用于保存提前曝光（属性还未注入）的单例对象

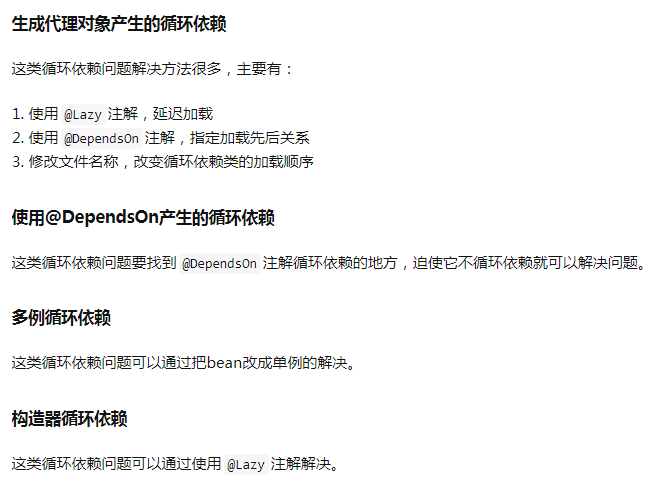
三级缓存，用于保存要被实例化的对象的对象工厂

问题：

如果项目中出现了循环依赖，就说明是spring本身无法解决的循环依赖，目前包含下列几种：



解决办法：



### bean的常用作用域

singleton:单例，当用scope设置为singleton时，加载spring配置时就会创建单例对象

prototype:多例，当scope设置为prototype时，在调用getBean()时才创建多例对象



### bean的生命周期

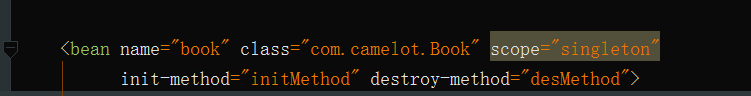
1.调用bean对象所属类的无参构造方法

2.调用类中的set方法进行属性注入

3.调用init方法进行对象初始化（在类中自定义initMethod）

4.得到bean，开始使用

5.销毁bean，在类中定义desMethod,调用ClassPathXmlApplicationContext的close方法



### IOC和DI常用注解

将该类对象放在IOC容器的注解，需要配合开启包扫描一起使用，否则找不到：

可以下边几个注解为传统配置文件中的<bean id=“” class=“”></bean>，这一步也相当于调用了该类的无参构造方法

@Component：公用

@Controller：放controller层

@Service：放service层

@Repository：放dao层

属性注入的注解：

下边几个注解可以理解为传统spring配置文件中的<property name=”” value=””></property>,这一步相当调用了类中的set方法，属性注入的注解就可以理解为封装了set方法

@Autowired:spring包下的，根据类型注入

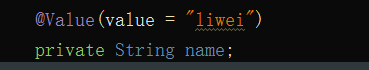
@Qualifier：spring包下的，根据名称注入，需要和@Autowired一起使用



@Resource：javax包下的，可以根据类型，可以根据名称



@Value



### AOP概念：

面向切面编程，降低耦合度

举例：本来有一个简单的登录功能，现在需要将该功能实现权限管理，如管理员与非管理员

这时候，修改登录的源代码的方式来实现，而是增加一个权限判断模块，通过配置引入到之前的登录中，之后不使用了，就可以取消配置 。

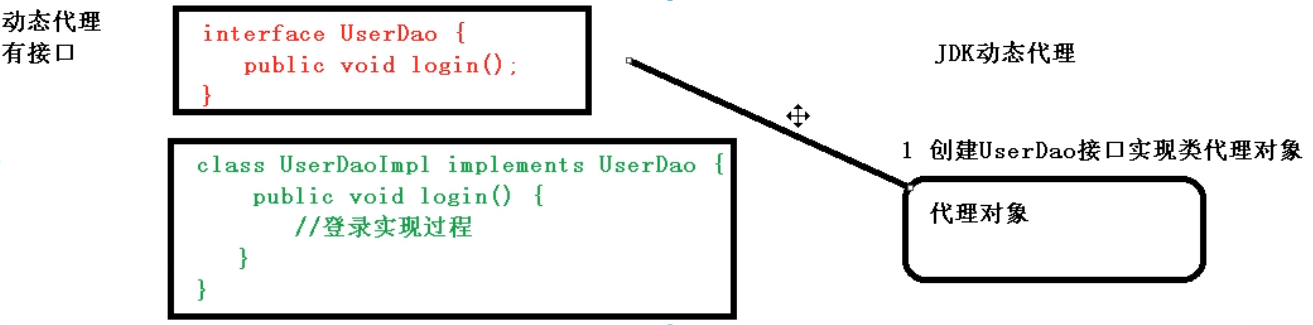
通俗描述：不通过修改源代码的方式， 在主干功能里添加新功能

### AOP底层解释

底层使用动态代理

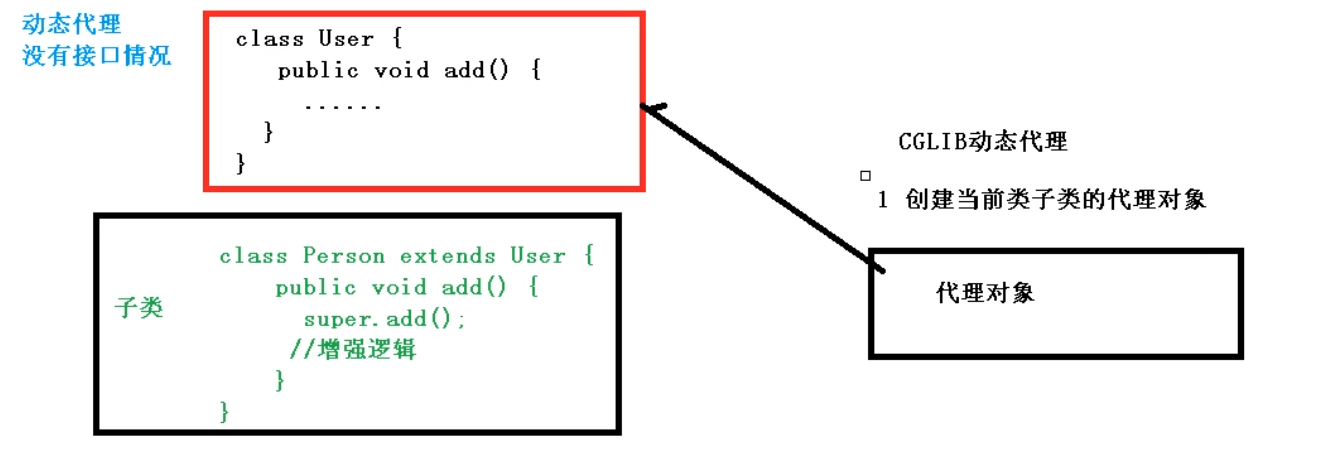
有接口的情况，使用JDK动态代理：

使用JDK动态代理，创建一个接口的代理对象，增强类的方法



没有接口的情况，使用CGLIB动态代理：

使用CGLIB动态代理，创建子类的代理对象，增强类的方法



### AOP术语

连接点：类里可以被增强的方法，称为连接点

切入点：实际被增强的方法，称为切入点

通知（增强）：

（1）增强的逻辑部分称为通知（如登录时的权限控制）

（2）通知有几种类型

前置通知

后置通知

环绕通知

异常通知

最终通知

切面：是个动作，将通知（增强的逻辑）应用到切入点的过程

### AOP应用场景：

日志记录

事务管理

线程池关闭

### spring事务：

分为编程式事务和声明式事务

常用注解方式的声明式事务

@Transactional()注解的属性介绍：

### spring事务的传播行为有7种

常用两种如下：

REQUIRED：调用方add()和被调用方update()，如果add()方法有事务，调用update()方法之后，update()会使用add()方法中的事务；如果add()方法没事务，调用update()方法后，创建新事务

REQUIRED\_NEW：使用add()方法调用update()方法，无论add()是否有事务，都创建新的事务

## mybatis：

### #{}和${}的区别

#{}：底层使用的preparedStatement,能够防止sql注入

${}: 底层是使用的是statement

### mybatis的执行流程

1.mybatis的配置文件包含全局配置文件和mapper映射文件，全局配置文件包含数据源、事务等，mapper映射文件包含sql执行相关的信息

2.通过加载mybatis的配置文件，构造出sqlSessionFactory工厂对象

3.通过sqlSessionFactory工厂可以创建出sqlsession会话对象

4.sqlsession本身并不能操作数据库，而是通过底层的Excutor执行器接口来操作数据库的

## springmvc

### 执行流程

1.首先dispactcherservlet前端控制器接收前端请求，对url进行解析。

2.前端控制器会调用处理器映射器handlermapping，处理器映射器根据请求地址找到对应的处理器方法，返回前端控制器。

3.前端控制器调用处理器适配器handleradapter得到model返回给前端控制器。

4.前端控制器调用视图解析器viewReslover解析生成view，返回给前端控制器。

5.前端控制器进行视图响应，展示数据

## springboot

### Spring Boot的核心注解有哪些？

启动类上的@SpringBootApplication为最核心的注解

该注解主要包含如下3个注解

@SpringBootConfiguration:组合了@Configuration注解，实现配置文件的功能

@EnableAutoConfiguration:打开自动配置的功能

@ComponentScan：组件扫描

### 如何禁用某些自动配置特性

@EnableAutoConfiguration(exclude=xxx)：表示自动配置的特性不包含xxx

### 如何在Spring Boot启动的时候运行一些特定的代码

可以实现ApplicationRunner或者CommandLineRunner接口，这两个接口实现方式一样，都提供了一个run()方法

### springboot启动流程

1.构造SpringApplication的实例

2.调用SpringApplication.run()方法

3.构造SpringApplicationRunListeners实例

4.发布ApplicationStartedEvent事件

5.SrpingApplicationRunListeners实例准备环境信息

6.创建ApplicationContext对象

7.ApplicationContext实例准备环境信息

8.刷新上下文

## Dubbo：

### RPC框架

RPC：远程过程调用

RPC框架的性能核心：通信、序列化

### 注册中心

zookeeper：端口号2181

### zookeeper宕机

部分宕机：会自动切换到其他注册中心

全部宕机：服务提供者和服务消费者能通过本地缓存通讯

### 负载均衡机制

随机调用<默认>

基于权重的轮询

最少活跃数

一致性哈希

## 多线程：

### 进程/线程

### java中获取多线程的方式

1.继承Thread

2.实现Runable接口，并重写run方法

3.实现Callable接口，并重写call方法

### sleep方法和wait方法的区别

sleep方法和wait方法都是用来设定放弃cpu特定时间，不同点是如果线程有否个对象的监视器(锁)，sleep方法不会释放监视器，wait方法会释放监视器

### volatile关键字

多线程主要围绕可见性和原子性展开

1.保证了线程间的可见性，即每次读到volatile变量，一定是最新的数据

2.volatile一个重要的作用的是和CAS结合，保证了原子性

### 线程安全等级

同一段代码在足够多的线程和单线程下执行结果永远一致，那么就是线程安全的

不可变：String、Integer、Long

绝对线程安全：CopyOnWriteArrayList

相对线程安全：Vector

线程非安全：ArrayList、LinkList、HashMap

### 获取线程堆栈（dump文件）

步骤1：通过ps –ef|grep java获取线程的pid

步骤2：jstack pid命令

### 一个线程出现运行时异常会怎样

线程停止、持有的锁被释放

### 两个线程直接共享数据

通过共享对象

## Nginx

### Nginx是如何处理一个http请求的

Nginx会根据过来的http请求头里的Host字段里的值，来判断使用哪个server{}

## 消息中间件：

### 消息队列模式（JMS规范）

点对点：（point to point,queue），不可重复消费

消息生产者生产消息发送到queue中，然后消费者从queue中取出并消费，消息被消费后，queue中不再存储，所以被被消费的消息不会被重复消费，queue支持存在多个消费者，但只有一个消费者可以消费

发布/订阅：(publish/subcribe，topic)，可以重复消费

消息生产者（发布）消息到topic中，同时有多个消费者（订阅）该消息

## mysql：

### 什么是索引

存储在表中一个特定列的值的数据结构

Hash结构：让查询、插入、修改、删除的平均时间复杂度为O(1)

B+树：可以让查询、插入、修改、删除的平均时间复杂度为O(log(n))

### 索引类型

Primary Key:主键

Unique:唯一索引，所标列数据不能重复，但可以为NULL

Index:普通索引，提升查询性能

Full Text:全文检索

### 索引的创建与删除

创建：create index idx\_username on tbl\_user(username);

删除：alter table tbl\_user drop index idx\_username;

### 索引失效

1. 索引失效（应该避免）
   1. 全值匹配我最爱 ：全值匹配是最好的，及建了多少索引都用了
   2. 最佳左前缀法则：如果索引了多例，要遵守最左前缀法则。指的是查询从索引的最左前列开始并且不跳过索引中的列。最左边的不能丢失。
   3. .不在索引列上做任何操作（计算、函数、（自动or手动）类型转换），会导致索引失效而转向全表扫描
   4. 存储引擎不能使用索引中范围条件右边的列
   5. .尽量使用覆盖索引（只访问索引的查询（索引列和查询列一致））
   6. mysql在使用不等于（！=或者<>）的时候无法使用索引会导致全表扫描
   7. is null,is not null 也无法使用索引
   8. like以通配符开头（'$abc...'）mysql索引失效会变成全表扫描操作
   9. 字符串不加单引号索引失效
   10. 少用or,用它连接时会索引失效
   11. 小节：
       1. 定值、范围还是排序，一般order by是给个范围
       2. group by 基本上都需要进行排序，会有临时表产生
       3. 对于单键索引，尽量选择针对当前query过滤性更好的索引
       4. 在选择组合索引的时候，当前Query中过滤性最好的字段在索引字段顺序中，位置越靠左越好。
       5. 在选择组合索引的时候，尽量选择可以能包含当前query中的where子句中更多字段的索引
       6. 尽可能通过分析统计信息和调整query的写法来达到选择合适索引的目的

## java8新特性

## jvm

### GC垃圾回收的基本原理：

将内存中不再被使用的对象进行回收，GC中用于回收的方法称为收集器，由于GC会小号一些资源，对java中对象的生命周期进行分析后，按照新生代、旧生代的方式开进行收集，尽可能减小资源损耗

### 垃圾回收算法

标记清除算法

复制算法

标记整理算法

分代收集算法

### 垃圾收集机制

引用计数法

可达分析算法