https://blog.csdn.net/weixin\_40986207/article/details/79703296

1. Static，Transient，foreach循环的原理

static修饰属性、方法、静态块、静态导包“类名.方法名”或者“类名.属性名”，静态块，有些代码必须在项目启动的时候就执行需要使用静态代码块，静态导包，将类的方法直接导入到当前类中。

transient，只要实现序列化的接口类的属性就可以被序列化，将不需要序列化的属性钱添加关键字transient该属性就不会被序列化（深入理解序列化）

无论是ArrayList之所以可以使用foreach循环是因为所有的List都是Collection的子类，Collection又是Iterable的子接口，Iterable有iterator的方法，迭代器，可以foreach（可以深入理解一下迭代器）

2.switch,java反射机制

反射机制，简单来说程序在运行的时候根据类的名称就可以知道类的所有信息。为什么要反射呢，静态编译，在编译时确定类型，绑定对象；动态编译，运行时确定类型，绑定对象。反射机制的有点就是可以动态的创建对象和编译，有很大的灵活性

3.ArrayList、HashMap、LinkedList、HashTable的源码

ArrayList相当于一个动态数组，初始长度是10，超出长度将按照增加50%的长度增加

HashMap、HashTable、ConcurrentHashMap之间的区别

HashMap扩容的时候会调用resize()方法，就是这里的并发操作容易在一个桶上形成环形链表，这样当获取一个不存在的key时，计算出的index正好是唤醒链表就会出现死循环

ConcurrentHashMap使用了分段锁机制segument

HashMap的遍历方式，一种是map.entrySet().iterator;还有一种是map.keySet().iterator;推荐第一种

4.使用的框架以及spring的原理，bean初始化和销毁的时候做的事；Springioc,aop原理，SpringMvc原理，Springboot原理，Spring clould原理， Dobbo原理，MyBaties原理，Redis原理

1. 数据库索引、连接、优化问题，数据库范式

什么样的字段适合做索引，跟业务相关的，且离散度大，分布均匀

myisum有三个文件，一个是表结构，一个是索引，还有一个是数据，每次查询具有原子性，但是不具有事务支持，表级锁

innodb有两个文件，一个是表结构，一个是主键建造的索引，最下面是存储数据的，提供事务支持，支持行级锁，只有where是主键的时候才会是行级锁，其他是表级锁

sql语句尽量避免!=huo<>操作符，以及用or来作为连接条件，in和not in也要慎用，否则不会用索引而是全表扫描，连续数值能用between就不要用in了；尽量避免在where子句中对字段进行表达式操作，例如select id from table where num/2=100;应该改为num=100\*2;尽量避免在where自居中对字段进行函数操作；

很多时候用exists代替in是一个好的选择，例如select num from a where num in(select num from b)用羡慕的语句替换select num from a where exists(select 1 from b where num=a.num)

单个索引要注意的点,索引列不要参与计算,or的两侧要么都索引，要么都不是索引列，模糊匹配的时候%不要在头部，使用or失效的原因是，假如or连接的两个查询条件字段中有一个没有索引的话，引擎会放弃索引而产生权标扫描

联合索引，比如(A,B,C)会先匹配A再匹配B,C，把最常用的，筛选最多的字段放在左侧

使用or前后两个条件如果前后两个条件都用索引才会索引，如果有一个不是索引则不会索引，使用union all或者in来优化

select pad from sbtest1 where k='501462' union all select pad from sbtest1 where k='502480';

会使用索引，or的条件不在同一个表的情况下，使用union all来改写扫描行数减少且会走索引

explain select a.pad,a.c from sbtest1 a,sbtest2 b where a.id=b.id and a.c='123' union all select a.pad,a.c from sbtest1 a,sbtest2 b where a.id=b.id and b.c='1234';

内连接外连接，左外连接，右外连接

select \* from A inner join B on A.id = B.id取得是一个交集

select \* from A left join B on A.id = B.id取的是所有A符合条件的B不满足的用NULL表示

主从表的复制原理，主表有一个bin-log日志，记录的是主表所有进行增删查改的日志，复制的时候会开一个线程将主表的bin-log传给从表，然后在从表中执行bin-log内容

第一范式

两列属性相近或相似或医院，尽量合并属性一样的列，，例如物品1，物品2，物品1价格，物品2价格这样是不符合第一范式的

第二范式

每一行的数据只能与其中一列相关，一行数据至做一件事，例如订单编号，房间号，联系人，联系人电话，身份证号，一个人定多个房间就会有多条数据，那么订单多条数据，联系人都是重复的，会造成冗余，可以拆分成这样的，订单编号，房间号，联系人编号还有一个联系人的表，联系人编号，联系人，联系人电话，身份证号

第三范式

表的每个属性要和主键有直接的关系而不是间接的关系比如，学号，姓名，年龄，性别，学校，学校地址，学校电话，应该拆分开来（学号，姓名，年龄，性别），（学校，学校地址，学校电话）分成学生的属性一部分，学校的属性一部分

6.设计模式，项目中用到的设计模式，怎么用的，设计模式优缺点，UML图

主要了解单例模式（饿汉，懒汉，枚举），工厂，代理，装饰， 适配器

单例模式，一个类只能有一个实例，并提供一个全局访问他的访问点，主要对应频繁创建与销毁节省资源，判断是否已经有，如果没有就创建，有就使用，关键点在于构造器是私有的

在容器中的对象如果实现了接口则采用jdk动态代理，没有实现接口使用cglib代理，cglib使用了非常底层的字节码技术，原理是使用字节码技术创建一个子类，并且在子类中采用方法拦截的技术拦截所有父类方法的调用，顺势植入横切逻辑，但因为采用的是继承，所以不能对final修饰的类进行代理，所以无法对final修饰的方法进行代理

静态代理在程序运行时，使用反射生成的；静态代理是程序员使用特定的工具，在程序运行之前已经生成了.class了

这里引申一个反射机制的问题，为什么使用反射，new出来不是很容易么，为什么要Class.forName(“类所在包以及包名”)

7.多线程同步、锁synchronized和ReentrantLock（重点）

线程独享和方法独享、ThreadLocal做缓存

8.数据库存储日期格式，如何考虑时区转换问题

使用timeStamp，与时区无关系，存储的才是统一的，可以拿出timestamp来转换相应的时区

9.java Object类中的方法

10.多服务器集群的情况下，session的同步问题

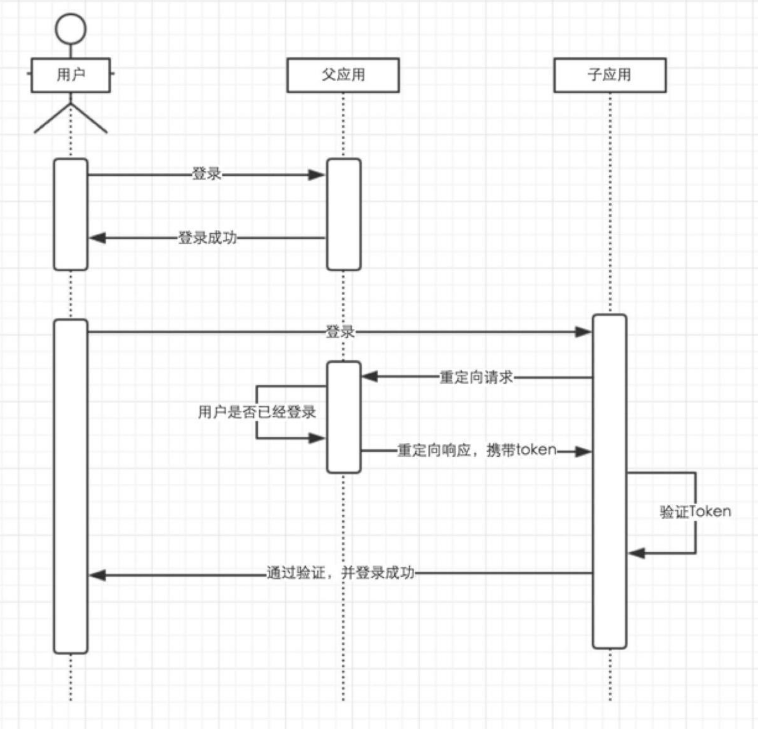
每个系统自己存储，有改动和通知其他的系统，这样不好；还有一种方式是使用一个专门的服务器来存储，使用redis来缓存session,但是断电之后数据会全部消失；和上一个方式相同，都是将session存储在同一个服务器中，但是把session信息存储在数据库中

11.我方系统单点登录所使用的方法

父应用和子应用来回重定向的方式来进行通信，实现信息的安全传递

比如说用户访问的是http巴拉巴拉/finace/index.html这是一个子应用的服务器地址，该服务器就会进行拦截重定向到sso服务器的登录地址并且将一开始用户访问的地址送给sso，访问sso服务器地址进入登录页面，登录成功后sso重定向响应到finace系统验票的地址并且给finace一开始用户访问finace的地址，finace拿到token就去指定的地方去验票，验票成功后进入http巴拉巴拉/finace/index.html让用户进行操作。

一般说来，大型应用会把授权的逻辑与用户信息的相关逻辑独立成一个应用，称为用户中心。   
用户中心不处理业务逻辑，只是处理用户信息的管理以及授权给第三方应用。第三方应用需要登录的时候，则把用户的登录请求转发给用户中心进行处理，用户处理完毕返回凭证，第三方应用验证凭证，通过后就登录用户。



12.网络的五层结构分别是干嘛的

网络把许多计算机连接在一起，而互联网则把许多网络通过路由器连接在一起。与网络相连的计算机常称为主机

互联网从工作方式上可以分为两大块，一个是边缘部分，还有一个是核心部分，边缘部分是用户直接使用的，而核心部分是路由器等，为边缘部分提供服务的

路由器是实现分组交换的关键构建，其任务是转发收到的分组，这是网络核心最重要的部分

数据的发送速率越高，其传送的总时延就越小

七层之间相互独立的，只要每一层提供的接口是不变的，无论内部怎么变换对其他层都是没有影响的

OSI概念清楚完整，但是复杂不实用，TCP/IP体系结构不同，但是却得到了非常广泛的应用。TCP/IP是一个四层的体系结构，它包含应用层、运输层、网际层、网络接口层，是为了解决不同网络的互联问题

五层协议的体系结构：物理层、数据链路层、网络层、运输层、应用层

应用层：是体系结构中的最高层任务是通过应用进程间的交互来完成特定网络应用。应用层协议定义的是应用进程间通信和交互的规则，对于不同的网络应用要有不同的应用层协议，比如DNS,HTTP,SMTP,TELENT我们把应用层交互的数据单元叫做报文

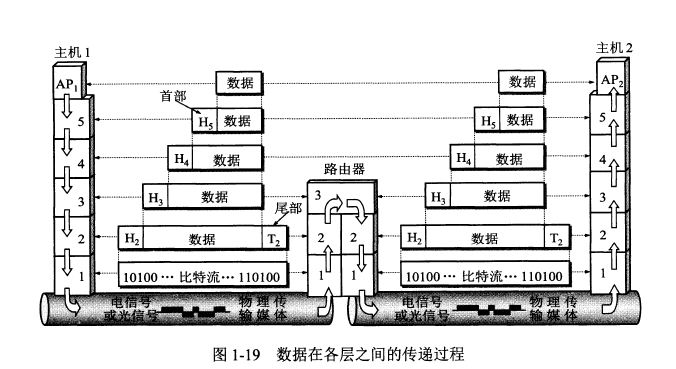
运输层：运输层就是负责向两台主机中进程之间的通信提供通用的数据传输服务。运输层主要使用以下两种协议，TCP传输控制协议，提供面向连接的、可靠的数据传输服务，其数据传输的单位是报文段；UDP用户数据报协议，提供无连接的、尽最大努力的数据传输服务，其数据传输的单位是用户数据报

网络层：负责为分组交换网上的不同主机提供的通信服务。网络层把运输层产生的报文段或用户数据报文封装成分组或包进行传送。网络层使用IP协议，因此分组也叫做IP数据报。网络层的另一个重要任务就是要选择合适的路由。互联网使用的网络层协议是无连接的网际协议IP和许多路由选择协议，因此网络层又叫做网际层或者IP层

数据链路层：数据链路层要把IP数据报文组装层帧，在两个相邻的结点间的链路上传送帧

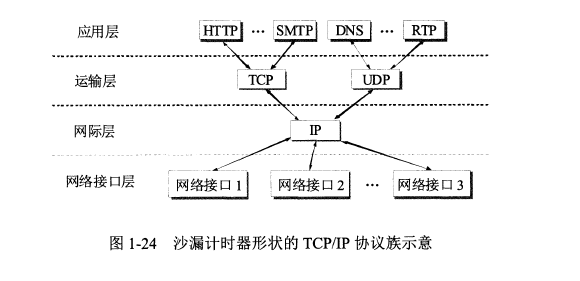
物理层：物理层上所传数据的单位是比特

互联网所使用的各种协议中，最重要的是TCP,IP两个协议。现在指的表示互联网所使用的整个TCP/IP协议族



一层一层的加自己需要的控制信息，但是到了数据链路层，信息被分为两部分，分别加到本层单元的首部和尾部，到了物理层则不需要控制了，因为是比特流的传送，所以不再加上控制信息

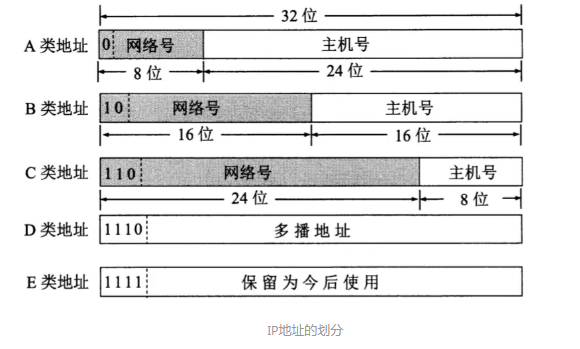
协议和服务在概念上很不一样。协议不能嘉假定一切都是正常的和非常理想的，还要非常仔细的检查这个协议能否应付各种异常情况。TCP/IP可以为各式各样的应用提供服务，运行IP协议在各式各样的网络构成的互联网上运行



数据链路层属于计算机网络的底层。数据链路层使用的信道主要有点对点信道，广播信道，数据链路层的三个基本问题：封装成帧、透明传输和差错检测，以太网的MAC层硬件地址

网络层通过选择算法，为IP分组从源主机到目的主机选择一条适合的传输路径，为传输层端对端数据传输提供服务

IP协议配套使用的还有三个协议：地址解析协议ARP，网际控制报文协议ICMP，网际组管理协议IGMP。整个互联网是一个单一的、抽象的网络。IP地址就是给互联网上每一台主机或是路由器分配一个在全世界范围内唯一一个2位的标识符，IP地址的结构使我们可以在互联网上方便的进行寻址



A类地址的地址范围是1.0.0.0到127.255.255.255，255.0.0.0

B类地址的地址范围是128.0.0.0到191.255.255.255,255.255.0.0

C类地址的地址范围是192.0.0.0到223.255.255.255,255.255.255.0

子网掩码的作用和计算方式是重点，后面要着重看一下



UDP没有连接，所以没有拥塞控制，支持一对一、一对多、多对一和多对多的交互通信，首部开销少

UDP的首部格式

(1)源端口 源端口号。在需要对方回信时选用，不需要时可全0

(2)目的端口 目的端口号，这是在重点交付报文时使用的

(3)长度 UDP用户报文的长度，其最小值是8

(4)检验和 检验UDP用户数据报在传输中是否有错，有错就丢弃

TCP只能有两个端点，每一条连接只能是点对点，通过TCP的连接传送的数据无差错、不丢失、不重复、并且按序到达，TCP提供全双工通信

滑动窗口确认，比如发送5个，返回一个确认，这种方式效率会比较高，不需要一个一个等待确认，但是不能从发送方反映出接收方已正确收到的所有分组信息

TCP报文首部是20个字节固定的，后面有4n字节，根据需要而增加的选项

(1)源端口和目的端口各占2个字节，分别写入源端口号和目的端口号

(2)序号占四个字节序号范围是[0,2的32次方-1]

(3)确认号占4个字节，期望收到对方下一个报文段的第一个数据字节的序号

(4)数据偏移占4位它指出，TCP报文段的数据起始处距离TCP报文段的其实处有多远，这个字段试计算是指出TCP报文的首部长度

(5)保留占6位，目前置为0

(6)紧急URG当URG为1时表妹紧急指针字段有效

(7)确认ACK，仅当ACK=1时确认号字段才有效

(8)推送，当两个应用进程进行交互式通信时，有时在一段的应用进程希望在键入一个命令后立即就能接收到对方的响应

(9)复位RST，当RST=1表明TCP连接严重出错，必须释放连接

(10)同步SYN，在连接建立时同步序号SYN=1,ACK=0表明是一个请求报文段，SYN=1,ACK=1表示为连接接收报文

(11)终止FIN用来释放一个连接

TCP进行拥堵控制的算法有四种，慢开始、拥塞避免、快重传、快恢复

慢开始和拥塞避免，拥塞窗口的大小取决于网络的拥堵程度，并且动态的在变化。当网络发生拥塞时路由器就要丢弃分组因此只要发送方没有按时收到到达的确认报文，也就是超时，就可以猜想网络拥堵了

TCP连接建立过程中要解决以下三个问题

(1)要使每一方能确知对方的存在

(2)要允许双方协商以下参数，比如说窗口最大值，是否使用窗口扩大选项以及服务质量等

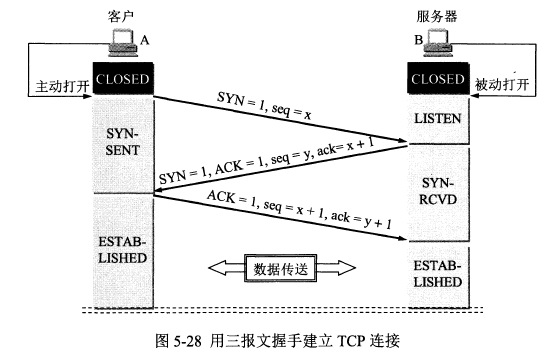
(3)能够对运输实体资源进行分配

TCP连接的建立采用的是客户服务器方式。主动发起连建立的叫客户，被动的叫服务器

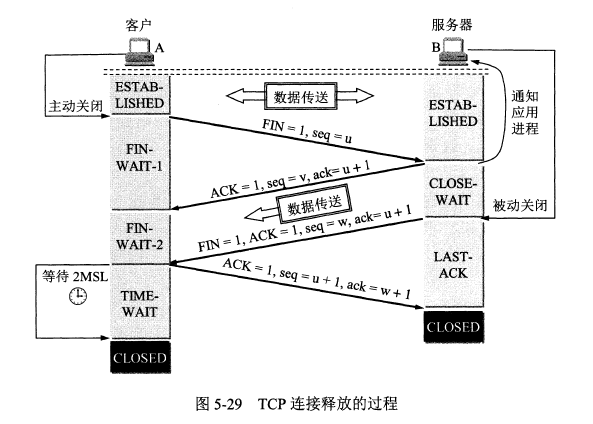
TCP建立连接的过程叫做握手，握手需要在客户和服务器之间交换三个TCP报文段

TCP为什么要进行三次握手，为了保证数据传输的可靠性，通信双方需要判断自己收到的包是否被接收方收到，如果没收到就需要重发。为了实现这个需求，很自然就会引出序号和确认号的使用

为什么A最后还要发送一次确认呢这主要是防止已失效的连接请求报文突然又传到了B，因而产生错误。



三次握手四次挥手重点



13.盲点：IO,NIO,多线程,消息通知MQ,kafuka,spring clould,jvm调优概述

MQ是消息中间件，是一种在分布式系统中应用程序借以传递消息的媒介常用的有activeMQ,kafka,rabbitMQ

支持多种协议传输，TCP,SSL,NIO,UDP

MQ的消费-生产者模型的一个典型代表，一端往消息队列中不断的写入消息，而另一端则可以读取或者订阅队列中的消息。将项目中一些无需及时返回且耗时的操作提取出来，进行了异步处理，而这种异步处理的方式大大的节省了服务器的请求响应时间，从而提高了系统的吞吐量

通信模式

点对点

(1)一个消息只能被一个服务接收

(2)消息一旦被消费，就会消失

(3)如果没有被消费就会一直等待

(4)多个服务监听同一个消费空间，先到先得

发布/订阅模式

(1)一个消息可以被多个服务接收

(2)订阅一个主题的消费者，只能消费自它订阅之后发布的消息

(3)消费端如果在生产端发送消息之后启动是接收不到消息的，除非生产端对消费进行了持久化

实际操作

不论是生产者还是消费者都要继承一个模板，消费者要继承一个MessageListener来监听生产者放进去的message好取用出来

，这个模板要有connectionFactory，生产者要做任务将信息放到message里面，消费者要重写MessageListener的onMessage方法获得生产者放进的数据并使用

生产者把数据放在哪个队列，消费者到同等姓名的队列去即可取到信息进行消费

redis采用的是定期删除+惰性删除策略

为什么不用定时删除呢，定时删除要检测key，看他的超时时间是否到了，在高并发的情况下是非常消耗内存的。

定期删除，比如redis采用的是每100ms就清理一次过期的，不是全部都清理，而是选择一部分进行清理，那么那些被忽略掉的key怎么办，就用到了惰性删除策略，就是说你在获取key的时候redis会检测，如果是过期的就会删除

如果没有清理掉key，并且这个key也没有被谁使用，则需要配置

maxmemory-policy volatile-lru后面这个volatile-lru是一种策略

有好几种策略，一种是allkeys-lru当内存不足以使用新增的key时，移除最近未使用的key，这种策略推荐使用

如何应对缓存雪崩和缓存穿透的问题

缓存穿透就是黑客故意去请求缓存中不存在的数据使得请求都怼在数据库上导致数据库异常，解决办法就是利用互斥锁，缓存失效的时候先去获得锁，如果没有获取到，则先休眠一段时间；采用异步更新策略

缓存雪崩，即缓存同一时间大面积的失效这时候又出来新的请求导致都到数据库中导致数据库连接异常

给缓存的失效时间，加上一个随机值，避免集体失效

使用互斥锁，但是该方案吞吐量明显下降了，双缓存，缓存A的失效时间是20分钟，缓存B不设置失效时间.

1. 多线程

线程之间的同步和异步，同步是指顺序执行，执行完这个才能执行下一个，需要等待协调运行；异步就是彼此独立，在等待某事件的过程中继续做自己的事。

并行和串行：并行和串行指的是任务之间的执行方式，串行是指多个任务时，各个任务按顺序执行，完成一个之后才能进行下一个，并行指的是多个任务可以同时执行。

并行和并发，并行指的是多个cpu或者多个机器同时执行一段处理逻辑，是真正意义上的同时

并发是通过cpu调度算法，让用户看上去同时执行，但是实际上cpu操作不是真正意义上的同时。并发往往在场景中有公用的资源，那么针对这个公用的资源往往产生瓶颈，我们会用TPS或者QPS来反映这个系统的处理能力

同步：java中的同步指的是通过人为的控制和调度，保证共享资源的多线程访问成为线程安全，来保证结果正确，如在代码上加入synchronize关键字，在保证结果准确的同时，还能提高性能，才是优秀的程序，线程安全的优先级别高于性能

Synchronize和ReentrantLock的区别

如果使用使用Synchronize，A不释放，B一直等下去不能被中断

如果使用ReentrantLock，A不释放，可以使B在等待了足够长的时间之后，中断等待，干别的事

什么是线程安全，一个对象被多个线程同时访问，还能保持其内部属性的顺序性以及同步性，则认定为线程安全，要想线程安全有三种方式，一种是被valatile还有被synchronize等关键词修饰，或者使用java.util.concurrent下面的类库

如果一个类集成Thread不适合资源共享，但是如果实现了Runnable接口的话，则很容易的实现资源共享

使用runnable的好处

Run(方法是多线程程序的一个约定，所有多线程代码都在run方法里面，Thread类实际上也实现了Runnable接口的类，所有的多线程代码都是通过运行Thread的start()，最终还是通过Thread的对象API来控制线程的，线程API是基础

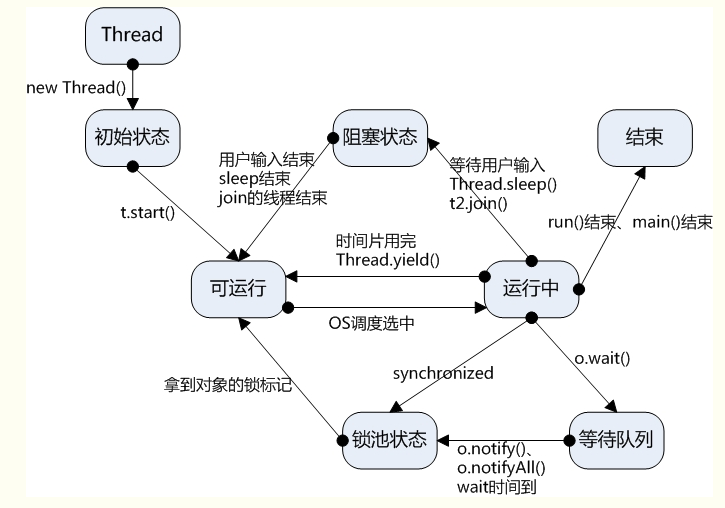
(1)适合多个相同的程序代码的线程去处理同一个资源

(2)可以避免java中的单继承限制

(3)增加程序的健壮性，代码可以被多个线程共享，代码和数据独立

(4)线程池只能放入事先Runnable或callable类线程，不能直接放入Thread类

以下是线程的调度



1. 调整线程的优先级，java优先级越高越越容易获得较多的机会运行

(2)线程睡眠：Thread.sleep（long millis）方法，使得线程转到阻塞状态，睡眠结束转为就绪态

(3)线程等待，Object.wait()方法，使用notify()或notifyAll()唤醒

(4)线程让步Thread.yield暂停当前执行的线程对象

(5)线程加入，join()方法，等待其他线程终止，在当前线程中调用另一个线程的join()方法，则当前线程转入阻塞状态，为什么要使用到join方法，有的主线程生成了子线程，但是子线程处理了大量的运算，导致主线程处理完其他事情之后需要使用到其他的事务，需要用到子线程的处理结果，这时候就需要用到join了，主线程一定会等子线程结束之后再结束

Yield()方法对应了如下操作，先检测当前是否有相同优先级处于同可运行状态，如果有就把cpu的占有权交给此线程，否则继续原来的线程，所以叫做退让把运行的机会让给了同等优先级的其他线程

Interrupt()不要以为他是中断某个线程，他只是发送一个中断信号，让线程无线等待时抛出异常结束进程，如果异常被吃掉还是不会中断

重点：wait和notify还有synchronize

使用上是这三个配合着一起使用，从功能来说线程在获取对象锁之后，主动释放锁，同时本线程休眠，直到有其他线程调用对象的notify()唤醒该线程，才能继续获得锁，注意，notify()方法在synchronize{}块执行之后才释放锁；wait()和sleep()的区别在于，wait()在释放了cpu的同时释放了对象锁的控制；sleep()在睡眠时仍占用对象锁，而wait()会释放锁；

用户线程和守护进程的区别在于是否等待主线程，依赖于主线程而结束

多线程的使用synchronize需要配合wait(),notify()使用，锁是加在当前class对象上，因此，所有对该方法的调用，都必须获得class对象的锁，notify()是随机唤醒一个等待的线程

调用join()和sleep()方法，sleep()时间结束或被打断，join()中断，io都会回到runnable状态，等待jvm调度

调用wait()，使该线程处于线程池(wait blocked pool)知道notify()或者notifyAll(0，线程被唤醒放到锁定池(lock blocked pool)，释放同步锁，使得线程回到可运行状态runnable

Start()方法的调用后并不是立即执行多线程代码，而是使得该线程变为可运行态，什么时候运行由操作系统决定，实际上多线程的执行顺序都是不确定的，每次执行的结果都是随机的

对于runnable状态的线程加同步锁，可使其进入(lock blocked pool)释放同步锁可回到runnable状态thread类中的yield方法让一个正在运行的状态线程进入runnable

Reentrantlock是重入锁，比上面的更强大，可以终端可定时、可中断，在高并发的情况下，比synchronize更有优势；Reentrantlock提供了公平锁和非公平锁，公平锁获取的是先进先出，而非公平锁是可以插队的。Reentrantlock在构造时可以指定锁是否公平，在使用重入锁时，最后一定有在finally里面释放锁，synchronize时jvm最后自动释放的

1. 算法，找出1000万个随机数，主要是效率和解决冲突

先建立一个数组，数组大小为1000万，主要是生成确定的1000万个数字，然后根据生成的不同下标来确定随机排列，代码如下

**int** index = 0;

**int** tmp = 0;

**int** count = 0;

**while**(value >0 ){

//这是生成的随机数的下标

index = random.nextInt(value);

//找到这个下标的值应该是什么

tmp = list[count+index];

list[count+index] = list[count];

//把这个值赋给当前第几个生成数

list[count] = tmp;

count++;

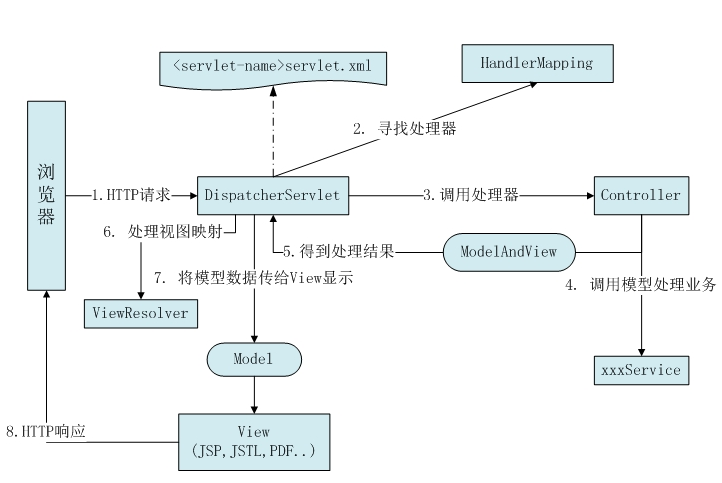
value--;

}

1. java对象的生命周期

创建、应用、不可见（超出其作用域）、不可达（不再被任何强引用所持有）、收集（对象已经不可达，最好不要重载falized()方法，影响jvm的对象分配与回收速度、可能造成对象的再次复活）、终结（执行了falized()方法后仍然处于不可达状态则该对象处于终结状态）、对象空间重新分配（对象已经彻底消失）

1. springMVC的原理



1. 客户端请求提交到DispatcherServlet
2. 由DispatcherServlet控制器查询一个或多个HandlerMapping，找到处理请求的Controller
3. DispatcherServlet将请求提交到Controller
4. Controller执行完业务逻辑之后将返回ModelAndView
5. DispatcherServlet查询一个或多个ViewResolver视图解析器，找到ModelAndView指定视图
6. 视图将负责把结果显示到客户端
7. equals,hashcode,==的区别

Java中有两类数据对象，一种是基本数据类型，还有是引用数据类型，基本数据类型使用==的时候就是比较他们的值是否相等，但是引用数据类型使用==的使用比较的是他们的内存地址，除非是同一个new出的对象，否则，一般都为false

可见==是比较的栈中对象引用的地址，要是想比较堆中对象的内容就要重写equals

如果重写equals方法有必要重写hashCode方法，确保两个equals的对象hashCode也是一样的，如果两个对象相同，那么他们的hashCode应该也相等，不过这只是个规范

1. jvm的结构
2. 类加载器，在jvm启动的时候，或者是在类运行的时候,将需要的类加载到jvm中
3. 执行引擎：负责执行class文件中包含的字节码指令
4. 内存区：在jvm运行的时候操作所分配的内存区，运行时内存区主要可以划分为方法区、java堆、java栈、程序计数器、本地方法栈

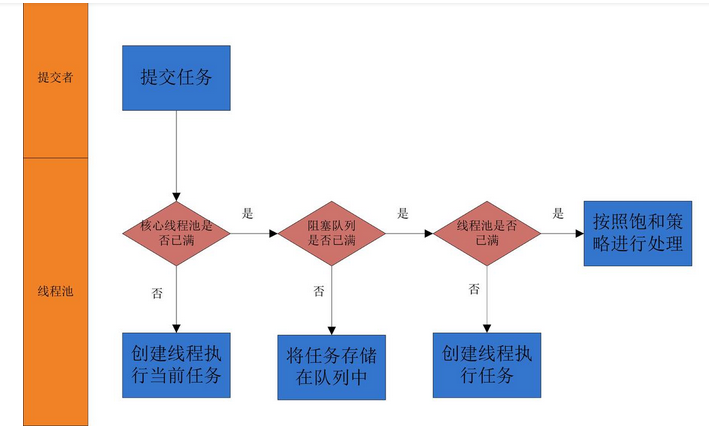
Java申请内存一般有两种，静态内存和动态内存，编译的时候就能确定的内存就是静态内存，内存是固定的，系统一次性分配的，运行时才能确定的内存叫动态内存

1. 标记-清除法：分为两个阶段，标记和清除，标记所有有清除的对象，然后清除，不足是清除后会产生大量的碎片并且效率比较低
2. 复制算法：该算法把内存空间分为两个部分，使用部分正在使用的对象复制到另一块区域并清除使用中的空间
3. 标记整理：整合了标记清除和复制两个算法的优点，先标记所有被引用的对象，第二阶段遍历整个堆，把所有存活对象按顺序压缩到一起
4. 分代收集算法：不同对象的生命周期是不一样的，不同生命周期使用不同的收集方式

年轻代，当Eden被填满时就会产生moinorGC

1. 线程池

线程池的好处：降低资源消耗、提升系统响应速度、提高线程的可管理性



看核心线程池是否都在执行任务，如果有空闲的就将提交的任务放入核心线程池执行任务

判断阻塞队列是否已满，如果未满将刚刚的任务提交到阻塞队列中；如果阻塞队列已满再判断线程池中所有的线程是否在工作，如果没有则创建新的任务，如果有则交给饱和策略进行处理

1. redis集群的几种方式

主从复制、哨兵模式、redis-cluster集群

22.如何实现分布式锁

23.spring的事务传播方式

24.spring bean的生命周期，如何改变生命周期

25.项目的启动加载顺序

1）web项目的启动加载顺序

启动web项目的时候会读取他的web.xml，读取context-param和listener两个节点，紧接着创建servletContext，这个项目的所有部分都共享上下文，创建listener和filter的类实例；最后得出的结论是ServletContext->listener->filter->servlet