# 1.Java 基础

## 1.1基础语法

1. ConcurrentHashMap key为什么不能null, 因为key或value为null,直接抛异常。

HashMap key值可以为null

HashTable key值不可以为null

2. **ArrayList：**

Jdk1.7:底层为数组，定义时就有长度为10的Object 类型数组，扩容时大小为数组的1.5倍。

Jdk1.8:底层为数组,定义时底层数组长度为0，使用add()添加元素时，数组才扩容到10，以后再扩容时大小为数组的1.5倍。

ArrayList 和Vector的区别：1.底层都是数组的扩容，但Vector 扩容为原来两倍。2.Vector 是线程安全的。

3. arrayList()转为数组：

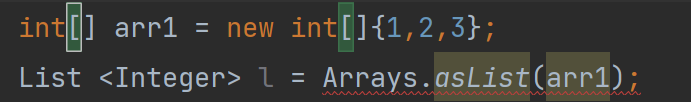
答：两种方法，一是Object[] strArray = strList.toArray();得到一个没啥用的Object[]数组。二是String[] strArray = strList.toArray(new String[strList.size()]);这是适合T[]类型，对于整形数组，会报错。这里同样提供两种方法：一是使用流式处理，int [] arr = list.stream().mapToInt(Integer::valueOf).toArray();二是使用for循环依次加入。

4.如何把int[] 数组转为list：

答：使用for循环将元素逐个添加进list。

能否用Arrays.asList():

答：Arrays.asList()语法如下：public static <T> List<T> asList(T…a),这是一个泛型方法，参数为可变参数，类型为T.我们可以用它来转换一个字符串数组，但是不能转换基本类型的数组，如下是转换int[] 数组时所报的错。



no instance(s) of type variable(s) exist so that int[] conforms to Integer inference variable T has incompatible bounds: equality constraints: Integer lower bounds: int[]

5. 数组和集合的对比：

数组缺点：长度一旦指定，就不可更改。

增删元素效率低。

数组中实际元素的数量无法获取。

有序，可重复。对于无序的，不可重复的 ，数组不能满足要求。

集合：只能存引用数据类型，

能存两种或两种以上不同类型的数据

集合的遍历：

1.for循环

2.增强for

3.iterator

6. transient：让某些被修饰的成员属性变量不被序列化。

7. Class 使用public 修饰，则此类与类文件名必须保持一致，否则编译会报错；一个.java 中之只能有一个类被public修饰，其它类不能再被public修饰，在编译后会生成对应的.class 文件。

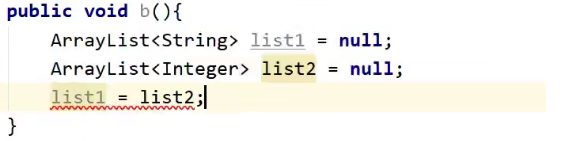
8.泛型：

（1）可以指定多个泛型

（2）泛型类的构造参数不能添加泛型：



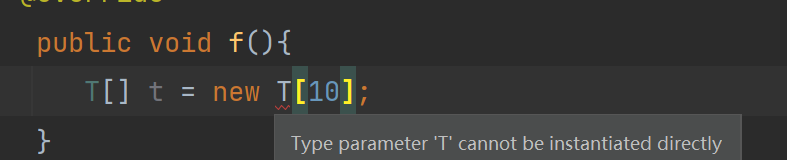
（3）不同的泛型的引用类型不能相互赋值：



（4）如果泛型不指定，泛型就会被擦除，反应对应的类型为Object类型。

（5）泛型类中的静态方法不能使用泛型类的泛型。

（6）不能直接使用T[]创建数组：



9.泛型方法：

1.语法：修饰符<T,E,…>返回值类型 方法名(形参列表){}

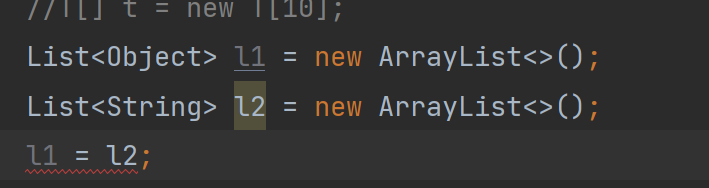
<T,E,…>为泛型列表，泛型列表中有的泛型，才可以在泛型方法中使用。

2.泛型类的泛型是在实例化的时候确定的，而泛型方法的泛型是在调用的时候确定的，所以

i.泛型方法中的泛型类型不受所在类的泛型约束（独立于泛型类而产生变化）

ii.泛型方法可以由static修饰，而普通成员方法不能。（static必须放在泛型列表之前）

3.A和B是继承关系，G<A>和G<B>是并行关系，如何解释？



10. 字符数组转为字符串：

String s = new String(chs);或者 String s = String.valueOf(chs);字符数组.toString()返回的并不是字符串。

字符串转为数值：

int value = Integer.valueOf(str);

或者 int value = Integer.parseInt(str);

或者 int value = new Integer(str);

11. 随机数生成：Random r = new Random(int seed);

r.nextInt(n);//生成[0,n)之间的随机数

如何生成[0，n）范围内的不重复的随机整机n个，由于r.nextInt(n)生成的时候会产生重复数字，所以需要使用set过滤重复的数，但是这样也有个问题，越是到后面，重复的概率越高，所以可以生成前一半随机数，然后加上n/2就是后一半随机数了。

12. HashSet的疑问？

1.数组长度是多少？

2.数组类型是什么？

3.hashCode,equals 方法真的调用了吗?验证

4.底层表达式是什么？

5.同一个位置的数据 向前放 还是向后放？

6.放入数组中的数据，是直接放的吗？是否封装为了对象？

13. lambda表达式：

（1）无参，无返回值



（2）lambda需要一个参数，但是没有返回值。

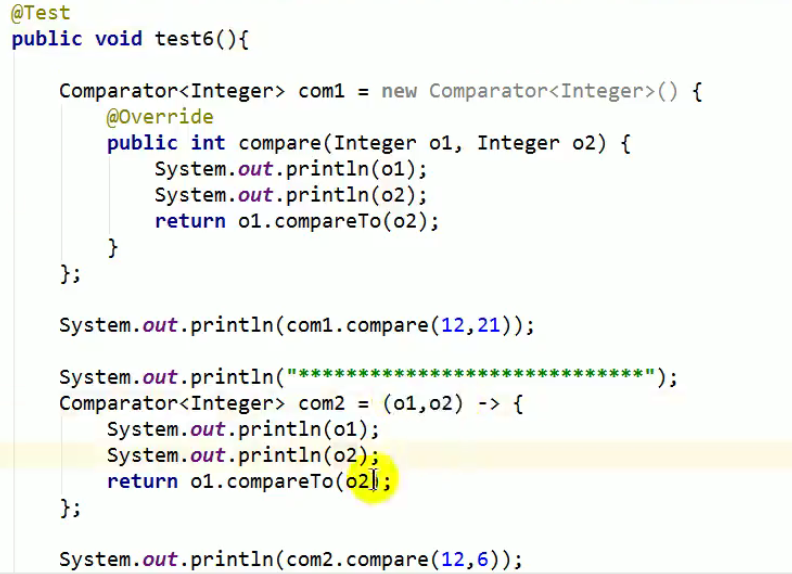


（3）数据类型可以省略，可由编译器推断。

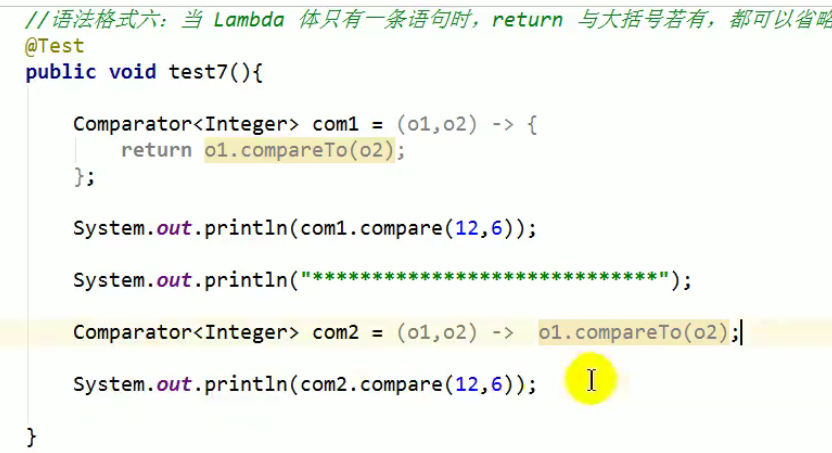


（4）lambda 若只需要一个参数时，参数小括号可以省略

（5）lambda需要两个或以上的参数时 ，多条执行语句且有返回值



（6）当lambda体只有一条语句时，return 与大括号若有，都可以省略



总结：

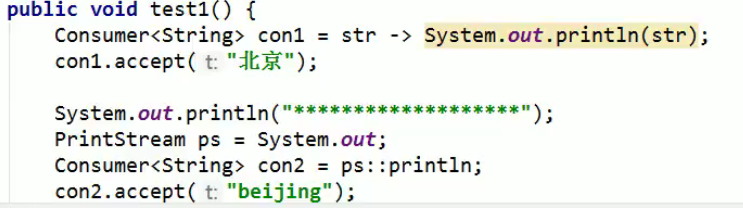
->左边：lambda形参列表的类型是可以省略（类型推断）；如果lambda表达式形参只有一个参数，小括号可以省略，其它情况都不能省略

->右边：lambda体应该使用{}包裹，如果lambda体只有一条执行语句，可以省略{}和return 关键字(如果是return语句).

函数式接口：接口只有一个抽象方法。

方法引用：方法引用是lambda表达式的深层次表达。当要传递给lambda体的操作已经有了实现的方法，可以使用方法引用！

方法引用的要求：要求接口中的抽象方法的形参列表和返回值类型与方法引用的方法的形参列表和返回值类型相同！



14. Stream API:

Stream 与Collection 的区别：Stream 关注的是对数据的运算，与cpu打交道

集合关注的是数据的存储，与内存打交道

Stream 自己不会存储元素

Stream不会改变源对象。相反，他们会返回一个持有结果的新Stream。

Stream操作是延迟执行的。这意味着他们会等到需要结果的时候才执行。

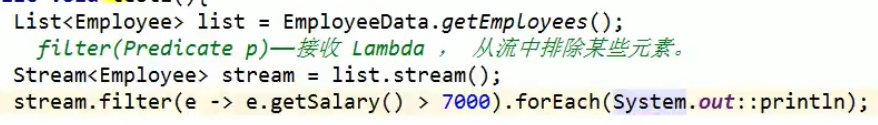
15. Stream执行流程

(1).Stream 的实例化 有四种实例化方式：通过集合;通过数组；通过Stream的of()方法；通过无限流

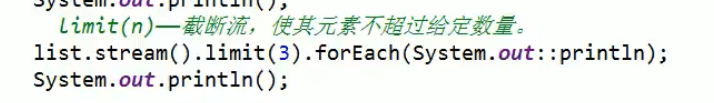
(2).一系列中间操作

筛选与切片

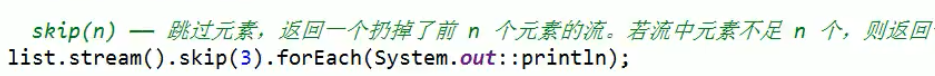
Filter(Predicate p) 从流中排除某些元素



Limit(n) 截断流



Skip(n) 跳过前几个数据



Distinct()

映射

Map(Function f) 接收一个函数作为参数，将元素转换成其他形式或者提取信息，该函数会被应用到每个元素上，并将其映射成一个新的元素。

flatMap(Function f)接收一个函数作为参数，将流中的每个值都换成另一个流，然后把所有流合成一个流。

排序

Sorted()

Sorted(comparator com) --定制排序

(3).终止操作

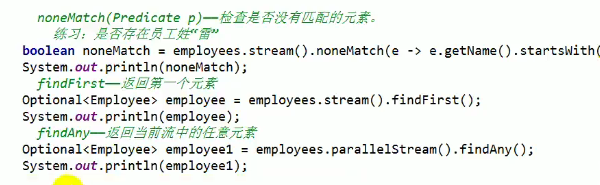
匹配与查找

allMatch(Predicate p)

anyMatch(Predicate p)

findFirst

FindAny

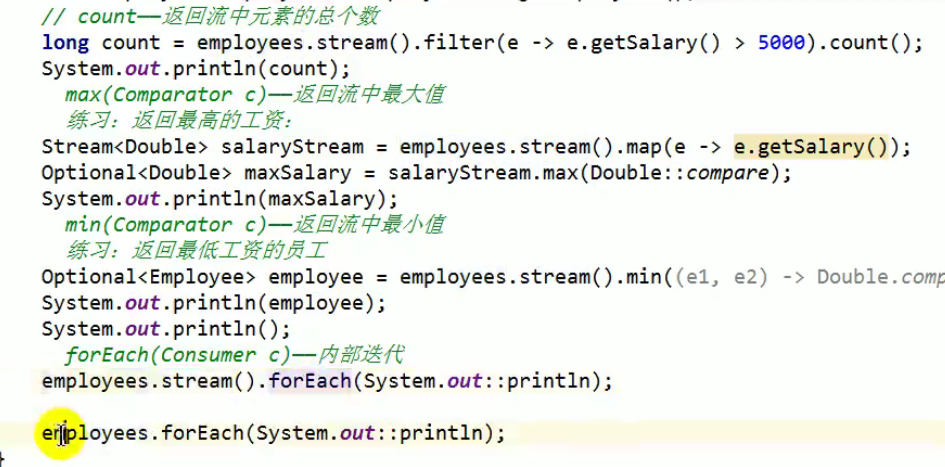


count

Max()

Min()

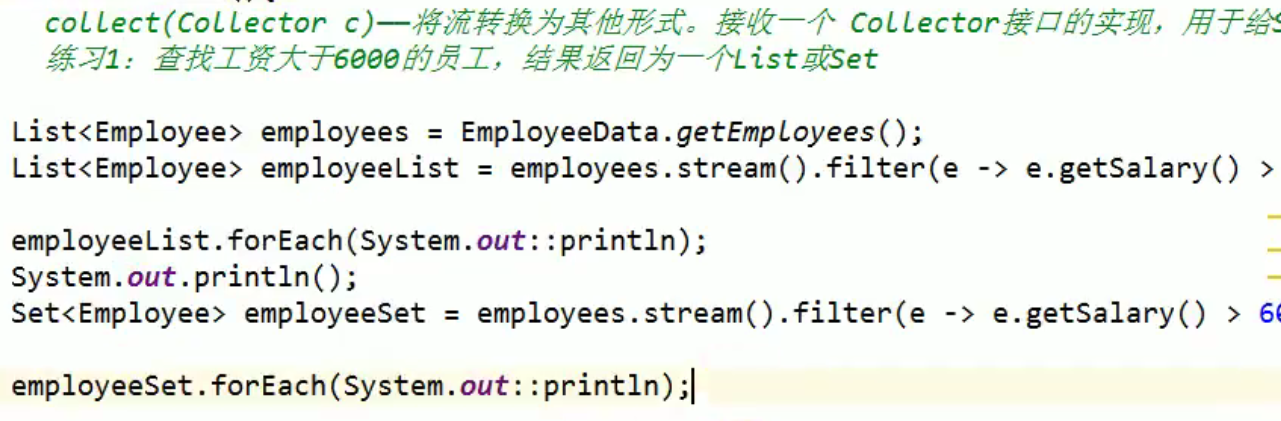
forEach()



归约

搜集

Collect 将流转换为其他形式。接收一个Collector 接口的实现，用于给Stream中的元素做汇总的方法。



16. Optional 类



17.接口中必须都是抽象方法吗？从java8开始，接口中就有默认的实现方法了和静态的实现方法。

18. Java 中的Math.log(n)函数是以自然数e为底。若想以其它为底，需要使用换底公式。Math.log(n)/Math.log(m) =

19. Java 打印两位小数的方法：

//方法一：最简便的方法，调用DecimalFormat类

DecimalFormat df = new DecimalFormat(".00");

System.out.println(df.format(d));

//方法二：直接通过String类的format函数实现

System.out.println(String.format("%.2f", d));

20. 数据是以补码的形式存在。-1 => 1111 1111 1111 1111 1111 1111 1111 1111;-1%16==15??答案是-1;-1>>4 == 0000 0000 1111 1111 1111 1111 1111 1111???答案是-1即-1>>4=-1;因为负数右移补的是1，想要前面补0，使用无符号右移>>>来实现。一个字符加上数字返回的是什么类型？答案是整形。’a’+1可以用字符接收吗？答案是不行。Java中怎么实现字符+数字还是字符呢？char = (char)(‘0’+num);

21. Math.abs(a)；如果a==Integer.MIN\_VALUE;返回的a的值没有发生变化。

10^9(10的9次方) 在java中表示法:1e9

22. 数在计算机中的是以补码表示的，~取反操作符，对整体取反，包括符号位。为什么正数的补码是自己而负数需要先取反再加1?这是为了简洁硬件设计。使用原码的进行加减运算，就需要设计专门的减法器。而使用补码则只需要设计加法器即可。

23. Java中的栈和队列都是用的双端队列 Deque<Integer> dq = new ArrayDeque();

在当作栈使用时，push入栈，pop出栈。

在当作队列使用时，offer入队，poll出队。

不要混用。

24. String的split()会产生空串，比如“1aa2”；用a切分时就会产生空串。

25. [String](https://so.csdn.net/so/search?q=String&spm=1001.2101.3001.7020)[] arr = split[0].split("+");//( [ { / ^ - $ ¦ } ] ) ? \* + .是转义字符 需要加\\

//java.util.regex.PatternSyntaxException: Dangling meta character '+' near index 0

26. 生成Integer的几种方式

Integer i = 100；//等价于Integer I = Integer.valueOf(100);返回缓存对象的地址

Integer i = 128;超出缓存阈值，就等价于Integer i = Integer.valueOf(128)；

Integer i = new Integer(100);

27. Deque 代替 Stack有两点原因：stack继承至Vector，使得可以在任意位置插入元素，破坏了栈的设计理念，二是vector是线程安全的，这导致单线效率低。

28. 数据库的四大特性：ACID。

一、原子性（atomicity)

一个事务要么全部提交成功，要么全部失败回滚，不能只执行其中的一部分操作，这就是事务的原子性

二、一致性（consistency)

事务的执行不能破坏数据库数据的完整性和一致性，一个事务在执行之前和执行之后，数据库都必须处于一致性状态。

三、隔离性（isolation）

事务的隔离性是指在并发环境中，并发的事务时相互隔离的，一个事务的执行不能不被其他事务干扰。

四.持久性

一旦事务提交，那么它对数据库中的对应数据的状态的变更就会永久保存到数据库中。

29.

# 2.java集合

# 3.java并发

## 3.1 并发基础

## 3.2 并发关键字

## 3.3 JUC全局观

## 3.4 JUC原子类

## 3.5 JUC锁

## 3.6 JUC集合类

## 3.7 JUC线程池

## 3.8 JUC工具类

# 4.java IO

## 4.1基础IO

## 4.2 5种IO模型

## 4.3零拷贝

# 5.JVM和调优

## 5.1类加载机制

## 5.2内存结构

## 5.3 GC垃圾回收

## 5.4 问题排查

# 6.java 新版本特性

# 7.数据结构和算法

# 8.数据库

## 8.1 原理和SQL

## 8.2 MYSQL

1. Mysql的三种join:

(1)inner join，内连接,显示两个表中有联系的所有数据;

(2)left join，左链接,以左表为参照,显示所有数据,右表中没有则以null显示;

(3)right join，右链接,以右表为参照显示数据，,左表中没有则以null显示;

2.mysql优化器

MySQL 优化器使用基于成本的优化方式（Cost-based Optimization），以 SQL 语句作为输入，利用内置的成本模型和数据字典信息以及存储引擎的统计信息决定使用哪些步骤实现查询语句，也就是查询计划。  
3.使用索引查询的过程：

B+树中查找的过程。

## 8.3 Redis

1.基本操作

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | String | list | Set（hash） | Hash(ziplist, hashtable) | Zset(hash, 跳表) | Bitmaps | hyperloglog | Geospatial |
| 增 |  |  |  |  | Zadd<key>  <score1><value1> |  |  |  |
| 删 |  |  |  |  | Zrem<key><value> |  |  |  |
| 改 |  |  |  |  | Zincrby<key>  <score><value> |  |  |  |
| 查 |  |  |  |  | Zrange<key>  <start><end>[withscores]  Zrangebyscore<key>  <score\_start><score\_end>  Zcount<key><min><max>返回分数区间内的元素个数  Zrank<key><value>返回该值在集合中的排名，从0开始 |  |  |  |

发布和订阅：消息通信模式。Redis支持任意数量的通信模式。

HyperLogLog:用于统计基数（不重复元素的个数），以往使用bitmap或set统计时，对大数据统计是耗费空间的，但HyperLogLog可以在只使用12KB的空间下，损失一定精度的情况下完成大数据的基数统计。

2.HyperLogLog的数学原理?

3.

## 8.4 MongoDB

## 8.5 ElasticSearch

# 9.开发基础

# 10.开发框架和中间件

# 11.开发工具

# 12.架构

# 13.分布式

# 14.微服务

# 15.DevOps

# 16.其它

## 16.1设计模式

## 16.2 开源协议

## 16.3 软件理论

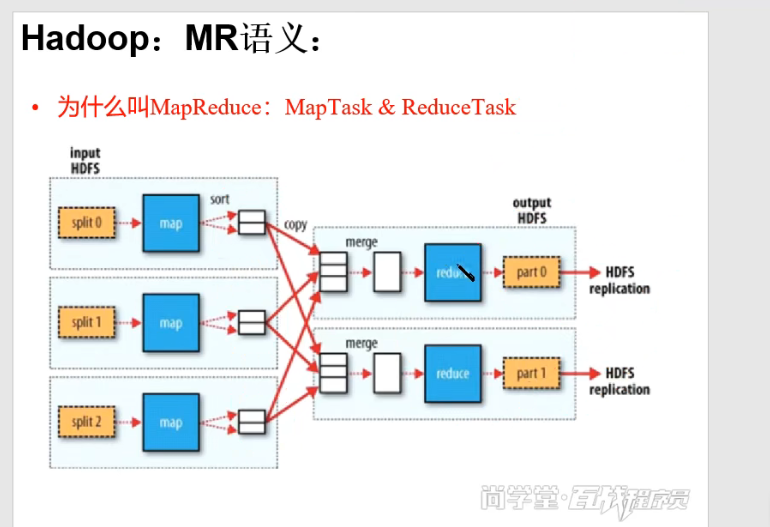
## 16.4 软件成熟度模型

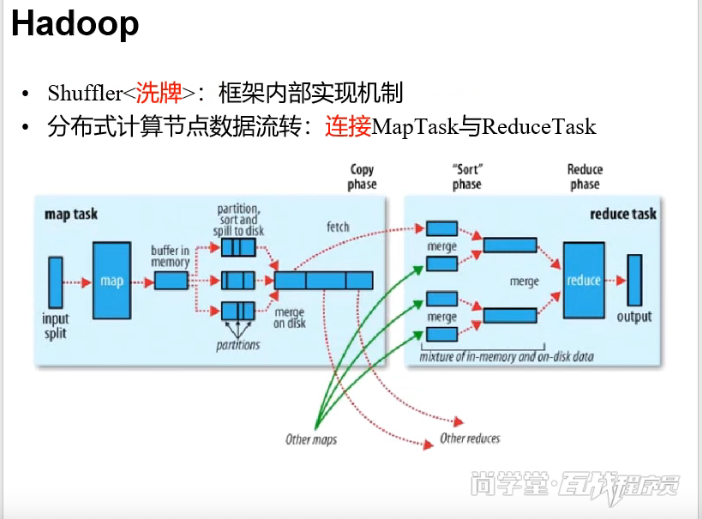
## 16.5 等级保护

## 16.6 ISO27001

其他：

Linux 文件权限：文件所有者（Owner）、用户组（Group）、其它用户（Other Users）。  
原语：相同的数据占一组，一组数据调用一次reduce方法。





# 1.Java 基础

## 1.1基础语法

1. ConcurrentHashMap key为什么不能null, 因为key或value为null,直接抛异常。

HashMap key值可以为null

HashTable key值不可以为null

2. **ArrayList：**

Jdk1.7:底层为数组，定义时就有长度为10的Object 类型数组，扩容时大小为数组的1.5倍。

Jdk1.8:底层为数组,定义时底层数组长度为0，使用add()添加元素时，数组才扩容到10，以后再扩容时大小为数组的1.5倍。

ArrayList 和Vector的区别：1.底层都是数组的扩容，但Vector 扩容为原来两倍。2.Vector 是线程安全的。

3. arrayList()转为数组：

答：两种方法，一是Object[] strArray = strList.toArray();得到一个没啥用的Object[]数组。二是String[] strArray = strList.toArray(new String[strList.size()]);这是适合T[]类型，对于整形数组，会报错。这里同样提供两种方法：一是使用流式处理，int [] arr = list.stream().mapToInt(Integer::valueOf).toArray();二是使用for循环依次加入。

4.

# 2.java集合

# 3.java并发

## 3.1 并发基础

## 3.2 并发关键字

## 3.3 JUC全局观

## 3.4 JUC原子类

## 3.5 JUC锁

## 3.6 JUC集合类

## 3.7 JUC线程池

## 3.8 JUC工具类

# 4.java IO

## 4.1基础IO

## 4.2 5种IO模型

## 4.3零拷贝

# 5.JVM和调优

## 5.1类加载机制

## 5.2内存结构

## 5.3 GC垃圾回收

## 5.4 问题排查

# 6.java 新版本特性

# 7.数据结构和算法

# 8.数据库

## 8.1 原理和SQL

## 8.2 MYSQL

## 8.3 Redis

## 8.4 MongoDB

## 8.5 ElasticSearch

## 8.6 分库分表

# 来源：https://mp.weixin.qq.com/s/oA2vbWnD1b3yBIrW65Nv3Q

分库场景：1.磁盘不够用了2.单个数据库的并发连接数量有限

分表场景：单表超过1000万行（一行记录为1k,高度为3的B+树最多存放两千一百万行数据，考虑到一行数据平均大小不到0.5K，高度为的B+树最多存放4000-5000万行）。

阿里java开发手册提出，单表数据超过500万行或超过2GB时，考虑分库分表操作。

怎么分？

分库：一般分3-10个库。可以根据业务场景，分成，用户库，订单库，商品库。

1. 按范围分 缺点：热点问题数据倾斜
2. 按hash值取模分 范围查询问题，二次扩容问题（动态伸缩问题）。
3. 一致性hash分 一致性hash能够尽可能小地改变已经存在的服务请求与处理请求请求服务器之间的映射关系。

如何避免热点数据倾斜的问题？

范围+hash取模的方式。

# 9.开发基础

# 10.开发框架和中间件

# 11.开发工具

# 12.架构

# 13.分布式

# 14.微服务

# 15.DevOps

# 16.其它

## 16.1设计模式

## 16.2 开源协议

## 16.3 软件理论

## 16.4 软件成熟度模型

## 16.5 等级保护

## 16.6 ISO27001

ACID:https://www.bilibili.com/video/BV1eb41137kX?from=search&seid=10122748078185040225 51：36

多线程调试

集合的转换?

泛型通配符：List<?> list = new ArrayList();

1.遍历，对于传进来的list，其元素类型为Object 类型；

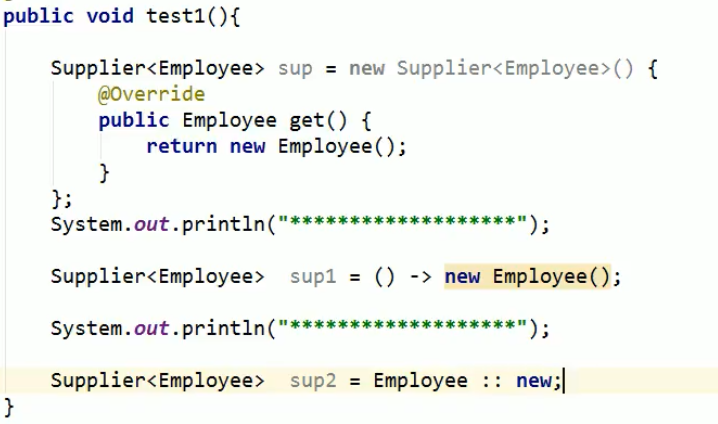
2.插入，除了null,无论什么类型都无法插入

3.读取，list.get(index)获取的元素是Object类型的。

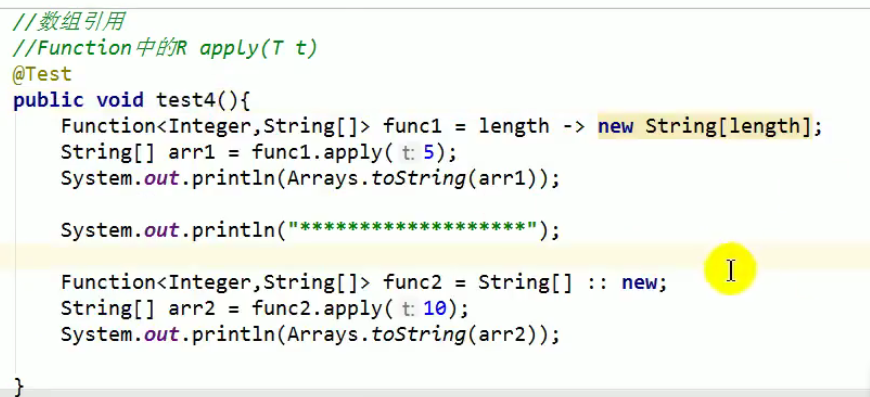
泛型受限：



构造器引用：和方法引用类似



数组引用:



//:在字符串尾部split(":")是不会切一个新的字符串出来的，但是在首部却可以

ACID靠什么保证？

A 原子性由undo log日志保证，它记录了需要回滚的日志信息，事务回滚时撤销已经执行成功的sql

C一致性由其他三大特性保证、程序代码要保证业务上的一致性

I 隔离性由MVCC来保证

D持久性由内存+redo log来保证，mysql修改数据同时在内存和redo log记录这次操作，宕机的时候可以从redo log恢复。

Java 生成对象的几种方式：直接new 反射(Class.forname(“xxx”).newInstance()) Constructor的newInstance clone deserialization

Redis 单线程为什么还很快：1.基于内存2.单线程没有线程切换的开销3.基于IO多路复用机制提升redis的I/O利用率4.高效的数据存储结构。

进程的五大状态：创建，就绪，执行，阻塞，终止。

HashMap 的长度为什么是2的幂次方？为了加快哈希计算以及减少哈希冲突。

双重检查锁

private volatile static Singleton singleton;

private Singleton (){}

public static Singleton getSingleton() {

if (singleton == null) {

synchronized (Singleton.class) {

if (singleton == null) {

singleton = new Singleton();

}

}

}

return singleton;

}

为什么必须加volatile?因为new Singleton();可分为三个步骤：

//1：分配对象的内存空间

memory = allocate();

//2：初始化对象

ctorInstance(memory);

//3：设置instance指向刚分配的内存地址

instance = memory;

在指令重排之后，步骤2，3会互换。此时，如果线程t2进入到第一个if,判断后不为空，直接返回未完成初始化的对象引用，造成后续错误。

TCP粘包和拆包：TCP报文段是面向字节流的，没有消息边界，于是就会将太小的报文段粘包在一起一起发送，对于过大的报文段就需要拆包发送。因为没有消息边界，粘包拆包后就会不知道怎么拆分和重组。解决办法：固定消息大小，规定消息边界，自定义协议。

https = http + TLS

http是明文传输，所以有可能被截获并篡改消息内容。

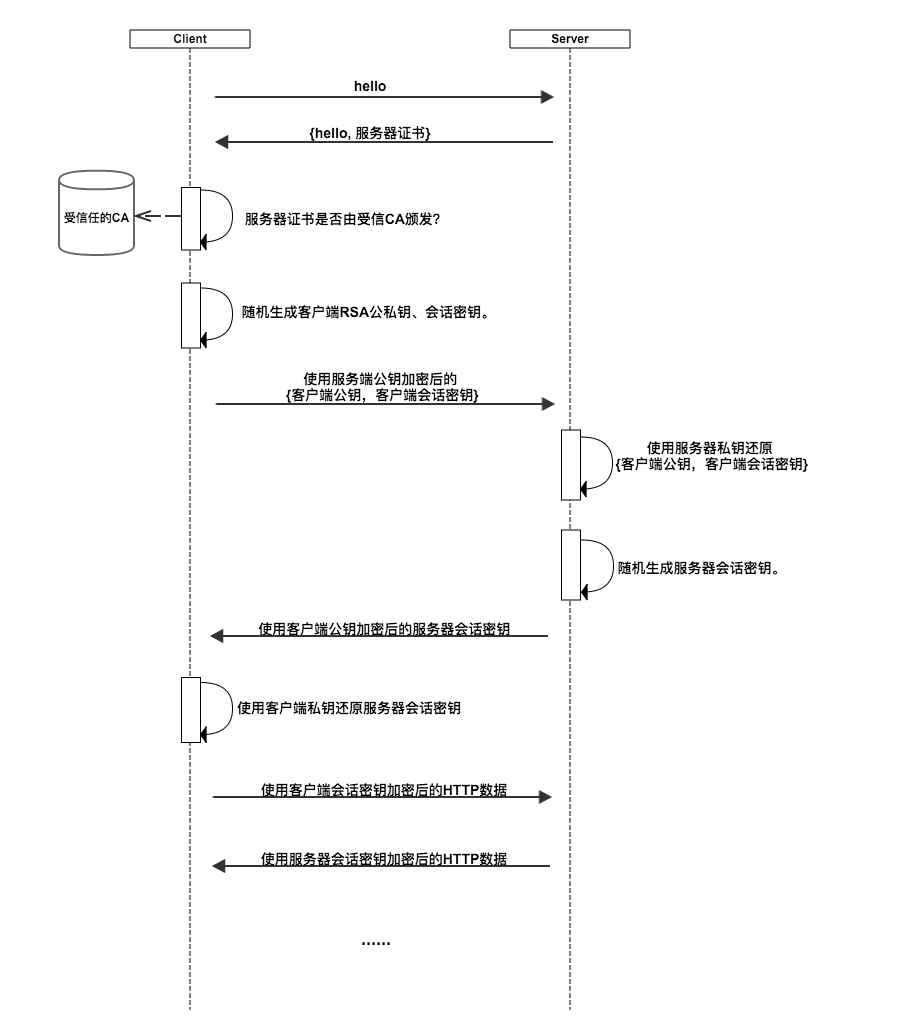
Https做了两点：1.CA(certificate authority)授权认证2. 保证所传输数据的私密性和完整性。

可以看到工作流程，基本分为三个阶段：

认证服务器。

协商会话密钥。客户端在认证完服务器，获得服务器的公钥之后，利用该公钥与服务器进行加密通信，协商出两个会话密钥，分别是用于加密客户端往服务端发送数据的客户端会话密钥，用于加密服务端往客户端发送数据的服务端会话密钥。在已有服务器公钥，可以加密通讯的前提下，还要协商两个对称密钥的原因，是因为非对称加密相对复杂度更高，在数据传输过程中，使用对称加密，可以节省计算资源。另外，会话密钥是随机生成，每次协商都会有不一样的结果，所以安全性也比较高。

加密通讯。



静态方法和非静态方法的区别

1、静态方法是使用static关键字修饰的方法，属于类的，不属于对象；非静态方法是不使用static关键字修饰的普通方法，属于对象，不属于类。2、静态方法可以直接调用，类名调用和对象调用；非静态方法只能通过对象调用。3、生命周期不同。

死锁产生的必要条件：互斥，不可剥夺，请求保持，循环等待。

写个死锁：

Https http2.0 quic的区别？

HTTP2具有更好的效率和资源利用率，尤其适用于页面比较重，有大量资源加载的场景。http2具有的新特性：二进制分帧，多路复用，服务器推送，头部压缩，ALPN 应用协议协商。（HTTPS 握手的时候，客户端会首先告诉服务端自己支持的协议，由服务端选择客户端服务端都支持的协议。如果服务端Nginx开启了HTTP2支持，服务端会选择HTTP2协议，否则，服务端就会选择HTTP1.1协议来通讯。）

为什么懒汉模式会线程不安全？

抽象类和接口的区别？

1. 关键字不同
2. 抽象类里可以有抽象方法，也可以有非抽象方法，接口中不能有实现方法（java1.8中有默认的实现）
3. 接口中除了static，final变量，不能有其他变量。
4. 一个类可以实现多个接口，但是只能继承一个类。
5. 接口中方法的默认修饰符是public,但是抽象方法可以是public，protected, default修饰符。
6. 抽象是对类的抽象，是模板设计。接口是对行为的抽象，是对行为的规范。

Private, public protected,以及default作用范围。



类加载：加载，连接（验证， 准备， 解析）和初始化三个阶段。

类的生命周期：加载， 验证， 准备， 解析， 初始化， 使用，卸载。

加载：是读取Class文件，将其转化为某种静态数据结构存储在方法区内，并在堆中生成一个便于用户调用的java.lang.Class类型的对象的过程。包含文件格式验证。

验证：元数据，字节码验证（保证不会对jvm产生危害）。

准备：为静态变量赋0值。

解析：可发生在初始化前和初始化后。将符号引用替换为直接引用。符号引用：加入Class A中有指向Class B的引用，但是在解析时，Class B未必被加载了，所以Class A中有指向Class B的引用就是一串字符串，这串字符串就是符号引用，然后触发Class B的加载，在Class B被加载后，就会将符号引用替换为Class B的内存地址，这就是将符号引用替换为直接引用。以上称为静态解析。当B是一个抽象类或者接口是，并不清楚具体的实现，那符号引用一直到发生具体的调用才会被替换为具体的直接应用。

初始化：类层面上，对成员变量显示赋值的操作。

双亲委派机制：类加载器有Bootstrap ClassLoader, Extension ClassLoader（加载lib/ext目录下的类）, Application ClassLoader（classpath路经中的类）, User ClassLoader（加载任何.class文件）。双亲委派机制说的是能让父亲加载器加载的就应该让父亲加载器加载，只有父亲加载器不能加载时才由孩子加载器加载。越核心的类越是被上层类加载器加载，而限定名的类一旦被加载过了，被动情况下，就不会再加载相同限定名的类。这样就能有效避免混乱。

为什么非要优先让父亲加载器加载？避免相同的类被不同的加载器重复加载。

能否让Extension ClassLoader中的类加载Application ClassLoader的类？可以，但工程上是分层的，应该各司其职。

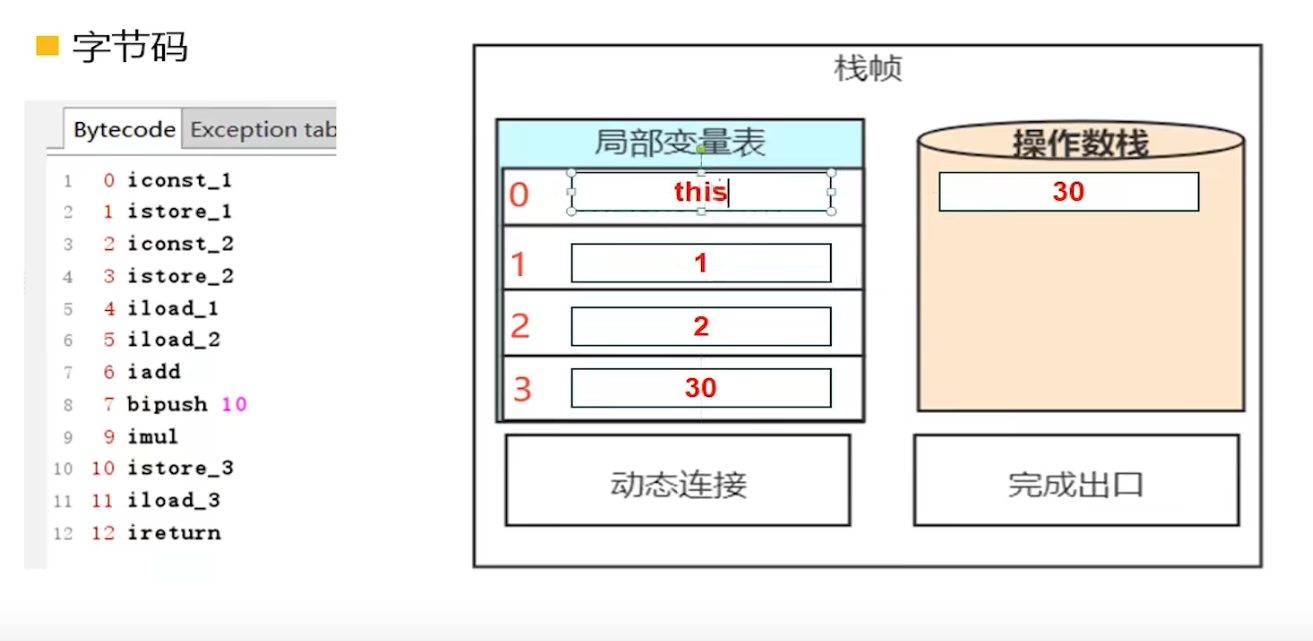
类加载器加载流程？加载，验证，准备，解析， 初始化， 卸载。自定义ClassLoader也只能读取类文件，其它的都有虚拟机默认完成。

三次破坏双亲委派机制分别是什么？ 在java1.2第一次引入双亲委派机制时，已经出现很多自定义的类加载方式，为了向下兼容，java选择接收，也给了补救措施，就是后来开发者可以重写findclass方法。第二次时java数据库的驱动，使用的时applicationClass Loader,没有遵照双亲委派机制。第三次时为了模块化设计，支持热部署，也会践踏双亲委派机制。

Java 内存结构

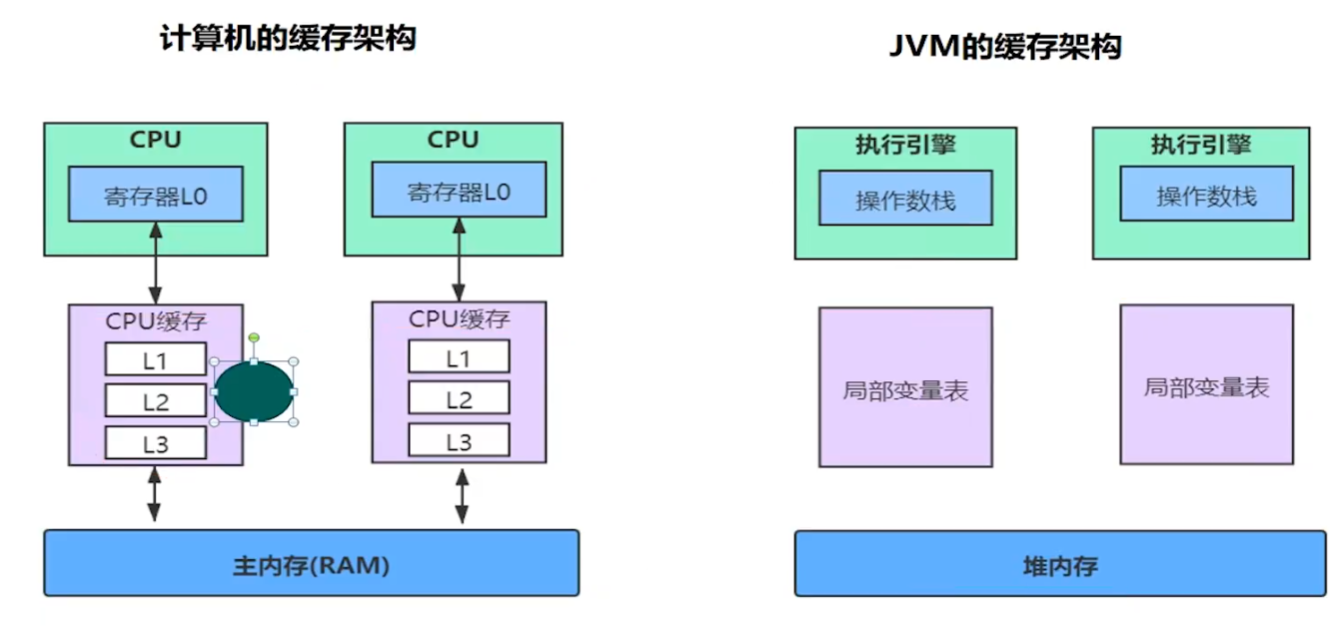
1. 方法区
2. 堆
3. Java 栈（）
4. 本地方法栈
5. 程序计数器

Java虚拟机栈存放栈帧，栈帧包含：局部表量表，操作数栈，动态链接，返回地址，以及其他附加信息。



为什么局部变量表里面0地址没有存放值？因为0地址存放的是this指针，存放的是当前实例的地址。注：字节码中的0下标是偏移量，不是行号。动态链接跟多态相关。

为什么会有操作数栈这个数据结构？不能直接在局部变量表里面取值吗？



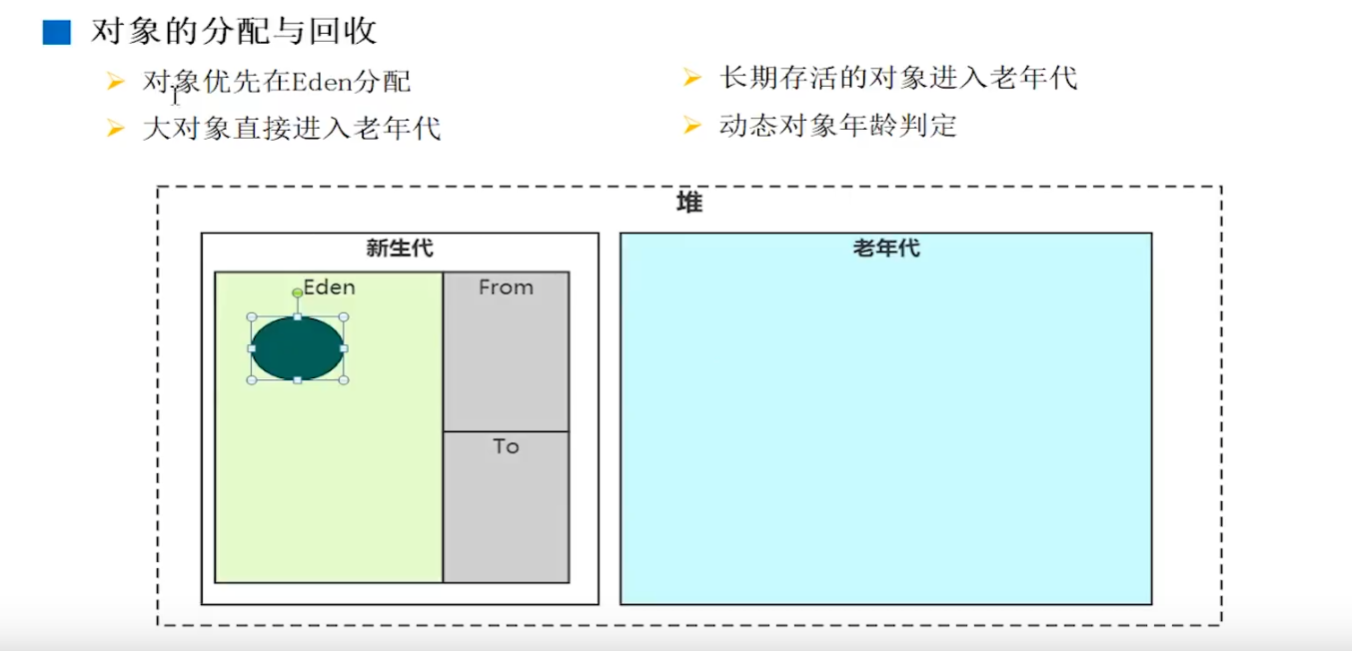
仿照计算机的缓存架构设计的java内存模型。

方法区：主要用于存储已被虚拟机加载的类信息（包括类的名称，方法信息，字段信息）， 常量（final, 字符串）， 静态变量， 静态方法， 成员方法，及时编译器编译后的代码等数据。

Jvm运行内存处理过程：

1. jvm申请内存
2. 初始化运行时数据区
3. 类加载
4. 执行方法
5. 创建对象

堆内存



长期存活的对象进入老年代：指的是如果垃圾回收的达到15次还没有回收该对象，则放入老年代。

如果大对象过大（跟参数设置有关，一般为10M）,则直接放在老年代。

动态对象年龄判定：如果有若干小对象，这些小对象之和达到了From区的50%,则直接放在老年代。

常量池和运行时常量池：

运行时常量池：1.编译期间产生的，主要是字节码中定义的静态信息，比如：由字节码产生的Class对象；有字节码生成的字面量

2.运行期间产生的，这部分比较灵活，虚拟机开发者可以将必要的数据放进去，比如：运行时会将一部分符号引用转换为直接引用，那么这些直接引用可以存储进来；常见的字符串常量池等等

常量池：

Java内存模型

Volatile如何保证可见性的：写valatile修饰的变量，主动写主存；读volatile变量，主动读主存。

元空间和永久代都是方法区的实现。Java1.8后元空间代替的永久代。

永久代的实现方法区的缺点：可能引起内存溢出；永久代本身的复杂设计并不是方法区需要的，并可能带来未知的异常。

面向对象的SOLID原则：

单一功能：应该有且只有一个原因引起类的改变。

开闭原则：类，模块和函数应该对扩展开放，对修改关闭。

里氏替换原则：只要有父类的地方，子类就可以出现，而且替换为子类也不会产生任何错误或异常。

接口隔离原则：客户端不应该依赖不需要的接口，类间的依赖关系应该建立在最小接口上。

缓存一致性协议：主要是为了解决多个cpu缓存之间的同步问题。可分为两类：窥探型和基于目录型。

程序计数器：记录程序运行地址。

创建线程的方式：

1. 继承Thread类创建线程类
2. 实现Runnable接口
3. 通过Callable和Future创建线程

采用这三者分别有什么优势劣势？

线程类只是实现Runnable接口或Callable接口，还可以继承其他类。在这种方式下，多个线程可以共享同一个target对象，所以非常适合多个相同线程处理同一份资源的情况，从而可以将cpu，代码和数据分开，形成清晰的模型，较好的体现了面向对象的思想。

劣势：编程相对复杂一点，如果要访问当前线程，则必须使用Thread.currentThread()方法。而继承Thread类，编写多线程相对简单，如果需要访问当前线程，使用this即可获得。但线程已经继承了Thread类，所以不能再继承其他父类。

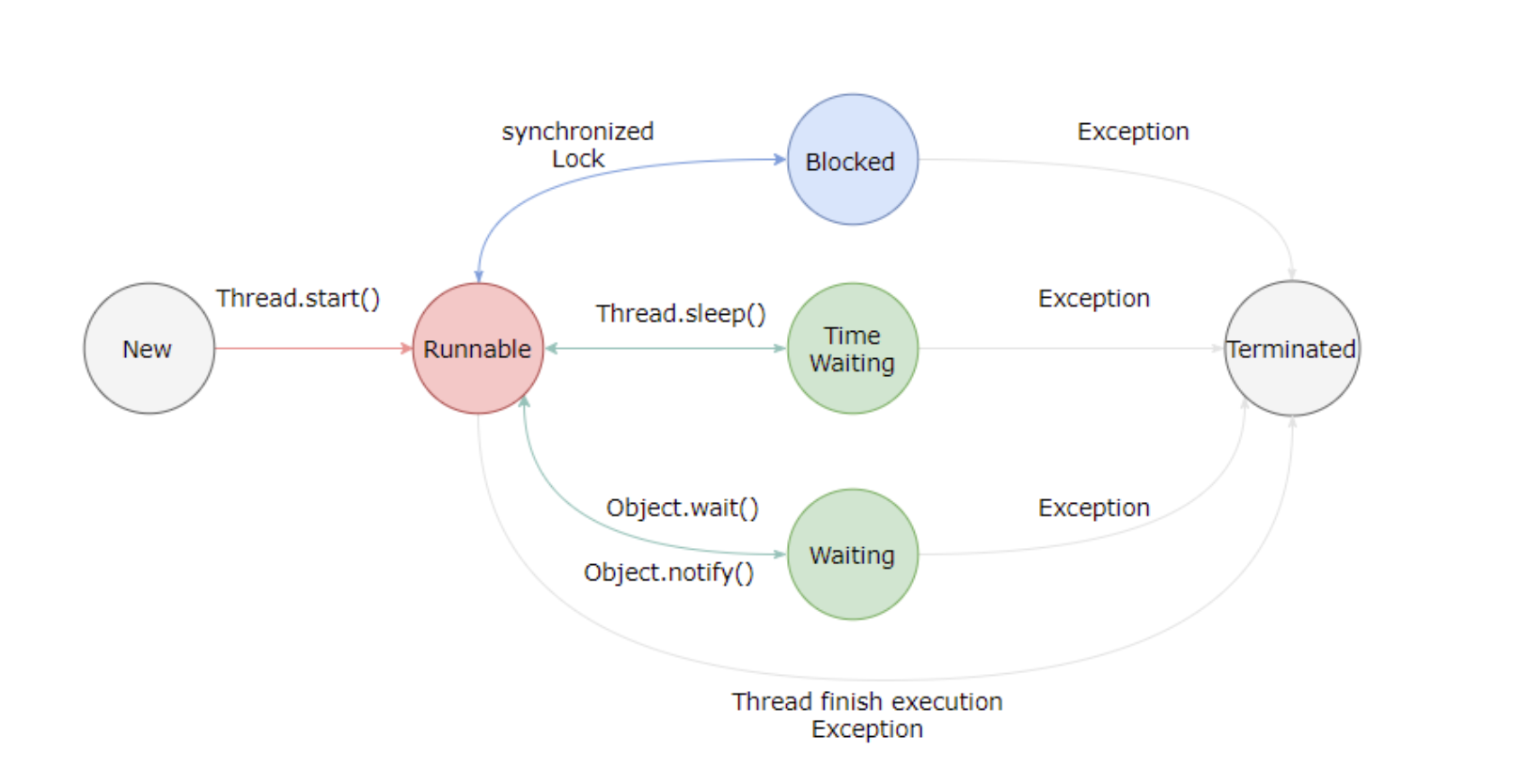
Runnable和Callable的区别

1. Callable规定（重写）的方法是call（）,Runnable规定（重写）的方法是run()。
2. Callable的任务执行后可返回值，而Runnable（）的任务是不能返回值的。
3. Call方法可以抛出异常，run方法不可以
4. 运行Callable任务可以拿到一个Future对象，表示异步计算的结果。它提供了检查计算是否完成的方法，以等待计算的完成，并检索计算的结果。通过Future对象可以了解任务的执行情况，可取消任务的执行，还可以获取执行结果。

进程间的通信：管道，命名管道，信号量，共享内存，消息队列

创建，就绪，执行，阻塞，终止。

线程:



MySQL

聚簇索引和非聚簇索引：

聚簇索引：将聚集索引表记录的排列顺序和索引的排列顺序一致。只要找到第一个索引值记录，其余就连续性的记录在物理也一样连续存放。聚集索引对应的缺点就是修改慢，因为为了保证表中记录的物理和索引顺序一致，在记录插入的时候，会对数据页重新排序。

非聚簇索引：非聚集索引制定了表中记录的逻辑顺序，但是记录的物理和索引不一定一致。

Limit优化：

1.子查询优化法 先找出第一条数据，然后大于等于这条数据的id就是要获取的数据

缺点：数据必须是连续的，可以说不能有where条件，where条件会筛选数据，导致数据失去连续性

2. 倒排表优化法

倒排表法类似建立索引，用一张表来维护页数，然后通过高效的连接得到数据

缺点：只适合数据数固定的情况，数据不能删除，维护页表困难

3.反向查找优化法

当偏移超过一半记录数的时候，先用排序，这样偏移就反转了

缺点：order by优化比较麻烦，要增加索引，索引影响数据的修改效率，并且要知道总记录数，偏移大于数据的一半

4.limit限制优化法

把limit偏移量限制低于某个数。超过这个数等于没数据，我记得alibaba的dba说过他们是这样做的

四种引用:

强引用：把一个对象赋给应用对象，那这个引用变量就是强引用。当一个对象被强引用变量引用时，同时处于可达状态，他是不可能被垃圾回收机制回收的。

软引用：软引用需要用soft reference类来实现。系统内存不足时，会被回收，当空间内存充足时不会被回收。通常用在内存敏感的程序。

弱引用：需要用weak reference 类来实现。只要垃圾回收机制一运行，必被回收。

虚引用：需要plantom reference类来实现。不能单独使用，必须和引用队列联合使用。作用是跟踪对象被垃圾回收的状态。

垃圾回收器的种类：Serial, parallel,CMS

Serial：单线程设计，只使用一个线程进行垃圾回收，会暂停所有用户线程。

Parallel:多个垃圾手机线程并行工作。用户线程是暂停的。适用科学计算场景。

CMS:并发垃圾回收器，用户线程和垃圾收集线程同时执行，不一定是并行，可能是交替执行。

线程池的底层为ThreadPoolExector:

七大参数int corepoolsize//常驻的核心线程数

Int maxmiumpoolsize //能够容纳的同时执行的最大线程数

Int keepalive//多余的空闲线程的存活时间

Int timeunit//keepalive的单位

BlockingQueue blockQueue<>();//候客区（提交但未被执行）

ThreadFactory //表示生成线程池的线程工厂，用于创建线程

Handler//拒绝策略

四种拒绝策略：AbortPolicy:直接抛出RejectedExecutionException

CallerRunsPolicy:将任务退还给调用者

DiscardOldestPolicy:丢弃等待最久的任务，然后把当前任务加入队列中尝试再次提交当前任务。

DiscardPolicy:直接丢弃，不予任何处理，也不抛任何异常。

Int maxmiumpoolsize //能够容纳的同时执行的最大线程数 这个值取决于是CPU密集型还是IO密集型。前者=cpu核心数\*2 + 1；后者= cpu核心数\* 2 /(1-阻塞系数（=0.9）)

虚拟内存：

轻量级锁，重锁，乐观锁，悲观锁 synchronize, reenterLock

Bean 的生命周期：

MySql:

Innodb 与 myisam的区别：

脏读：

不可重复读：

幻读：

读已提交

可重复读：能但不能完全解决幻读问题。特例：如果两次快照读之间存在当前读，幻读问题依然存。

Mysql:

优化

覆盖索引：一个索引包含了（或覆盖了）满足查询结果的数据就叫做覆盖索引。

慢查询优化：

索引失效：模型数空运最快 模：模糊查询 型：数据类型 数：函数 空：null, 运：运算 最：最左匹配原则 快：全表扫描更快

1. 联合索引不满足最左匹配原则
2. 使用了select \*
3. 索引列参与运算
4. 索引列参使用了函数
5. 类型隐式转换：参数类型与字段类型不匹配，导致类型发生了隐式转换，索引失效。特例：如果字段类型为int类型，而查询条件添加了单引号或双引号，mysql会将参数转为int类型，上述语句会走索引。
6. 使用OR操作。使用不当，也会导致索引失效。1.使用or关键字时，其中一个字段没有创建索引，会导致整个查询语句索引失效；or两边为“>”和“<”范围查询时，索引失效。
7. 两列做比较。
8. 使用<>不等于比较，
9. 查询条件使用不等进行比较时，需要慎重，普通索引会查询结果集占比较大时索引会失效。
10. 查询条件使用is null时正常走索引，使用is not null时，不走索引。
11. 查询条件使用not in时，如果是主键则走索引，如果是普通索引，则索引失效。查询条件使用not exists时，索引失效。等情况 。

当前读：insert update delete select … for update select … lock in share mode

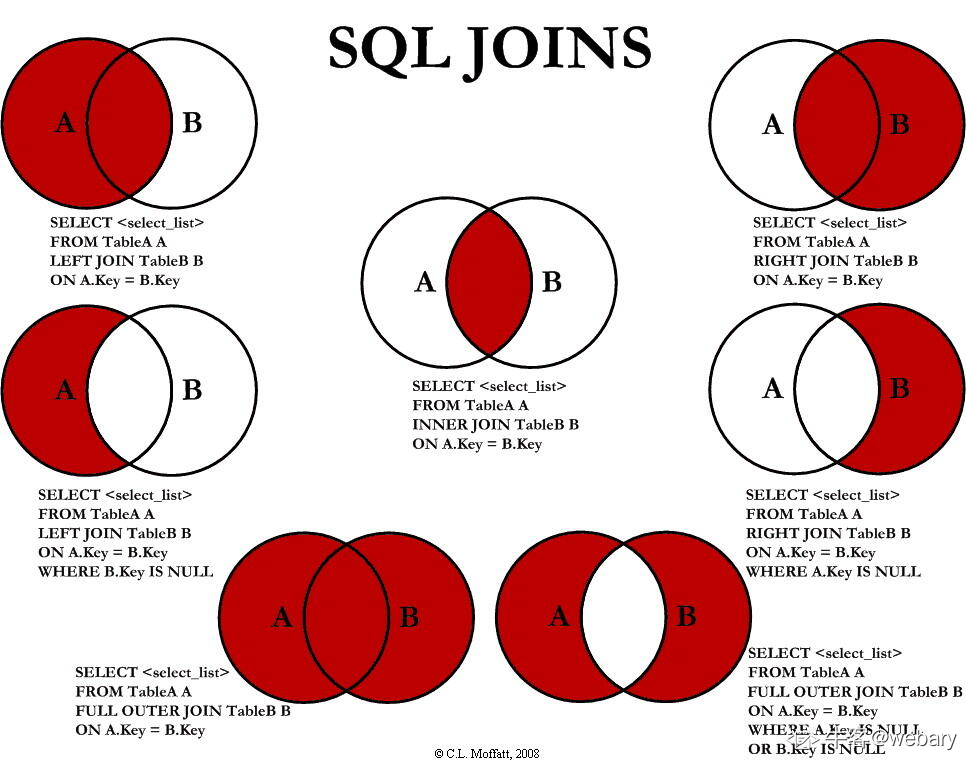
快照读：select



SQL 统计AAA出现的次数？ select length(ziduan)-length(replace(ziduan,'、','')) from biao

Where与having的区别：where用于过滤行，having用于过滤分组。如果过滤条件有索引，放在where子句中可以过滤索引，而放放在having子句中不能走索引。

Mysql的几种7种join:



切割、截取、删除、替换

select

-- 替换法 replace(string, '被替换部分','替换后的结果')

-- device\_id, replace(blog\_url,'http:/url/','') as user\_name

-- 截取法 substr(string, start\_point, length\*可选参数\*)

-- device\_id, substr(blog\_url,11,length(blog\_url)-10) as user\_nam

-- 删除法 trim('被删除字段' from 列名)

-- device\_id, trim('http:/url/' from blog\_url) as user\_name

-- 字段切割法 substring\_index(string, '切割标志', 位置数（负号：从后面开始）)

device\_id, substring\_index(blog\_url,'/',-1) as user\_name

MapReduce的切片过程。

Spring的IOC。

Session和cookie的区别?

Cookie是保存用户信息的一种最简单的方式，cookie中可以保存用户的一些信息，存放在浏览器的本地存储中，并随着每次请求发送到服务器。

用户首次认证并登陆服务后，服务返回给用户的HTTP中带着一个服务器生成的cookie信息，浏览器会自动保存到本地，带用户下次再访问该网站时会自动带上该cookie；

不同的网站会发送不同的cookie，cookie和域名绑定，并能指定path；

cookie有超时机制，如果超时需要重新认证再让服务方重新分配一个cookie；

cookie可以指定secure=true属性，指定后只有https才能使用该cookie；

cookie还可以指定httponly=true属性，可以防止js拿到cookie，只有服务器可以操作cookie

不足：

最大的原因是因为它存储在浏览器端，一些别有用心的人能够通过浏览器截获cookie（脚本、利用工具抓取等）；

Cookie在网络中很容易被劫持，这个问题需要用https解决；

Cookie的失效时间是不受服务端控制的，用户可以修改；

同一站点能设置的cookie有限。

Session是比cookie安全高的另一种会话保持技术，Session把信息存储的服务器端，在浏览器端只需要存储一个session id，服务端通过用户请求中带的session id自动获取用户信息。

session id虽然存储在cookie中，但所有用户信息都存放在服务端，通常公司对自己的用户信息都有更强的安全保护；

服务端可以存储用户大量信息，用户可以用一次登录就可以免密登录同一个公司的不同服务和网站；

session id通常会使用HMAC生成，并结合cookie使用，较为灵活，兼容性较好，防止篡改。

不足：

1.服务端存储session id需要承受更高的计算成本；

2.负载均衡需要考虑共享session机制；

3.查询session id是个查库操作，用户很多时会有性能问题；

4.无法跨域。

jwt是基于token验证机制的一种，目前受大多数网站和框架支持。jwt的原理是用户认证后发一个jwt token给用户，其中包含用户的信息，用户后续请求带上这个jwt，服务端就可以自动解析出用户信息。

AK/SK:



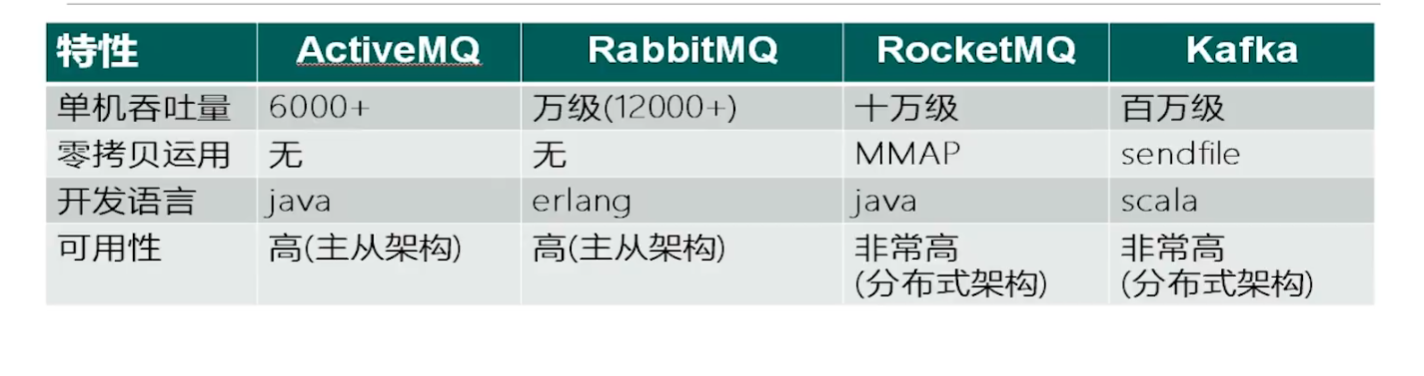
Docker虚拟化和VM的区别

Milvus架构

Cpu虚拟化技术

NIO,BIO

MQ产品



亿级流量，平均每个用户点击20-40次。对应250万到500万的用户，其中500万用户中只有10%的会付费。如果是秒杀或大促，那么这50万用户会在2-4分钟内下单，这意味着平均每秒2000-4000笔订单。假如有若干台服务器，那每台大概处理500单。每个订单大概占0.2MB的堆空间。这样每秒就会消耗100MB。如果jvm只有4G,其中堆空间3G，新生代和老年代分别占1G，2G。伊甸园区占800MB，From 和 To区各占100MB，由于From太小，在经过8s中之后，会发生youngGC，垃圾清扫后会残留100MB,这100MB本应该进入From区，但是根据动态动态对象年龄判断，这100MB会直接进入老年代。经过160s后，老年代空间使用完毕，会发生fullGC。此GC会STW(stop the world)。

mysql 优化器

SQL联表细节,MySQL JOIN 的执行过程

Linux 文件权限

机试题目

**长**连接 指在一个连接上可以连续发送多个 数据包 ，在连接保持期间，如果没有数据包发送，需要双方发链路检测包。 mysql的长连接如果长期闲置，mysql会8小时后（默认时间）主动断开该连接。 短连接：是指通讯双方有数据交互时，就建立一个连接，数据发送完成后，则断开此连接，即每次连接只完成一项业务的发送。

幂等性问题