## jvm

五大区域

类装载--->堆,方法区,栈区,程序计数器 ,本地方法区 --->本地方法 字节码执行引擎

年轻代,老年代比例 1/3

年轻代分为 Edon survivor0 survivor1 比例为8:1:1

启动一个线程会马上给线程分配一个栈空间和一个程序计数器.

栈区存放栈帧,一个方法就是一个栈帧,内部有局部变量,操作数栈,动态连接和方法出口

本地方法区是调用c,c++的

堆和方法区是全局共享的,其中方法区存的是常量，静态变量+类信息,1.8叫元空间,放在直接内存中.

新建的对象都是通过堆分配内存.所以java的内存管理通常指的堆区内存管理.

edon满一执行minorgc

老年代满了会执行fullgc

gcroot：可作为gcroot的对象 线程栈里的本地变量，静态变量，本地方法栈里的变量

可达分析：gcroot作为起点向下一直找到堆区，找到的就是非垃圾对象，其余标记为垃圾。

垃圾移动到survivor1或者0然后把edon清空，如果超过survivor一般，直接进入老年代。每移动1次分代年龄+1，达到15移动到老年代。先从 edon和s0找非垃圾移动到s1，然后清空edon和s0 下次从edon和s1找非垃圾移动到s1.

jvisualvm jdk自带监控工具

每个订单1kb 每秒产生300个，300kb，放大20倍，再加上订单查询，再放大10倍，

每秒产生60M的对象进入堆。1秒后变垃圾

初始3个G，堆1个G， edon-800M s0-100M s1-100M

每秒产生60M垃圾，13秒左右执行一次minorgc。第13秒产生的对象，因为还活着，被存入了s0结果超过100m一半，直接进入老年代。也就是13秒产生了60M。

130秒产生600M 300秒左右 大概2G老年代就没了。产生一次fullgc

通过Xmn 2048M指定年轻代，这样edon就是1.6G s0 s1都是200M了。这样25秒左右一次minorgc，只有一个50M在年轻代。

## redis

基础类型

string+list+ hash+ set+ zset

SETNX -用于分布式锁 没有就设置，成功返回true 有就返回false

lua脚本 保证原子性的设置和设置超时时间 一起提交。

redisson就是基于lua脚本的，同时有个watchdog一直监测状态，如果没获得锁。自旋。

redisson支持充入锁，以及红锁，并发锁

分布式序列号生成：可以用 INCRBY 生成序列号

INCR可以作为计数器

web集群session共享

当然也可以用雪花算法。

hash槽 crc16（key）%16384

节点比较少可以使用虚拟slot 就是加权hash

redis有一个pipe line管道 可以讲数据放入管道，然后一次执行。

使用的IO多路复用，单线程。

集群：master-slave 哨兵 cluster集群

zookeeper保证CP

c绝对一致性，a可用性，p分区容错。

redis保证ap。

当然使用红锁可以保证cp 但是就保证不了a了 有个投票的过程。

zk主节点叫leader，从节点叫follower

redis master和slave

## mysql索引

使用b+树

btree的基础上横向扩展。

非叶节点不存索引，只存冗余，右边大于左边。

去小数构建上一层冗余。

select global status like '%Innodb\_page\_size%';

大约16384字节 约等 16KB

int类型的是8bit，下一个索引位置约6bit 那么一个值和一个索引约14个bit

16KB约能存放1024个组合。

叶子节点算1KB 一个，因为里面有数据，能存差不多16个索引

1层1024 2层1024\*1024\*16约等 200万，如果超了就是要加索引层级了。

hash索引：范围查找不好查找，但是在 相等值查询上很快。

联合索引：基于b+tree的

先以字段1排序构建B+tree

所以如果以第三个字段直接查询是不会走索引的

## mysql问题

2pc 2阶段事物

一阶段准备,二阶段完成

消息：本地消息表,在表中保存消息情况。可以异步的，通过异步改这个表。轮询这个表，OK就提交，失败回滚。

MQ事务（RocketMQ）：消息队列，基于消息处理事务。

中间件（GTS-阿里）：

**CAP很重要的理论，无法三个都满足。**

C:一致性：数据的一致性，某个节点上的数据更新能否实时的被其它节点知道。

A:可用性：非故障节点在合理时间返回合理相应

P:分区容错性：一个分布式项目必须保证某个节点宕机，整个系统可用

**BASE**

BA:基本可用（也不能保证合理的时间，也许没办法给很好的相应）

S:软性状态，同一个数据的不同副本状态，可以不需要实时一致

E:最终一致性。

**数据库的四大特性：ACID**

A:原子性-不可分割的，事物涉及的若干步骤要么全部完成，要么全不完成。

C:一致性-事物开始前后，数据具有一致性

I:隔离性-在并发环境中，会有不同事物同时操作相同数据,而相互之间并不影响。

D:持久性-只要事物成功了，对数据库所做的更新会永久保持，即便数据库崩溃，恢复后，也要保证提交的结果不会丢失。

## 事务的隔离级别

1， read uncommited 读未提交 可避免更新丢失

原理是，一个事务在对一行数据进行修改时，可以允许另一事务来读取，但不允许另外的事务来修改，这样就不会导致数据同时被二个事务修改，可避免更新丢失。

可能出现脏读。

2，read commited 读提交 ：还没提交的事务不允许其它事务访问，可以避免脏读，不能避免不可重复读

3，repeatable read 重复读：读事务禁止写事务，不会出现同一事务两次读到不同数据的情况

4，序列化：所有事务必须串行执行，因此可以避免一切因为并发引起的问题，效率很低。

默认的隔离级别一般都是2或者3。

oracle默认好像2 mysql默认是3。

## zookeeper

分布式协调框架

leader follower

zookeeper管理的数据更小一点，比redis

主要管理 统一命名服务，状态同步服务器，集群管理，分布式应用配置项的管理

文件系统数据结构，有事件监听机制

文件夹也可以存数据

四种类型：持久化节点，有序节点，临时节点，临时顺序节点。

其中后二个基于会话，会话结束就删除。

事件监听：可以针对所有节点，目录，进行事件监听。只要监听所有客户端都会收到通知

必须删除子目录才能删除父目录

临时节点不能有子节点

配置中心、注册中心watch事件，分布式锁

分布式锁都是在业务上创建顺序节点，每一个节点监听小于他的上一个节点的状态，上个一放弃，检查自己是不是所有里最小的，是就获取锁。

三种角色

leader，Follower，observer

L读写数据，F读数据-L挂了参与选举，O负责读数据不参与选举

ZAB协议 L收到数据不会马上写，记录log，通知F写，F写完通知L，一半+1以上的完成数据存储，才会更新自己，自己才写，这样很慢，但是很安全。

## rocketmq

使用mq就可以消峰，解耦，异步分发数据的目的。

并发转为串行

PHP也可以调用队列

多个系统关注不同的topic主题，订阅自己的。

NameServer是一个无状态的，所有broker都要向每个NameServer上报自己的信息。

Producer本身就是集群状态，没有数据的同步。

Consumer也是没有集群状态，启用多个就可以了。

Broker：这个区分MASTER和SLAVE

master写操作、slave读操作

nameserver 通过Broker name区分组，Broker id区分主从 1主 0从。

每个主题4个队列。

普通消息，有序消息，延时消息（订单超时删除），事物消息（处理异步事物）

生产者消息返回方式：同步返回，异步返回，无返回（oneway）

延迟消息18个级别

1，5，10，30，1m，2，3.。。。10m，20m，30m，1h，2h

事物消息

发送一个半消息-成功执行本地事物，然后发送消息，消息状态更新被客户端消费，失败了调用回查，然后回滚主消息。

## hashmap

1.7数组+链表

1.8 数组+链表+红黑树，速度优化20%左右

使用2的n次幂，作为初始容量，为了扩容，好对key取模，正常取模，位运算2n-1

就是（00001000 -1）00000111 用2的n次幂-1值 &位运算，比取模快多了

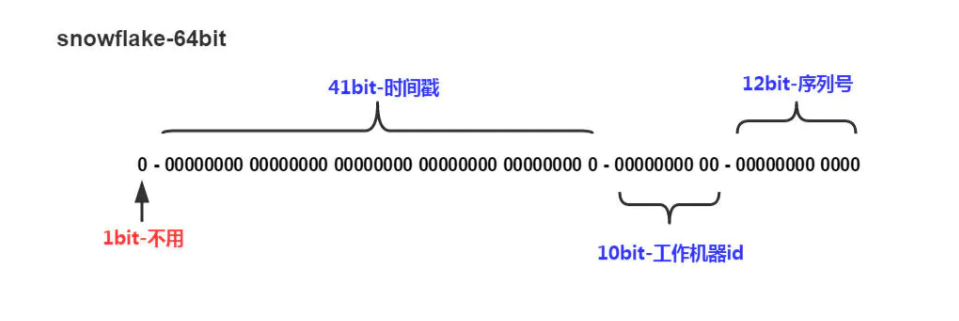
初始0.75 是经过计算的最佳扩容比例。

## spring模式

Spring 框架中用到了哪些设计模式：

* 工厂设计模式 : Spring使用工厂模式通过 BeanFactory、ApplicationContext 创建 bean 对象。
* 代理设计模式 : Spring AOP 功能的实现。
* 单例设计模式 : Spring 中的 Bean 默认都是单例的。
* 模板方法模式 : Spring 中 jdbcTemplate、hibernateTemplate 等以 Template 结尾的对数据库操作的类，它们就使用到了模板模式。
* 包装器设计模式 : 我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。
* 观察者模式: Spring 事件驱动模型就是观察者模式很经典的一个应用。
* 适配器模式 :Spring AOP 的增强或通知(Advice)使用到了适配器模式、spring MVC 中也是用到了适配器模式适配Controller。

## 雪花算法



1毫秒产生212=4096

## kafka

## 类加载的顺序：

父类静态变量 静态语句块

子类的静态变量 静态语句块

父类的普通变量以及语句块

父类的构造方法

子类的普通变量和普通方法

子类的构造器

## 线程池的五种状态：

running 运行中 可以添加线程，也可以处理线程

shutdown 不接收新任务，但可以处理已添加的任务 running-shutdow()-shutdow

stop 不接收 中断正在处理的 running-shutdownow()-top

tidying 所有任务已终止，shutdow状态下线程池为空时 或者stop状态下线程池为空变为tidying

terminated 线程池处于tidying状态，执行完terminated()方法后 变为tidying

## 线程5种状态

创建(New) 就绪(Runnable) 运行状态(Running) 阻塞状态(Blocked) 死亡状态(Dead)

new调用start()进入就绪状态(Runnable)

就绪状态(Runnable) 获取cpu进入到 运行状态(Running)

运行状态(Running)通过sleep()、join()进入阻塞状态，sleep结束或者join结束进入到运行状态(Running)

运行状态(Running)wait阻塞等待 通过notify()和notifyAll()，interrupt() 进入阻塞状态的锁定

运行状态(Running)也可以通过synchronized进入状态锁定。

运行状态(Running)run结束 结束 异常退出进入dead状态

## 反射机制：

java类的编译成字节码的时候，通过JDK工具反射方式访问类字节码的字段，方法。常用spring ioc容器。

反射机制常用的类：

Java.lang.Class;

Java.lang.reflect.Constructor;

Java.lang.reflect.Field;

Java.lang.reflect.Method;

Java.lang.reflect.Modifier;

## 序列化

缺点，不能跨平台

改进 使用jackson

对象转二进制传输

缺点不能跨平台

使用spring的jakson序列化

可以跨平台

--------------------------------------------------

## mybatis执行器三种？

SimpleExecutor：每执行一次update或select，就开启一个Statement对象，用完立刻关闭Statement对象。

ReuseExecutor：执行update或select，以sql作为key查找Statement对象，存在就使用，不存在就创建，用完后，

不关闭Statement对象，而是放置于Map内，供下一次使用。简言之，就是重复使用Statement对象。

BatchExecutor：执行update（没有select，JDBC批处理不支持select），将所有sql都添加到批处理中（addBatch()），

等待统一执行（executeBatch()），它缓存了多个Statement对象，每个Statement对象都是addBatch()完毕后，

等待逐一执行executeBatch()批处理。与JDBC批处理相同。

作用范围：Executor的这些特点，都严格限制在SqlSession生命周期范围内。

Mybatis中如何指定使用哪一种Executor执行器？

答：在Mybatis配置文件中，可以指定默认的ExecutorType执行器类型，也可以手动给DefaultSqlSessionFactory的

创建SqlSession的方法传递ExecutorType类型参数。

--------------------------------------------------

## spring cloud组件

euraka

zuul

ribbon

hystrix

config

jwt

auth2.0

zipkin

--------------------------------------------------

## java如何保证多线程运行安全

lock 和 synchronized

## SQL范式

1、第一范式（1NF）：所有的域都应该是原子性的（不可拆分，不能重复）

2、第二范式（2NF）：属性完全依赖于主键（要求表中所有的列都依赖主键,而不能有任何一列与主键无关，也就是说一个表只描述一件事）

3、第三范式（3NF）：任何非主属性不依赖于其它非主属性(表中每一列只与主键直接相关 而不是间接相关

## 设计模式

Java中的23种设计模式：  
Factory（工厂模式），      Builder（建造模式），       Factory Method（工厂方法模式），  
Prototype（原始模型模式），Singleton（单例模式），    Facade（门面模式），  
Adapter（适配器模式），    Bridge（桥梁模式），        Composite（合成模式），  
Decorator（装饰模式），    Flyweight（享元模式），     Proxy（代理模式），  
Command（命令模式），      Interpreter（解释器模式）， Visitor（访问者模式），  
Iterator（迭代子模式），   Mediator（调停者模式），    Memento（备忘录模式），  
Observer（观察者模式），   State（状态模式），         Strategy（策略模式），  
Template Method（模板方法模式）， Chain Of Responsibleity（责任链模式）

## 多线程

BIO同步阻塞模式,传统模式,blocking InputStream OutputStream

NIO同步非阻塞模式：none blocking IO

1.Buffer缓冲区，相当天BIO里的byte[]数组，缓冲数据，包括ByteBuffer,CharBurrer,ShortBuffer,IntBuffer,LongBuffer,FloatBuffer,DoubleBuffer

2.channel通道，包括selectabelChannel和FileChannel

相当于BIO的Inputstream和OutputStream。

但是区别是Inputstream只可以往里面读，OutputStream只能往外面写。

而Channel既可以读 也可以写。

其中selectabelChannel用于网络通信

FileChannel用于文件读取。

3.selctor多路复用器，也是NIO实现非阻塞模式的关键

AIO-异步的，非阻塞模式。Asynchronous IO.

它的问题是使用比Netty繁琐。

它的设计是不需要selector在服务端轮轮，它是事件驱动的。也就是说客户端来消息，会主动通知我们的服务器，然后服务器进行读写操作。

## Netty

核心：Encoder编码器 Decoder解码器 Handler业务处理器，ByteBuf缓冲区。

1，客户端发送消息通过一个ChannelOutboundHandler去处理，处理完返回给编码器，编码器把消息编码成二进制流，发送，然后给服务端。

服务端先进入Decoder解码器进行解码，然后把消息扔给ChannelnboundHandler去处理。

2，服务端如果需要回的消息，和客户端一样，调用ChannelOutboundHandler去处理，处理完毕调用Encoder编码器编程成二进制流，提交到服务器 进入到Decoder解码器解码成消息，传入客户端的ChannelInboundHandler去处理。

3，其中客户端有个ChannelPipeLine-----管道流水线。

在实际业务中，我们会定义编码器，解码器，ChannelInboundHandler,ChannelOutboundHandler，我们会把所有的这些都方在管道流水线上。这样对消息进行串行化处理。

服务端同样会有管道流水线，ChannelPipeline。

4，在编码解码过程中会用到ByteBuf缓冲器，这个是Netty独有的。

## springcloud alibaba

nacos替代euraka

Sentinel替代hystrix

seata分布式事务

TC,TM, RM 开源服务，tm全局事务发起者和回滚者，RM一个个微服务系统

gateway网关