# 1基础篇

## 1.1 system.out.pringin(3|9);输出什么？

3|9 = 0011 | 1001 =1011 =11

## 1.2 说一下转发（Forward）和重定向（Redirect）的区别？

转发是服务器行为，重定向是客户端行为

调用方式：

request.getRequestDispatcher("new.jsp").forward(request, response);  //转发 response.sendRedirect("new.jsp");   //重定向到new.jsp

数据共享方面：

forward:转发页面和转发到的页面可以共享request里面的数据  
 redirect:不能共享数据

地址栏显示方面：

forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发 给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址栏还是原来的地址.  
 redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL.

## 1.3 在浏览器中输入URL地址到显示主页的过程，整个过程会使用那些协议？

（1）首先是域名解析：

（2）浏览器发起HTTP请求

（3）选择传输协议，TCP或者UDP，对请求进行封装，加入端口号等信息

（4）网络层，通过IP协议将IP地址封装为IP数据报；然后会用到ARP协议，主机发送信息时将包含目标ip地址的ARP请求广播到网络上的所有主机，并接收返回信息，以此确定目标的物理地址，找到MAC地址；

（5）数据链路层，把ip数据报添加首部和尾部，封装为MAC帧，根据目的mac建立TCP连接，三次握手，接收端受到物理层上交的比特流后，将中间数据部分上交给网络层，然后层层向上传递到应用层；

（6）服务器响应请求并请求客户端要的资源，传回给客户端

（7）断开TCP连接，浏览器对页面进行渲染呈现给客户端；

## 1.4 TCP三次握手和四次挥手？

三次握手：

户端发送带SYND的数据给服务端，服务端传回SYN+ACK给客户端，然后客户端在发送ACK数据给服务端。

目的是为了建立可靠的通信渠道，第一次握手，服务端确定客户端发送正常，第二次，客户端确认自己发送、接收正常，对方发送接收正常。但服务端只能确定自己接收正常，对方发送正常。

直到第三次握手，服务端确定了自己发送正常，对方接收也正常。

四次挥手：

在三次握手过程中，任何一方都可以在数据传送结束后发出连接释放的通知，待对方确认后进入半关闭状态。当另一方没有数据再发送的时候，则发出连接释放通知，对方确认后就完全关闭了TCP连接。四次挥手是为了终止连接。

举个例子：A 和 B 打电话，通话即将结束后，A 说“我没啥要说的了”，B回答“我知道了”，但是 B 可能还会有要说的话，A 不能要求 B 跟着自己的节奏结束通话，于是 B 可能又巴拉巴拉说了一通，最后 B 说“我说完了”，A 回答“知道了”，这样通话才算结束。

**流程：**

1. 第一次挥手：Client发送一个FIN,用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态
2. 第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1，Server进入CLOST\_WAIT状态
3. 第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态
4. 第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入到CLOSED状态，完成四次挥手

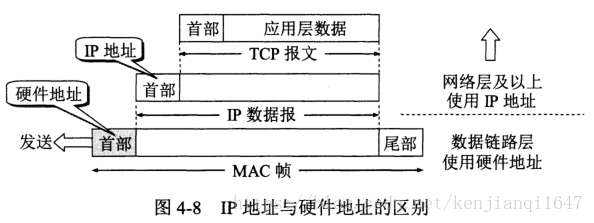
为啥会有TIME\_WAIT状态？

* 确保有足够时间让对方收到ACK包
* 避免新旧连接混淆

服务器出现大量CLOSE\_WAIT状态的原因？  
原因：可能是对方关闭socket连接，我方忙于读或写，没有及时关闭连接  
解决：

* 检查代码，特别是释放资源的代码
* 检查配置，特别是处理请求的线程配置

## 1.5 IP地址与MAC地址的区别？



IP地址是网络层及其以上层使用的地址。mac地址是数据链路层和物理层使用的地址。

通常来说，标识在网络中的一台计算机有三种基本方式，域名地址、ip地址和mac地址，分别对应应用层、网络层、物理层。网络管理就是在网络层针对ip地址进行管理，但是ip是可以自由设定的，管理很困难，mac地址则一般不可更改，所以ip地址和mac地址组合一起管理。

mac地址：

是媒体接入层使用的地址，由网络设备制造商生产时写在硬件内部。与网络无关，一般采用6字节（48比特）的mac地址。前24位由生产网卡的厂商想IEEE申请厂商地址，后24位自行分配，目前价格是1000美元买一个地址块。

mac地址通常表示为12个16进制数，每2个数之间用冒号隔开，如：08:00:20:0A:8C:6D。

IP地址基于逻辑，比较灵活，不受硬件限制，容易记忆。MAC地址与硬件一致，基于物理。各有好处。使用时根据条件来选择。

为什么用到MAC地址：

如今电脑接入互联网的方式是把主机通过局域网组织在一起，然后通过交换机与Internet连接。由于ip可以随意修改，局域网采用了mac地址来标识具体用户，就是在交换机内部通过表的方式把mac地址和ip地址一一对应，也就是Ip和mac绑定。

## 1.6 Http请求，响应报文格式？

请求报文格式：

主要由请求行、请求头部、请求正文三部分组成



（1）请求行:请求方法、URL以及协议版本组成

请求方法包括GET、HEAD、PUT、POST、TRACE、OPTIONS、DELETE以及扩展方法

协议版本格式为：HTTP/主版本号.次版本号，常用的有HTTP/1.0和HTTP/1.1

（2）请求头部:为请求报文添加一些附加信息，由名/值对组成

常见请求头如下：

|  |  |
| --- | --- |
| **请求头** | **说明** |
| **Host** | 接受请求的服务器地址，可以是IP:端口号，也可以是域名 |
| **User-Agent** | 发送请求的应用程序名称 |
| **Connection** | 指定与连接相关的属性，如Connection:Keep-Alive |
| **Accept-Charset** | 通知服务端可以发送的编码格式 |
| **Accept-Encoding** | 通知服务端可以发送的数据压缩格式 |
| **Accept-Language** | 通知服务端可以发送的语言 |

请求头部的最后会有一个空行，表示请求头部结束，接下来为请求正文，这一行非常重要，必不可少

（3）请求正文

可选部分，get请求没有请求正文

响应报文格式：HTTP响应报文主要由状态行、响应头部、响应正文3部分组成



（1）状态行：协议版本、状态码、状态码描述

状态代码为3位数字，200~299的状态码表示成功，300~399的状态码指资源重定向，400~499的状态码指客户端请求出错，500~599的状态码指服务端出错（HTTP/1.1向协议中引入了信息性状态码，范围为100~199）

这里列举几个常见的：

|  |  |
| --- | --- |
| **状态码** | **说明** |
| **200** | 响应成功 |
| **301** | 永久重定向，搜索引擎将删除源地址，保留重定向地址 |
| **302** | 暂时重定向，重定向地址由响应头中的Location属性指定（[JSP中Forward和Redirect之间的区别](http://blog.csdn.net/a19881029/article/details/45643431)）  由于搜索引擎的判定问题，较为复杂的URL容易被其它网站使用更为精简的URL及302重定向劫持 |
| **304** | 缓存文件并未过期，还可继续使用，无需再次从服务端获取 |
| **400** | 客户端请求有语法错误，不能被服务器识别 |
| **403** | 服务器接收到请求，但是拒绝提供服务（认证失败） |
| **404** | 请求资源不存在 |
| **500** | 服务器内部错误 |

（2）响应头部：与请求头部类似，为响应报文添加了一些附加信息

常见响应头部如下：

|  |  |
| --- | --- |
| **响应头** | **说明** |
| **Server** | 服务器应用程序软件的名称和版本 |
| **Content-Type** | 响应正文的类型（是图片还是二进制字符串） |
| **Content-Length** | 响应正文长度 |
| **Content-Charset** | 响应正文使用的编码 |
| **Content-Encoding** | 响应正文使用的数据压缩格式 |
| **Content-Language** | 响应正文使用的语言 |

引申：URI、URL和URN之间的区别

URI全名为Uniform Resource Indentifier（统一资源标识），用来唯一的标识一个资源，是一个通用的概念，URI由两个主要的子集URL和URN组成

URL全名为Uniform Resource Locator（统一资源定位），通过描述资源的位置来标识资源

URN全名为Uniform Resource Name（统一资源命名），通过资源的名字来标识资源，与其所处的位置无关，这样即使资源的位置发生变动，其URN也不会变化

HTTP规范将更通用的概念URI作为其资源标识符，但是实际上，HTTP应用程序处理的只是URI的URL子集

## 1.7为什么要使用索引？谈一下索引？

使用索引可以大大提高系统的性能

（1）唯一性索引保证数据库表中数据的唯一性

（2）加快检索速度，是索引创建的最主要原因

（3）加速表和表之间的连接

（4）使用分组和排序进行检索时，减少查询中分组和排序的时间

（5）通过索引，在查询过程中，使用优化隐藏器，提高系统的性能

索引的缺点：

创建索引和维护需要耗费时间

而且索引占用物理空间

表数据变化时，索引要动态维护

创建索引应该建在经常使用的列上，主要是外键，需要根据范围搜索和需要经常排序，还有经常条件判断的列上建立索引，这样可以根据索引自身的特性，加快已有的操作

创建方法：直接创建和间接创建

直接是使用CREATE INDEX 语句，间接是通过定义主键约束或者唯一性键约束

索引特征：有两个，唯一性和复合索引

## 1.8进程与线程的区别是什么？进程间的几种通信方式说一下？线程间的几种通信方式知道吗？

进程：是并发执行程序在执行过程中资源分配和管理的基本单位。就是一个应用程序的执行过程。

线程：任务调度和执行的最小单位

每个进程都有自己的地址空间，大量不确定数量的并发请求，为每个请求创建一个进程是行不通的，因此引入线程概念。

区别：线程是进程的一部分，同一进程的多线程共用内存空间，共享资源。

求稳用进程，求快选线程

## 1.9为什么要使用单例模式？手写几种线程安全的单例模式？

核心只有一个单例类的特殊类，保证系统中一个类只有一个实例，且实例易于外界访问。

目的是保证一个对象的唯一性。节省内存，但是不适用于变化的对象。

常用有懒汉式和饿汉式：

懒汉式：意味着非常懒，只在自身需要的时候才会行动，在外部第一次请求实例的时候才会创建

饿汉式：单例非常饿，在类加载的时候就迫切的创建对象

懒汉模式，它的特点是运行时获得对象的速度比较慢，但加载类的时候比较快。它在整个应用的生命周期只有一部分时间在占用资源。

饿汉模式，它的特点是加载类的时候比较慢，但运行时获得对象的速度比较快。它从加载到应用结束会一直占用资源。

从用户体验角度，首选饿汉式。

## 1.10简单介绍一下bean？知道Spring的bean的作用域与生命周期吗？

javaBean指一种特殊的java类，实现一些经常使用的简单功能，并且能够简单的重用或者插入其他应用程序中。

基于java的组件模型，由属性、方法和事件组成，能够被改动或者与其他组件结合，生成新的组件或程序。说白了，bean就是可以被反复使用的组件。

而组件是一个内部自己管理，只能通过接口代用的程序方法。

bean循序接口格式，不需要继承特别的基类和接口。

bean有五种重要的工作机制：

（1）**内省(Introspection)：**组建能够发表其支持的操作和属性。同一时候也支持在其它组件中发现反复利用的对象库，如用户权限控制和电子邮件自己主动回复等。

（2）**通信(Communication)：**生成和收集组件的消息事件。

（3）**持续(Persistence)：**存放组件的状态。

（4）**属性(Properties)：**支持组件布局的控制，包含组件占用的空间和组件的相对位置。

（5）**定制(Customization)：**开发人员可控制组件所需的改变机制。

编写要求：

（1）全部的JavaBean必须放在一个包(Package)中。

（2）JavaBean必须生成public class类，文件名应该与类名称一致。

（3）全部属性必须封装，一个JavaBean类不应有公共实例变量，类变量都为private。

（4）属性值应该通过一组存取方法（getXxx 和 setXxx）来訪问：对于每一个属性。应该有一个带匹配公用getter 和 setter方法的专用实例变量。

（5）Java Bean 类必须有一个空的构造函数：类中必须有一个不带參数的公用构造器。此构造器也应该通过调用各个属性的设置方法来设置属性的默认值。

命名规范：

（1）包命名：所有字母小写。

（2）类命名：每一个单词首字母大写。

（3）属性名：第一个单词所有小写，之后每一个单词首字母大写。

（4）方法名：与属性命名方法同样。

（5）常量名：所有字母大写。

作用域：

生命周期：

## 1.11 Srping中的事务传播行为了解吗？TransactionDefinition接口中那五个表示隔离级别的常量？

## 1.12SpringMvc原理了解吗？

## 1.13SpringAOP IOC实现原理？

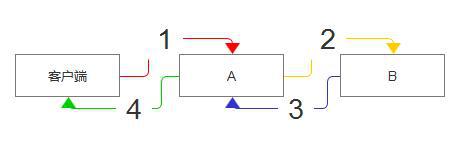
# 2进阶篇

## 2.1 消息队列MQ的套路

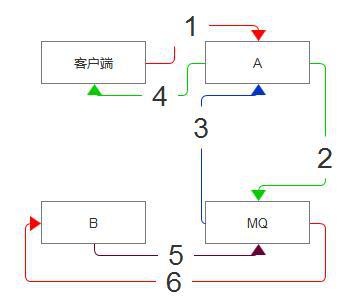
### 2.1.1 MQ应用场景？

一个队列，用来存放消息，简单来说mq就是一个容器。

假设一个场景，客户端访问A，A依赖B，但B是类似插入日志功能，并不重要。



加入一个MQ中间件，把B从主业务流程中剥离出来。



消息就可以理解成一个业务场景，队列是容器，B服务队消息的处理就是异步的，消息队列本质就是将某个业务场景暂存下来，异步处理。

因此MQ的特性就可以理解：

异步

解耦

消峰

### 2.1.2 消息队列带来的问题？

增加了系统复杂性，毕竟加了一个mq，看图就可以对比出来

降低了系统可用性，还是那句话，毕竟加了一个mq

但是一项新技术就是这样的，学时很痛苦，学会了，就成了一把利剑。上面两个问题，说到底是给开发人员带来新的学习成本。

### 2.1.3 有几种消息队列，如何选择？

中间件很多，比较成熟的也就那么几种，比如ActiveMQ、RabbitMQ，RocketMQ，Kafka。

简单说架构：https://author.baidu.com/home?from=bjh\_article&app\_id=1634119736146096

ActiveMQ早期用的比较多，现在少了，社区不够活跃，RabbitMQ现在比较多，性能也好，erlang语言开发，性能好，但源码看不懂。可惜文档资料比较少。RocketMQ用的人也比较多，集成了kafka和RabbitMQ的优点，但是文档资料少，开源软件也没人维护。kafka是业内标准，社区活跃，可以放心大胆的用，一般小公司选kafka和RabbitMQ，实力强可以去试一试RocketMQ

### 2.1.4 关于消息队列其他一些常见问题的展望？

如何保证消息队列的高可用性，消费时的幂等性，传输可靠性，顺序性，如何解决消息队列的延时以及过期失效问题，满了以后如何处理，有几百万消息持续积压几小时怎么解决

## 2.2 谈谈InnoDB和Mylsam两者的区别？

### 2.2.1 两者对比？

### 2.2.2 两者总结？

## 2.3 聊聊java中的集合？

### 2.3.1 ArrayLsit与LinkedList有什么不同？

ArrayList底层是一个数组,所以查询快,LinkedList底层是一个链表,所以增删快.

基于这个特点，ArrayList用在查询多，插入少的情况，而LinkedList用在查询少，而插入多的情况。

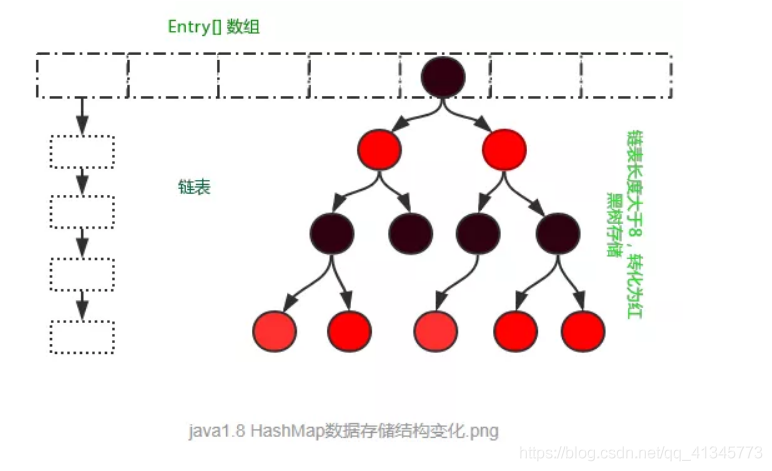
### 2.3.2 HashMap底层实现？

HashMap基于Map接口实现，元素以键值对方式存储，key不允许重复，允许null值。不能保证放入元素的顺序，是线程不安全的。

hashmap扩容很耗时，最好避免扩容，因为线程不安全，多线程推荐使用ConcurrentHashMap

HashMap用Entry数组存储key-value，每个entry类是一个单向链表，用来解决hash冲突的问题，

我们能够看出HashMap就是用数组+链表组成，数组的每个元素都是链表的头结点。



在jdk1.8中HashMap采用数组+链表+红黑树实现，链表长度超过阈值(8)时，表转换成红黑树性能进一步提升。

### 2.3.3 对红黑树的理解？

### 2.3.4红黑树这么优秀，为何不直接使用红黑树？

### 2.3.5 HashMap和Hashtable的区别？

都实现了Map接口，主要区别在三个方面：线程安全性、同步、以及速度

HashMap是Hashtable的轻量级实现，因此HashMap不安全，但单线程效率高一点；

HashMap允许键为null，HashTable不允许；

HashMap把Hashtable的contains方法改成了containsValue和containsKey

扩容方法不一样

通过hash值散列到hash表的算法不一样

迭代器不同

拓展1：为啥HashMap不安全

多个线程同时使用put方法添加，hash值一样，就会添加到同一个位置

多个线程同时检测到元素个数超过数组大小，最终只有一个线程扩容后数组会赋给table，其他线程会丢失，且会引起死循环

# 3终结篇

## 3.1 Object类有哪些方法？

### 3.1.1 Object类的常用方法总结？

Object是所有类的父类，提供了11个方法

9个被public访问修饰符修饰，使用了final关键字，不能被重写的方法有6个

需要重写的方法有三个，hashcode（）、equals（Object obj）、toString（）

2个被protect访问修饰符修饰的方法，clone（）和finalize（）

### 3.1.2 hashCode与equals？

这两个方法都是用来对比两个对象是否相等一致,但是equals()是重写过的，比较的比较全面且复杂，效率自然就低。hashCode（）效率比较高。

hashCode（）效率比较高。相对来说，就不是很可靠。

在大量并且快速对比中，先用hashCode去对比，不一样就代表肯定不相等，相同再用equal去对比。

总结来说:hashCode 是为了提高在散列结构存储中查找的效率，在线性表中没有作用。

equals和hashCode需要同时覆盖。

### 3.1.3 ==与equals？

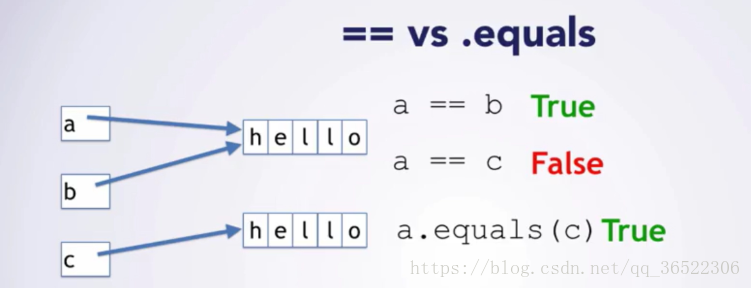
JVM中内存分为堆内存和栈内存，我们创建一个对象时，会调用对象的构造函数来开辟空间，将对象数据存储到堆内存中，同时在栈内存中生成对应的引用。后续代码调用的时候用的都是栈内存中的引用，当然，基本数据类型是存储在栈内存中的。

==和equals的区别

==是判断两个变量或实例是不是指向同一个内存空间，equals是判断两个指向的内存空间的值是不是相同。

==针对内存地址，equals对字符串内容进行比较

一个是引用，一个是指



## 3.2 ConcurrentHashMap相关问题？

### 3.2.1 ConcurrentHashMap和Hashtable的区别？

是java5中支持高并发、高吞吐线程安全的HashMap实现。

而hashtable是一种能快速插入和查询的数据结构

他们存储的内容都是键值对，而且线程安全。区别在于他们针对线程安全的策略不一样，相比来说concurrentHashMap锁定的粒度更细，所以性能更高一点。

### 3.2.2 ConcurrentHashMap线程安全的具体实现方式？

## 3.3 谈谈synchronized和ReentrantLock的区别？

都是加锁方式同步，而且都是阻塞式的同步

最大区别就是对于Synchronized来说，它是java语言的关键字，是原生语法层面的互斥，需要jvm实现。而ReentrantLock它是JDK 1.5之后提供的API层面的互斥锁，需要lock()和unlock()方法配合try/finally语句块来完成

便利性：很明显Synchronized的使用比较方便简洁，并且由编译器去保证锁的加锁和释放，而ReenTrantLock需要手工声明来加锁和释放锁，为了避免忘记手工释放锁造成死锁，所以最好在finally中声明释放锁。

锁的细粒度和灵活度：很明显ReenTrantLock优于Synchronized

## 3.4 线程池相关问题？

### 3.4.1 为什么要用线程池？

多线程可以发挥多核处理器的算力，但也有自己的问题。

线程创建和销毁需要时间，本身也要占用内存空间，线程回收也会给GC带来压力。因此有了线程池的概念，让线程进行复用。

### 3.4.2 java提供了哪几种线程池？他们各自的使用场景是什么?

java提供四种线程池

**newSingleThreadExecutor**

创建一个单线程的线程池。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

**newFixedThreadPool**

创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

**newCachedThreadPool**

创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说JVM）能够创建的最大线程大小。

**newScheduledThreadPool**

创建一个定长线程池，支持定时及周期性任务执行。

### 3.4.3 创建线程池的方式？

Executors是一个线程相关的工具类。主要提供了以下几种创建线程池的方法：

**ThreadPoolExecutor**

ThreadPoolExecutor(int corePoolSize, //核心线程数量

int maximumPoolSize, // 最大线程数量

long keepAliveTime, // 存活时间

TimeUnit unit, // 时间单位

BlockingQueue<Runnable> workQueue, //阻塞队列

ThreadFactory threadFactory) // 拒绝策略

## 3.5 Nginx相关问题？

### 3.5.1 简单介绍一下Nginx？

反向代理服务器，Nginx本身就能够托管站点，进行http服务处理，也能够作为反向代理服务器使用。

### 3.5.2 为什么要用Nginx？

Nginx优势：高性能，高稳定，低消耗，高可扩展性，支持热部署，免费且开源

### 3.5.3 Nginx的四个主要组成部分？

Nginx由内核和模块组成，内核微小而简洁，仅仅通过参照配置文件将客户端请求映射到一个location block，而在这个location中所配置的每个指令将会启动不同的模块去完成相应的工作

* Nginx 二进制可执行文件：由各模块源码编译出一个文件
* Nginx.conf 配置文件：控制Nginx 行为
* acess.log 访问日志： 记录每一条HTTP请求信息
* error.log 错误日志:定位问题

# 4面试篇

## 4.1 基础

### 4.1.1 描述java的Error，Exception的区别，举例几种RuntimeException？

在java中Error和Exception都继承与Trowable类，是java对于异常处理的良好总方式。

区别在于，Exception是可预料的，异常发生，可以获取，并针对其进行业务外处理，而Error是不可预料的。这个异常会导致JVM不可处理。

而Exception分为检查性异常和非检查性异常，根本区别在于前者可以用try catch捕获，后者可以通过规范避免发生。

RuntimeException的子类都是非检查型异常，

 //常见的RuntimeException

 //1.ArithmeticException()运算异常

 //2.ClassCastException()类的转换异常

 //3.NullPointerException()空指针异常

 //4.IndexOutOfBoundsException()数组指针越界异常

### 4.1.2讲解java克隆的作用以及深浅克隆的原理区别？

指针概念给开发人员带来不便，java中取消了指针的概念，但事实上只是没有明确提出而已，实质上每个new 语言返回的都是一个指针的引用，但大部分情况下开发人员不需要关心如何去操作这个指针。

java中所有类默认继承自Object类，Object中提供了一个clone方法，这个方法返回一个Object对象的复制，这个复制函数返回一个新的对象。（新开辟一个存储空间，但是存储空间的存储内容一样）

需要注意的是，clone方法的保护机制被声明protected。保证在User类中才能被克隆User对象。

深浅复制的区别:

浅复制（Shallow Clone）：被复制对象的所有变量都含有与原来对象相同的值，而所有对其他对象的引用仍然指向原来的对象。换言之，浅复制仅仅复制所考虑的对象，而不复制它所引用的对象。

深复制（Deep Clone）：被复制对象的所有变量都含有与原来对象相同的值，除去那些引用其他对象的变量。那些引用其他对象的变量将指向被复制的新对象，而不在是原有的那些被引用的对象。换言之，深复制把复制的对象所引用的对象都复制了一遍。

### 4.1.3线程同步的几种方式？

多个线程共享同一个全局变量，可能受到其他线程的干扰，导致数据问题，这种现象较线程安全问题。

线程安全解决方法？

synchronized --自动锁

lock--手动锁

线程之间如何同步？

同步是保证数据的原子性，原子性就是数据不能受到其他线程的干扰

多线程中的线程执行顺序是依靠那个线程先获得CPU的执行权，线程的优先权只是获取CPU执行权的概率高点，不保证一定先执行。

保证线程顺序的几种方式总结：

1. 通过Object的wait和notify
2. 通过Condition的awiat和signal
3. 通过一个阻塞队列
4. 通过两个阻塞队列
5. 通过SynchronousQueue
6. 通过线程池的Callback回调
7. 通过同步辅助类CountDownLatch
8. 通过同步辅助类CyclicBarrier

### 4.1.4 TheadLocal的原理及使用场景？

ThreadLocal一般被用来存储线程变量的副本，各个线程存储数据都是独立的，互不产生影响。由于这种隔离性，使得线程之间没有资源竞争的关系，因此也不需要做线程同步。

ThreadLocal不是线程，而是变量，使用它的时候，为每个线程提供独立的变量副本，每个线程独立改变自己的副本，而不会影响其他线程对应副本。

原理:

在ThreadLocal类中有一个Map，用来存储每个线程的变量副本，键为线程对象，值对应变量副本。

同步机制:

多线程同步机制采用以时间换空间，而ThreadLocal采用以空间换时间。前者用一份变量，让不同线程排队访问，而后者给每个线程提供一个变量。

当然同步机制针对的多个线程对相同资源的并发访问，而ThreadLocal是隔离多个线程的数据共享。

使用场景:：用来解决数据库连接，Session管理

## 4.2 代码编写

### 4.2.1 编写一个单例类？

单例类注意点：

将构造方法私有化；

有一个静态属性指向实例；

提供一个公有的静态方法想外面提供这个实例。

饿汉式：

public class Singleton{

private Singleton(){}

private static Singleton instance =new Singleton();

public Singleton getInstance(){

return instance;

}

}

懒汉式：

public class Singleton {

private Singleton() {

}

private static Singleton singleton;

public static synchronized Singleton getInstance() {

if (singleton == null) {

singleton = new Singleton();

}

return singleton;

}

}

## 4.3 算法及数据结构

### 4.3.1如何判断两个单向链表相交，如何找相交点？

两个单向链表有共同点，那么第一个共同点开始，后面节点都会重叠。

我们可以使用栈来解决问题，创建两个栈，每遍历到一个节点，就将该节点入栈，两个链表都入栈后，通过top判断栈顶节点是否相等。

### 4.3.2系统有大量的订单数据日志，每条数据都有一个时间戳，要求对这批数据进行时间戳排序？

## 4.4 数据库

### 4.4.1一个表tb1，字段name，calss，score，用一条语句查出每个班的及格人数和不及格人数

**SELECT**

**t1.class AS "班级编号",**

**t1.total1 AS "及格人数",**

**t2.total2 AS "不及格人数"**

**FROM**

**(SELECT COUNT(\*) AS "total1",class FROM tb1 WHERE score >= 60 GROUP BY class) t1,**

**(SELECT COUNT(\*) AS "total2",class FROM tb1 WHERE score < 60 GROUP BY class) t2**

**WHERE t1.class = t2.class;**

## 4.5 设计

设计一个流量控制服务，对访问的ip地址做白名单和访问频率控制，白名单全部房型，其他ip访问频率按照每小时访问次数来限制，按照合作等级，分为高、中、普通三级，请给出结构世界，考虑ipv4即可，方案包含下面几点:

a 服务接口描述

b 预估单机实现的资源消耗

c 高可用性如何实现

## 4.6 附加

### 4.6.1统计日志中频次醉倒的10个ip地址

### 4.6.2根据规律填充括号

1 、11 、21 、1211、111221 、312211

# 5面试汇总篇

## 5.1 说一下你项目的架构

## 5.2 如何保证多线程安全

## 5.3 服务器的架构

## 5.4 HashMap的put

## 5.5 SpringCloud

## 5.6 如何做负载平衡