Java

1 重写（override）与重载（overload）的差别

重写（override）

Class Son extends Father{

@Override

Public void sayHello(){

//一般可以自动生成，然后自己根据需要重写。Father里面有这个hello函数

System.out.println(“hello by ”);

}

}

重写发生在父子之间Son extends Father，有@override注释。声明为final、static、private的方法不能被重写。重写的时候方法的方法名、参数列表、返回类型都需要父子类保持一致，特殊情况是子类中方法返回类型是父类中返回类型的子类，这个时候可以变换返回类型为父类返回类型的子类(JDK1.5 及更早版本返回类型要一样，JDK1.7 及更高版本可以不同)权限上，子必须比父更开放（父为public，子不能声明为protected或者private

重写方法不可以抛出新的检查异常和比被重写方法更宽泛的异常（也就是只能抛出父类方法抛出的异常）

（可以扩展[Java学习笔记13---如何理解“子类重写父类方法时，返回值若为类类型，则必须与父类返回值类型相同或为其子类” - 蝉蝉 - 博客园 (cnblogs.com)](https://www.cnblogs.com/chanchan/p/7796472.html)）

重载（overload）

public class Father {

public static void main(String[] args) {

Father s = new Father();

s.sayHello();

s.sayHello("ABC");

}

public void sayHello(){

System.out.println("hello");

}

public void sayHello(String name){

System.out.println("hello" + " " + name);

}

}

重载函数集的任意两个函数必须具有不同的参数列表，参数顺序、个数、类型不同都可以（形参名字不纳入考虑），仅返回类型不同的两个方法不算重载，会直接报方法名错误。返回值类型是完全自由的。重载是一个类中多态性的一种表现

//方法的区分是靠名字和参数，而不靠返回类型

答：

最大区别就是重载是参数不同的同名方法，重写是子类重写父类同名方法。

重载函数集的任意两个函数必须具有不同的参数列表， 参数顺序、个数、类型不同都可以（形参名字不纳入考虑），仅返回类型不同的两个方法不算重载，会直接报方法名错误。返回值类型是完全自由的。重载是一个类中多态性的一种表现

重写发生在父子之间Son extends Father，有@override注释。声明为final、static、private的方法不能被重写。重写的时候方法的方法名、参数列表、返回类型都需要父子类保持一致，特殊情况是子类中方法返回类型是父类中返回类型的子类，这个时候可以变换返回类型为父类返回类型的子类(JDK1.5 及更早版本返回类型要一样，JDK1.7 及更高版本可以不同) 权限上，子必须比父更开放（父为public，子不能声明为protected或者private

重写方法不可以抛出新的检查异常和比被重写方法更宽泛的异常（也就是只能抛出父类方法抛出的异常）

重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。

2 final finally finalize() 区别（这是初级版本，高级版本在这里：[Java中final、finally、finalize的区别与用法 - smart\_hwt - 博客园 (cnblogs.com)](https://www.cnblogs.com/smart-hwt/p/8257330.html)）

Final：修饰类、变量和方法。表示最终的、不可改变的。

Finally：异常处理的一部分。只用在try/catch语句中，表示希望finally语句块中的代码最后一定被执行（但事实上不一定，系统退出、抛出异常未执行到finally都不会运行。Try运行的代码出错或者跳出方法就会执行finally）

Finalize()在Java.lang.Object里定义的，这里的finalize()方法什么都不做，对象被垃圾回收机制回收之前finalize()方法才会被调用。特殊情况：可以重写finalize方法，当对象被回收的时候释放一些资源。注意要调用super.finalize()。

3jdk和jre区别

Jdk：java develop kit（java开发工具包）就是java编写代码所需的开发工具包、jre、java代码编译器javac、监控工具jconsole（检测死锁这些）、分析工具jvisualvm（能够监控线程，内存情况，查看方法的CPU时间和内存中的对象）这些

Jre：java runtime environment（java运行环境）包括java虚拟机、java基础类库

4==与equals()区别

==是运算符，equals()是方法，数组或者其他对象一般用equals方法，不然比较的是内存地址

==：遇到基本类型比较值是否相等；遇到引用类型的话就去比较内存地址是否相等；没有父子关系的两个对象没法比较（可以用来比较enum实例，编译器会自动提供equals（）和hashCode（）方法）

Equals()：JDK中的类一般都把equals（）重写成比较内容的了。自己写的类没有重写equals（）的话就会调用父类（默认Object类）的equals（）方法，Object的equals（）方法判断的是两个对象的引用指向的是不是同一个对象（使用this==obj）。如果要重写对象的equals（）方法的话，一般需要重写hashcode方法（[重写hashcode和equals方法\_简乐君-CSDN博客\_重写hashcode](https://blog.csdn.net/u012557538/article/details/89861552)）

5浅拷贝与深拷贝的区别

区别在于复不复制堆内存中的对象。都要复制基本类型的属性、引用类型的属性还有栈中的变量。浅拷贝复制的变量指向堆内存中的对象的指针，母本和子本指向同一段堆内存中的引用对象；深拷贝因为子本拷贝了新的堆内存中的对象，它的母本和子本的变量指向堆内存中的对象的指针分别指向各自的不同地址的引用对象。

6接口和抽象类有什么区别

a抽象类可以有构造方法；接口类不能有构造方法

（构造方法是一个比较特殊的方法，通过构造方法可以完成对象的创建，以及实例变量的初始化。换句话说：构造方法是用来创建对象，并且同时给对象的属性赋值。注意：实例变量没有手动赋值的时候，系统会赋默认值。）

b抽象类中可以有普通成员变量；接口中没有普通成员变量（普通成员变量是每个对象单独享用的，和对象共存亡，又名局部成员变量）

c抽象类中可包含非抽象普通方法；jdk1.8以前接口中所有方法默认为抽象的，jdk1.8开始方法可以有default实现和static方法。

d抽象类中的抽象方法的访问权限可以是public、protected和default；接口中的抽象方法只能是public类型，且默认为public abstract类型

e抽象类中可以包含静态方法；jdk1.8前接口中不能包含静态方法，jdk1.8及以后可以包含已实现的静态方法。

f抽象类和接口中都可包含静态成员变量，抽象类中静态成员变量可以是任意访问权限；接口中变量默认且只能是public static final类型

g一个类可以实现多个接口（用逗号隔开），但只能继承一个抽象类

h接口不可以实现接口，但是可以继承接口，接口可以多个（用逗号隔开）

抽象类：

abstract class Role{

private String name; //私有属性

public String getName(){

return this.name;

}

public abstract void play(); //抽象方法

public Role(String name){

this.name = name;

}

}

class Employee extends Role{ //继承

private double salary; //私有属性

public double getSalary(){

return this.salary;

}

public void setSalary(double salary){

this.salary = salary;

}

public Employee(String name,double salary){

super(name);

**//super用于多重继承中调用父类方法**

this.salary = salary;

}

final void sing(){

System.out.println("name:"+this.getName()+

"\nsalary:"+this.getSalary());

}

}

class Manager extends Employee{ //继承

final String vehicle;

public Manager(String name,double salary,String vehicle){

super(name,salary);

this.vehicle = "自行车";

}

}

Java 有具体类和抽象类。其中抽象类是没有描述具体对象的类。Class前面要写abstract

接口：

package com.wz.interfacedemo;

interface A{//定义一个接口A

public static final String MSG = "hello";//全局常量

public abstract void print();//抽象方法

}

interface B{//定义一个接口B

public abstract void get();

}

class X implements A,B{//X类实现了A、B两个接口

@Override

public void print(){

System.out.println("接口A的抽象方法print（）");

}

@Override

public void get(){

System.out.println("接口B的抽象方法get()");

}

}

public class TestDemo {

public static void main(String[]args){

X x = new X();//实例化子类对象

A a = x;//向上转型

B b = x;//向上转型

a.print();

b.get();

}

}

官方解释：Java接口是一系列方法的声明，是一些方法特征的集合，一个接口只有方法的特征没有方法的实现，因此这些方法可以在不同的地方被不同的类实现，而这些实现可以具有不同的行为（功能）。

我的解释：接口可以理解为一种特殊的类，里面全部是由全局常量和公共的抽象方法所组成。接口是解决Java无法使用多继承的一种手段，但是接口在实际中更多的作用是制定标准的。或者我们可以直接把接口理解为100%的抽象类，既接口中的方法必须全部是抽象方法。（JDK1.8之前可以这样理解）

接口只有方法标识符，而没有方法体

抽象方法只能存在于抽象类或者接口中，但抽象类中却能存在非抽象方法，即有方法体的方法。接口是百分之百的抽象类

为啥用接口：因为Java不像C++一样支持多继承，所以Java可以通过实现接口来弥补这个局限

7final与static的区别

Static可以修饰类的代码块、final不可以

Static不可以修饰方法内的局部变量，final可以

（都可以修饰类、方法、成员变量

都不能用于修饰构造方法）

Static：

Static修饰表示静态或全局，被修饰的属性和方法属于类，可以用类名.静态属性/方法名 访问

Static修饰的代码块表示静态代码块，当java虚拟机（JVM）加载类时，就会执行该代码块，仅一次

Static修饰的变量可重新赋值

Static方法中不能使用this和super关键字

Static方法必须被实现，不能是抽象的（abstract）

Static方法不能被重写

Final：

Final修饰表示常量、被创建了就不能改变了

标记的成员变量必须在声明的同时或者该类的构造方法中赋值，不可以重新赋值

不能被子类重写

不能被继承，没有子类，final类中的方法默认是final的

import java.util.Random;

public class TestStaticFinal {

public static void main(String[] args) {

StaticFinal sf1 = new StaticFinal();

StaticFinal sf2 = new StaticFinal();

System.out.println(sf1.fValue == sf2.fValue);//打印false

System.out.println(sf1.sValue == sf2.sValue);//打印true

}

}

class StaticFinal {

final int fValue = new Random().nextInt();

static int sValue = new Random().nextInt();

}

8 hashmap的原理（java）

（？为什么要用红黑树）

常用参数）

DEFAULT\_INITIAL\_CAPACITY **Table数组的初始化长度**： 1 << 4 2^4=16 (？为什么要是2的n次方)

MAXIMUM\_CAPACITY **Table数组的最大长度** 1<<30 2^30=1073741824

DEFAULT\_LOAD\_FACTOR **负载因子**：默认值为0.75。 当元素的总个数>当前数组的长度\*负载因子。数组会进行扩容，扩容为原来的两倍（为什么是两倍：因为数组用二进制存储，延申一个就是两倍的可存储量）

TREEIFY\_THRESHOLD **链表树化阈值**：默认值为8 。表示在一个node（Table）节点下的值的个数大于8的时候，将会把链表转换成红黑树。

UNTREEIFY\_THRESHOLD **红黑树链化阈值**：默认值为6。表示在扩容期间，单个Node节点下的红黑树结点的个数小于6的时间会将红黑树转化成链表。

MIN\_TREEIFY\_CAPACITY = 64 **最小树化阈值**，当Table所有元素超过该值，才会进行树化（为了防止前期阶段频繁扩容和树化过程冲突）。

【有table数组初始化长度、最大长度、负载因子以及链表树化、红黑树链化等各种阈值】

hashCode

①主要是用于查找快捷性，如Hashtable、HashMap等，hashCode是用来在散列存储结构中确定对象的存储地址的。

②如果两个对象相同，就是适用于equals（java.lang.Object）方法，那么这两个对象的hashCode一定要相同

③如果对象的equals方法被重写，那么对象的hashCode也尽量重写，并且产生hashCode使用的对象，一定要和equals方法中使用的一致，否则就会违反上面提到的第二点

④两个对象的hashCode相同，并不一定表示两个对象就相同，也就是不一定适用于equals（java.lang.Object）方法，只能够说明这两个对象在散列存储结构中，如Hashtable（存于一个篮子中）

hashCode用于查找使用的，equals用于比较两个对象是否相等。

HashMap是基于哈希表的Map接口的非同步实现。提供所有可选的映射操作，并允许使用null值和null键。不保证映射的顺序，也不保证映射顺序永远不变。

HashMap是“链表散列”的数据结构，也就是数组和链表的结合体。Table数组结合Entry链表。（1.8加入红黑树）K和V代表一个key-value键值对，有一个指向下一个元素的引用next，以及自身的hash值。

根据hash值得到元素在数组中的位置（下标号），如果数组该位置上已有其他元素了，那么这个位置上的元素将以链表的形式存放，新加入的放在链头，最先加入的放在链尾。如果数组该位置上没有元素就直接放在数组里的该位置上

（同义）当程序试图将一个key-value对放入HashMap中时，程序首先根据该 key的 hashCode() 返回值决定该 Entry 的存储位置：如果两个 Entry 的 key 的 hashCode() 返回值相同，那它们的存储位置相同。如果这两个 Entry 的 key 通过 equals 比较返回 true，新添加 Entry 的 value 将覆盖集合中原有 Entry的 value，但key不会覆盖。如果这两个 Entry 的 key 通过 equals 比较返回 false，新添加的 Entry 将与集合中原有 Entry 形成 Entry 链，而且新添加的 Entry 位于 Entry 链的头部——具体说明继续看 addEntry() 方法的说明。（同义）

hash(int h)方法根据key的hashCode重新计算一次散列。此算法加入了高位计算，防止低位不变高位变化时，造成hash冲突。

HashMap中要找某个元素需要根据key的hash值来求得对应数组中的位置。从HashMap中get元素时，首先计算key的hashCode，找到数组中对应位置的某一元素，然后通过key的equals方法在对应位置的链表中找到需要的元素。

HashMap 在底层将 key-value 当成一个整体进行处理，这个整体就是一个 Entry 对象。

（总结）HashMap 底层采用一个 Entry[] 数组来保存所有的 key-value 对，当需要存储一个 Entry 对象时，会根据hash算法来决定其在数组中的存储位置，在根据equals方法决定其在该数组位置上的链表中的存储位置；当需要取出一个Entry时，也会根据hash算法找到其在数组中的存储位置，再根据equals方法从该位置上的链表中取出该Entry。

**HashMap—resize**

为了在数据量越来越多碰撞几率越来越高的时候提高查询效率，对hashmap数组进行扩容。原数组中数据需要重新计算其在新数组中的位置并放进去。

当hashmap中的元素个数超过数组大小\*loadFactor时，就会进行数组扩容，loadFactor的默认值为0.75，当hashmap中元素个数超过该值的时候，就把数组的大小扩展为2两倍，然后重新计算每个元素在数组中的位置（是一个非常消耗性能的操作），所以如果我们已知hashmap中元素的个数，那么预设元素的个数能够有效的提高hashmap的性能如果已知元素个数，就可以预设大小（让容量\*0.75>已知元素个数）。

扩充：[HashMap原理详解，看不懂算我输（附面试题） - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/127147909)

8.5重写hashcode

[重写hashcode和equals方法\_简乐君-CSDN博客\_重写hashcode](https://blog.csdn.net/u012557538/article/details/89861552)

9hash冲突的时候怎么办

构造性能良好的哈希函数可以减少冲突

**a开放定址法（再散列法）**

当关键字key的哈希地址p=H（key）出现冲突时，以p为基础，产生另一个哈希地址p1，若p1仍冲突，再以p为基础产生另一个哈希地址p2，直到找到一个不冲突的哈希地址pi并将元素存入其中【Hi=（H（key）+di）% m |i=1，2，…，n】

H（key）是哈希函数，m表长，di增量序列

di有（①线性探测再散列 di=1，2，3…m-1 冲突发生时顺序查看下一单元，找出空单元或查遍全表。

②二次探测再散列 di=12，-12，22，-22，…，k2,-k2【k<=m/2】优点是灵活，缺点是探测不完全

③伪随机探测再散列 随机查看一个单元，为空则填入）

**b再哈希法**

Hi=RH1（key）i=1，2，。。。，k 当哈希地址Hi=RH1（key）发生冲突，再计算Hi=RH2（key）。。。直到不再发生冲突。优点不易产生聚集，缺点增加了计算时间

**c链地址法**

将哈希地址为i的元素构成一个同义词链的单链表，将单链表的头指针存在哈希表的第i个单元中，因而查找、插入和删除主要在同义词链中进行。链地址法适用于经常进行插入和删除的情况。（hashmap采用的这个方法，如果元素多余总容量75%则扩展哈希表）

**d建立公共溢出区**

将哈希表分为基本表和溢出表两部分，凡是和基本表发生冲突的元素，一律填入溢出表

10sql如何优化，优化索引。

[MySQL如何进行慢SQL优化？ - Java面试题 (javanav.com)](https://www.javanav.com/interview/7b1d6961b1e344fa9b6ea74819d4f417.html)

通过慢查询日志查哪些SQL执行效率低

用explain获取低效率SQL执行计划

结合SQL与执行计划，进行分析与优化

引起sql查询慢的原因与办法：

a没有索引。解法：

根据where 和order by使用比较频繁的字段创作索引（提高查询效率）

索引过多会导致占用存储空间变大；insert、update变慢。所以索引不要过多，单表最好不超过6个。

删除未使用索引

b索引未生效。解决办法：

避免在where子句中对字段进行null值判断，创建表默认值是NULL。尽量使用NOT NULL，或使用特殊值，如0、-1

避免在where子句里面使用!=或<>操作符，MySQL只有对以下操作符才使用索引：<、<=、=、>、>=、BETWEEN、IN、非 % 开头的LIKE

避免在where子句中使用 or 来连接条件， 可以使用UNION进行连接

能用union all就不用union，union过滤重复数据要耗费更多的CPU资源

避免部分like查询，如‘%ConstXiong%’（？可以的话我想查一查like查询是啥，后面又是啥）

避免在索引列上使用计算、函数

慎用in和not in ，能用between不用in

避免使用selet子句

c单表数据量太大。解决办法：

分页查询（在索引上完成排序分页操作、借助主键进行关联）

单表数据过大，进行分库分表

考虑使用非关系型数据库提高查询效率

全文索引场景较多，考虑使用ElasticSearch、solr

提升性能的技巧：

尽量使用数字型字段

只需要一行数据时使用limit1

索引尽量选择较小的列

不需要的数据在GROUP BY之前过滤掉

大部分时候exists、not exists比in、not in效率高（例外：子查询是小表情况下，in效率高于exists）

不确定长度的字符串字段使用varchar/nvarchar，如用char/nchar定长存储会带来空间浪费

不要使用select\*，去除不需要的字段查询

避免一次性查询过大的数据量

使用表别名，减少多表关联解析时间

多表join最好不超过5个，视图嵌套最好不超过2个（？join是什么）

Or条件查询可以拆分成UNION多个查询

count(1)比count(\*)有效

判断是否存在数据使用exists而非count，count用来获取数据行数

10.5百万条数据如何优化

a表容量问题

对于过多数据量需提前考虑分表分库。单表设计在200-500万最佳；优化做好可以让单表数据到一两千万还保持良好的性能。（出现单表几千万上亿的数据量是设计出现问题）

b总页数的问题

页面不需要显示总页数，仅显示附近页码 就可以避免单表总行数的查询

**需要显示总页数：**MySQL的MyISAM引擎把一个表的总行数记录在磁盘中，查询count(\*)可以直接返回；（InnoDB引擎是一行行读出来累加计数，大数据量时性能堪忧 || 大几秒甚至几十秒都有可能）。所以一般情况下MyISAM的总行数查询速度是比InnoDB快的，（仅限于不带where条件时I比M更快）。MyISAM还有个硬伤，不支持事务。

**既支持事务又快速的查出总数：**用InnoDB引擎新建一张表记录业务表的总数，新增、删除各自在同一事物中增减总行数然后查询，保证事务的一致性和隔离性。这里更新总行数要借助分布式锁

c具体的SQL优化

新增表记录业务表的总数也无法彻底解决带查询条件的总行数查询慢的问题。只能借助具体的SQL优化。

不带条件+自增id字段连续的理想条件：通过pageNo和pageSize算出id的起始与结束值

where id >= ? and id <?

where id between

where id >= ? limit 10

Δ带查询条件+主键id不连续：用limit分页有查询耗时与起始记录的位置成正比的问题，所以不能直接使用。

可以根据主键进行关联查询（？什么是关联查询）

select \* from table t1

join (select id from table where condition limit 10) t2

on t1.id = t2.id

order by t1.id asc

其中condition是包含索引的查询条件，使用id字段进行具体信息的关联回查。查询条件condition中索引是否生效对性能影响也大

索引没生效：

组合索引的【最左前缀】原则

or的使用可能导致索引未生效，可使用union all代替

like查询以 % 开头

对null值判断

使用!=或<>操作符

索引列上使用计算、函数

d其他解法

继续优化数据库配置

提升数据库服务器硬件性能

引入大数据组件

引入大型商业数据库或者非关系型数据库解决大表问题

注：

MySQL 大表分页问题，一般效果比较好的是，使用记录页面最大最小 ID 或统计表优化 count 查询。

从面试回答问题的角度看，如果能结合索引的实现，比如 InnoDB 的索引使用 B+ 树，子查询中索引如何生效与失效，说清楚问题的本质是就是用空间去换取查询时间，把问题提高到计算机原理（I/O、CPU 之间的权衡）、数据结构与算法的层面去阐述，肯定会加分不少。

（？这里又是一个未解之谜）

11静态变量和实例变量的区别（其实就是static变量和普通变量的区别

Java中被static修饰的方法，就不属于对象，使用的时候不会初始化对象

静态变量前面要加static关键字，而实例变量前面不加

实例变量属于某个对象的属性，必须创建了实例对象才可以给其中的实例变量分配空间，此时才可以使用这个实例变量。

静态变量看起来像全局变量，但java没有全局变量。不属于某个实例对象，而是属于类（又名类变量），只要程序加载了类的字节码，不用创造任何实例对象，静态变量就会被分配一份存储区域，不会再次被初始化

实例变量必须创建对象后才可以通过这个对象来使用，静态变量可以直接使用类名来引用

12spring boot概念

--如何把类交给容器管理（？。。。）

a添加配置文件，配置文件里面添加方法，并加@Bean注释

调用对象的时候可以注释@Autowired或者@Resource直接注入

b **@Component、Controller、Service、Repository**

是四个主要的容器添加方式

@Component注解在一个类上，表示将此类标记为Spring容器中的一个Bean

@Controller控制层类，@Service 业务层类，@Repository 持久层类，@Component 无法归类到前3种时就称为组件

Controller、Service、 Repository本质是Component

只是给开发者看的，对Spring而言 都是Component

@Service注解在类名上面添加

c **@Import方法**

直接装配，可用逗号隔开添加多个

自动从类的无参构造函数创建一个实例注册到IOC容器中

创建的实例在IOC容器中默认的id名为类的全限定名，如User类是：com.atguigu.bean.User

**proxyBeanMethods**

Lite模式：（类组件之间无依赖关系）proxyBeanMethods = false，加速容器启动过程，减少判断

Full模式：（类组件之间有依赖关系）proxyBeanMethods = true，方法会调用得到之前单实例组件

多次调用@Bean方法都是单实例，组件依赖必为full

**@Configuration 常用的配置类**



导入容器后用@Autowired就交给容器创建对象了

@Conditional

它简化spring的大量重复工作（spring.xml的配置，spring和第三框机集成需要的配置，重复jar的添加）通过特定装置进行项目的配置，不需要开发人员定义样板化的配置，实现项目的快速开发。（总之就是简化编码、配置、部署、监控）

特点：

1.可以创建独立的spring应用程序，基于它的maven，可以创建可执行的jar、war。

2.内嵌tomcat或jetty等servlet容器

3.自动配置spring容器，（自动注入，自动扫描等）

4.无代码生成，不需要xml配置

5.提供自动配置的“starter”项目对象模型（POMS）以简化maven配置

--spring boot的容器对象什么时候销毁

框架自动维护的话，是程序运行停止的时候销毁对象

13jvm内存模型

**0x01:程序计数器（Program Counter Register）**

程序计数器是一块较小的内存空间，它可以看作是当前线程所执行的字节码的行号指示器。在虚拟机概念模型里（概念模型，各种虚拟机可能会通过一些更高效的方式实现），字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令：分支、跳转、循环、异常处理、线程恢复等基础操作都会依赖这个计数器来完成。每个线程都有独立的程序计数器，用来在线程切换后能恢复到正确的执行位置，各条线程之间的计数器互不影响，独立存储。所以是一个“线程私有”的内存区域。从内存区域是唯一一个在JVM规范中没有规定任何OutOfMemoryError情况的区域。

**0x02:虚拟机栈（VM Stack）**

JVM栈是线程私有的内存区域，描述的是java方法执行的内存模型，每个方法执行的同时都会创建一个栈帧（Stack Frame）用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每个方法从调用直至完成的过程，都对应着一个栈帧从入栈到出栈的过程。每当一个方法执行完成时，该栈帧就会弹出栈桢的元素作为这个方法的返回值，并且清除这个栈帧，Java栈的栈顶就是目前正在执行的活动栈，也就是当前正在执行的方法。就像是组成动画的一帧一帧的图片，方法的调用过程也是栈帧切换来产生结果。

局部变量表存放了编译器可知的各种基本数据类型（int、shrot、byte、char、double、float、long、boolean）、对象引用（reference类型，它不等同于对象本身，可能是一个指向对象起始地址的引用指针，也可能是指向一个代表对象的句柄或其他与此对象相关的位置）和returnAddress类型（指向了一跳字节码指令的地址）。

//在JVM规范中，对这个区域

**0x03:本地方法栈（Native Method Stack）**

本地方法栈和虚拟机栈所发挥的作用是很相似的，它们之间的区别不过是虚拟机栈为虚拟机执行Java方法（字节码）服务，而本地方法则为虚拟机使用到的Native方法服务。Sun HotSpot直接就把本地方法栈和虚拟栈合二为一。本地方法栈也会抛出StackOverflowError和OutOfMemoryError异常

**0x04:堆（Heap）**

是OOM故障最主要的发源地（？OOM故障是什么），它存储着几乎所有的实例对象，堆由垃圾收集器自动回收，堆区由各子线程共享使用；通常它占用的空间是所有内存区域中最大的，但如果无节制的创建大量对象也容易消耗完所有空间；堆的内存空间既可以固定大小，也可以运行时动态的调整，通过参数-Xms设定初始值、-Xmx设定最大值。

**0x05:方法区（Method Area）**

是所有线程共享的内存区域，用来存储已被虚拟机加载的类信息、常量、静态变量、JIT（just in time，即时编程技术）编译后的代码等数据。运行时常量池是方法区的一部分，用于存放编译期间生成的各种字面常量和符号引用。

通过反射获取到的类型、方法名、字段名称、访问修饰符等信息就是从方法区获取到的。在使用到CGLib对类进行增强时，增强的类越多，就需要越大的方法区类存储动态生成的class信息，当存放方法区数据的内存溢出时，会报OutOfMemoryError异常。在jdk1.8中也就是metaspace内存溢出，可通过参数JVM参数-XX:MetaspaceSize和-XX:MaxMetaspaceSize设置Metaspace的空间大小。Jdk1.8后方法区（Method Area）被元空间（Metaspace）代替。

[Java虚拟机—Java8内存模型JVM（整理版） - 牧梦者 - 博客园 (cnblogs.com)](https://www.cnblogs.com/swordfall/p/10723938.html)

14jvm调优

一般出性能问题（TPS上不去，内存撑爆了），先监控一下程序的日志和性能数据，

调优方式：

①记录好日志；（异常日志）

②对程序做好性能监控；（调用链路、耗时接口、耗时sql）

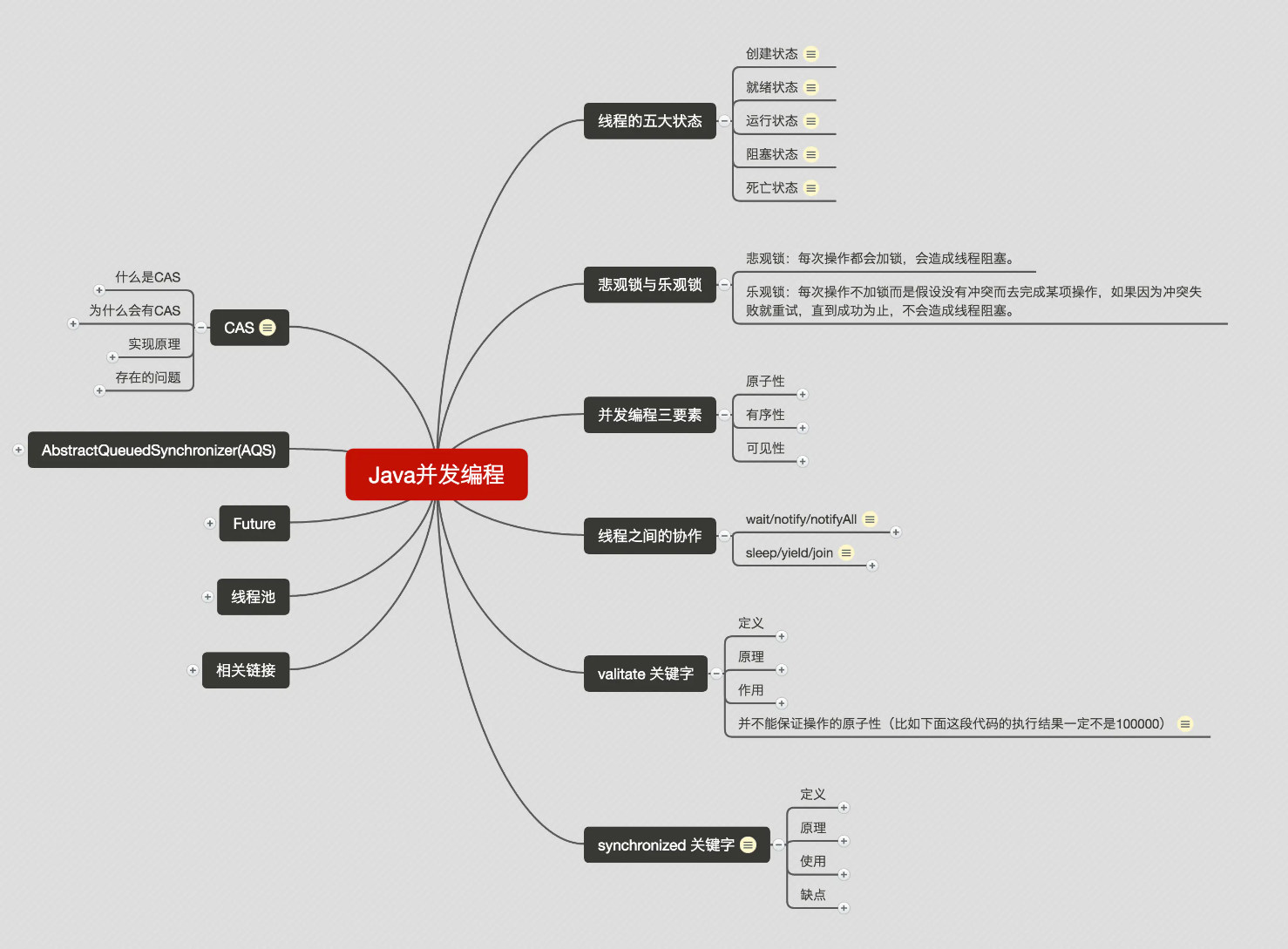
③根据日志和性能监控数据修改程序；

④使用专业工具通过不同的JVM参数进行压测并获得最佳配置（JVM内存镜像分析、做出优化方案）（？[如何做压测？-阿里云开发者社区 (aliyun.com)](https://developer.aliyun.com/article/779380) 一般不用做第四步）

一般用不上，比起jvm性能，更重要的是提高代码质量

[深入浅出JVM调优，看完你就懂\_Javazhoumou的博客-CSDN博客\_jvm调优](https://blog.csdn.net/Javazhoumou/article/details/99298624)

15并发编程



Ⅰ并发编程三要素

①原子性

一个或多个操作，要么全部执行成功要么全部执行失败。

②有序性

程序执行的顺序按照代码的先后顺序执行（处理器可能会对指令进行重排序）

③可见性

当多线程访问同一个变量时，如果其中一个线程对其做了修改，其他线程能立即获取到最新的值。

Ⅱ线程的五大状态

①创建状态

用new操作符创建一个线程的时候

②就绪状态

调用start方法，处于就绪状态的线程并不一定会马上执行run方法，haixuyaodengdaiCPU调度

③运行状态

CPU开始调度线程，并开始执行run方法

④阻塞状态

线程的执行过程中由于一些原因进入阻塞状态

比如：调用sleep方法、尝试去得到一个锁

⑤死亡状态

run方法执行完或者执行过程中遇到了一个异常

Ⅲ悲观锁与乐观锁

悲观锁：每次操作都会加锁，会造成线程阻塞。

乐观锁：每次操作不加锁而是假设没有冲突而去完成某项操作，如果因为冲突失败就重试，直到成功为止，不会造成线程阻塞。

Ⅳ线程之间的协作

wait/notify/notifyAll

这一组是Object类的方法（这三个方法都必须在同步的范围内调用）

①wait

阻塞当前进程，直到notify或者notifyAll来唤醒

三种调用方式：

1. wait() 必须用notify或者notifyAll来唤醒
2. wait(long timeout)规定时间内没有被notify/notifyAll唤醒就会自动唤醒。
3. wait(long timeout, int nanos)等待前面的（timeout）毫秒加上后面的（nanos）纳秒。两个参数都为0则等同于A方法。

当前线程必须是此对象的监视器所有者，否则发生IllegalMonitorStateException异常

如果当前线程在等待之前或在等待时被任何线程中断，则抛出InterruptedException异常

如果传递的参数不合法或nanos不在0-999999范围内，则抛出IllegalArgumentException异常

②notify

只唤醒一个wait进程

③notifyAll

唤醒全部处于wait的进程

这三个是Thread类方法

④sleep

让当前线程暂停指定时间，只是让出CPU的使用权，并不释放锁

⑤yield

暂停当前线程的执行，即当前CPU使用权，让其他线程有机会执行，不可指定时间。

会让当前线程从运行状态转变为就绪状态。很少使用。

⑥join

等待调用join方法的线程执行结束才执行后面的代码

调用一定要在start方法之后

使用场景：当父线程需要等待子线程执行结束才执行后面内容，或者需要某个子线程的执行结果才会用到join方法

Ⅴ关键字valitate

Java编程语言允许线程访问共享变量，为了确保共享变量能被准确和一致的更新，线程应该确保通过排他锁单独获得这个变量。volatile在某些情况下比锁更加方便。字段被声明为volatile，java线程内存模型确保所有线程看到这个变量的值是一致的。

Validate是轻量级synchronized，不会引起线程上下文的切换和调度，执行开销更小

（原理

①使用volitate修饰的变量在汇编阶段，会多出一条lock前缀指令

②它确保指令重排序时不会把其后面的指令排到内存屏障之前的位置，也不会把前面的指令排到内存屏障的后面；即执行到内存屏障这句指令时，前面的操作已经全部完成

③它会强制将对缓存的修改操作立即写入主存

④如果是写操作，它会导致其他CPU里缓存了该内存地址的数据无效）

作用：

内存可见性

多线程操作的时候，一个线程修改了一个变量的值，其他线程能立即看到修改后的值

防止重排序

即程序的执行顺序按照代码的顺序执行（处理器为了提高代码的执行效率可能会对代码进行重排序）

并不能保证操作的原子性（？代码样例。。。）

Ⅵ synchronized关键字

确保线程互斥的访问同步代码。

是JVM实现的一种锁，其中锁的获取和释放分别是monitorenter和monitorexit指令，该锁在实现上分为了偏向锁、轻量级锁和重量级锁，其中偏向锁在Java1.6是默认开启的，轻量级锁在多线程竞争的情况下会膨胀成重量级锁，有关锁的数据都保存在对象头中

（原理

加了synchronized关键字的代码段，生成的字节码文件会多出monitorenter和monitorexit两条指令（利用javap –verbose 字节码文件可看到关于两条指令的文档））

加了synchronized关键字的方法，生成的字节码文件中会多一个ACC\_SYNCHRONIZED标志位，当方法调用时，调用指令将会检查方法的ACC\_SYNCHRONIZED访问标志是否被设置，如果设置了，执行线程将先获取monitor，获取成功之后才能执行方法体，方法执行完成后再释放monitor。在方法执行期间，其他任何线程都无法再获得同一个monitor对象。其本质上没有区别，只是方法的同步是一种隐式的方式来实现，无需通过字节码来完成。

（使用

①修饰普通方法

同步对象是实例对象

②修饰静态方法

同步对象是类本身

③修饰代码块

可以自己设置同步对象

（缺点

会让没有得到锁的资源进入Block状态，争夺到资源之后又转为Running状态，这个过程涉及到操作系统用户模式和内核模式的切换，代价比较高。Java1.6为synchronized做了优化，增加了从偏向锁到轻量级锁再到重量级锁的过度，但是在最终转变为重量级锁之后 性能仍然较低。

Ⅶ CAS

AtomicBoolean，AtomicInteger，AtomicLong以及Lock相关类等底层就是用CAS实现的，在一定程度上性能比synchronized更高。

（定义

CAS全称是Compare And Swap，即比较替换，是实现并发应用到的一种技术。操作包含三个操作数—内存位置（V）、预期原值（A）和新值（B）。如果内存位置的值与预期原值相匹配，那么处理器会自动将该位置值更新为新值。否则处理器不做任何操作。

（目的

弥补synchronized保证同步时存在的问题

Synchronized是一种悲观锁，在使用上会造成一定的性能问题。在多线程竞争下，加锁、释放锁会导致比较多的上下文切换和调度延时，引起性能问题。一个线程持有锁会导致其它所有需要此锁的线程挂起。

（原理

Java不能直接访问操作系统底层，是通过native方法（JNI--Java本地接口）来访问。CAS底层通过Unsafe类实现原子性操作。

（问题

①ABA问题

线程ABC，AC先执行，导致B获得的参数并非设计时所期望的参数。

解决方法：

给变量加版本号，在比较时不仅比较当前值，还需要比较当前变量版本号。（Java中AtomicStampedReference解决了这个问题）

②循环时间长开销大

在并发量比较高的情况下，如果许多线程反复尝试更新某一个变量，却又一直更新不成功，循环往复会给CPU带来很大的压力

CAS只能保证一个共享变量的原子操作

Ⅷ AbstractQueuedSynchronizer（AQS）

AQS抽象的队列式同步器，是一种基于状态（state）的链表管理方式。State是用CAS去修改的。它是java.util.concurrent包中最重要的基石。ReentrantLock、CountDownLatcher、Semaphore实现的原理就是基于AQS。

[Java并发之AQS详解 - waterystone - 博客园 (cnblogs.com)](https://www.cnblogs.com/waterystone/p/4920797.html)

Ⅸ Future

并发编程一般使用Runable去执行异步任务，然而这样做我们是不能拿到异步任务的返回值的，但是使用Future就可以。使用Future只需要把Runable换成Future Task即可。

Ⅹ线程池

如果我们使用线程的时候去创建另一个线程，虽然简单但是会导致并发的线程数量很多，而且每个线程都是执行很短时间的任务就结束，这样频繁创建和销毁线程会浪费时间，大大降低系统效率。线程池通过复用可以大大减少线程频繁创建与销毁带来的性能上的损耗。

Java中线程池的实现类ThreadPoolExecutor，关键参数：

①corePoolSize：**核心线程数**即一直保留在线程池中的线程数量，即使处于闲置状态也不会被销毁。要设置allowCoreThreadTimeOut为true才会被销毁

②maximumPoolSize：线程池中允许存在的**最大线程数**

③keepAliveTime：非核心线程允许的最大闲置时间，超过这个时间就会本地销毁。

④workQueue：用于**存放任务的队列**

⑤SynchronousQueue：这个队列会让新添加的任务立即得到执行，如果线程池中所有的线程都在执行，那么就会去创建一个新的线程去执行这个任务。当使用这个队列的时候maximumPoolSizes一般都会设置一个最大值Integer.MAX\_VALUE

⑥LinkedBlockingQueue：这个队列是一个无界队列。有多少任务来就执行多少，如果线程池中线程小于corePoolSize，我们创建一个新的线程去执行这个任务，如果线程池中线程数等于corePoolSize，就会将任务放入队列中等待，由于队列大小没有限制所以也被称为无界队列。当使用这个队列的时候maximumPoolSizes不生效（线程池中线程数量不会超过corePoolSize），所以一般都会设置为0.

⑦ArrayBlockingQueue：这是一个有界队列。可以设置队列的最大容量。当线程池中线程数大于或者等于maximumPoolSizes的时候，就会把任务放到这个队列中，当前队列中任务数大于队列最大容量时就会丢弃掉该任务，交由RejectedExecutionHandler处理。

16 tomcat配置方法

1)直接把Web项目放在webapps下，Tomcat会自动部署  
2)在server.xml配置<Context>节点,并设置相关属性  
3)通过Catalina来进行配置:进入到conf\Catalina\localhost文件下,创建一个xmI文件,该文件的名字就是站点的名字。

编写XML的方式来进行设置（mybatis、tomcat都是在XML里类似三段式的样子进行配置的）

17sql锁

17.5索引

18实例对象（即对象属性，field）

Java中对象就feild和method（属性和方法

18面向对象三大基本特征

封装、继承、多态

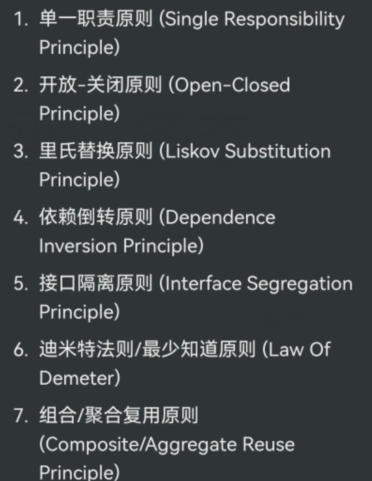
1bfs、dfs

2多服务器登陆验证

3微服务

https://www.jianshu.com/p/8c3d8b067f26

4设计模式七大基本原则



5.GC垃圾回收机制

垃圾回收(Garbage Collection)是Java虚拟机(JVM)垃圾回收器提供的一种用于在空闲时间不定时回收无任何对象引用的对象占据的内存空间的一种机制。

6.堆内存

JVM 所管理的最大的一块内存空间，主要用于存放各种类的实例对象。

项目：

1项目有没有考虑并发性

压力测试

[我没有高并发项目经验，但是面试的时候经常被问到高并发、性能调优方面的问题，有什么办法可以解决吗？ - 知乎 (zhihu.com)](https://www.zhihu.com/question/421237964)

2md5技术细节：

a.

MD5 消息摘要算法第五版（英文：MD5 Message-Digest Algorithm），一种被广泛使用的密码散列函数，可以产生出一个128位（16字节）（更换为十六进制为128/4=32位）的散列值（hash value），用于确保信息传输完整一致的加密算法。[16位md5的来历为32位头尾各去8位]

被经常用于数据加密以及确认文件是否被篡改。Md5有单向加密性（不可逆性），可以用于数据和密码的加密。前台输入密码后可以通过md5加密，然后直接将md5字符串保存在数据库，用户登录时只需将数据库中的md5字符串获取并与之输入的密码md5字符串比对即可通过验证。

每个文件都可以计算出一个特定的md5值，比如文件的上传和下载时都可以计算md5值并进行比对，如果相同则文件未被修改

实现：在java.security.MessageDigest包中的MessageDigest类为应用程序提供信息摘要算法的功能（如MD5/SHA算法）。

【md5值唯一吗】不唯一，一个原始数据对应一个md5；但是一个md5可能对应多个原始数据

MessageDigest类使用步骤：

①初始化生成MessageDigest对象

MessageDigest m=MessageDigest.getInstance("MD5");

MessageDigest类也是一个工厂类，其构造器是受保护的，不允许直接使用new MessageDigist()来创建对象，而必须通过其静态方法getInstance（）生成MessageDigest对象。其中传入的参数指定计算消息摘要所使用的算法，常用的有“MD5”，“SHA”等

②调用update方法传入要进行加密的byte数据

m.update(x.getBytes(“UTF8”));

x为需要计算的字符串，update传入的参数是字节类型or字节类型数组，对字符串需要先使用getBytes()方法生成字符串数组。

③计算消息摘要

byte s[]=m.digest();

执行MessageDigest对象的digest（）方法完成计算，计算的结果通过字节类型数组返回。

④调用digest方法对传入的数据进行加密

必要的话用代码将结果s转换为字符串

for (int i=0; i<s.length;i++){

result+=Integer.toHexString((0x000000ff & s[i]) | 0xffffff00).substring(6);

}

System.out.println(result);

完整：

public static void main(String[] args) {

String x="pwd";

MessageDigest m=MessageDigest.getInstance("MD5");

m.update(x.getBytes("UTF8"));

byte s[ ]=m.digest( );

String result="";

for (int i=0; i<s.length;i++){

result+=Integer.toHexString((0x000000ff & s[i]) | 0xffffff00).substring(6);

}

System.out.println(result);

}

常见加密方式

方式一：使用位运算符，将加密后的数据转换成16进制

方式二：使用格式化方式，将加密后的数据转换成16进制（推荐）

方式三：使用算法，将加密后的数据转换成16进制

Object clone() 如果实现是可克隆的，则返回一个克隆。

byte[] digest(byte[] input) 使用指定的字节数对摘要执行最后更新，然后完成摘要计算。

String getAlgorithm() 返回标识算法的字符串，与实现详细信息无关。

int getDigestLength() 返回摘要的长度（以字节为单位），如果此操作不受提供程序支持并且实现不可克隆，则返回0。

static MessageDigest getInstance(String algorithm) 返回实现指定摘要算法的MessageDigest对象。

Provider getProvider() 返回此消息摘要对象的提供程序。

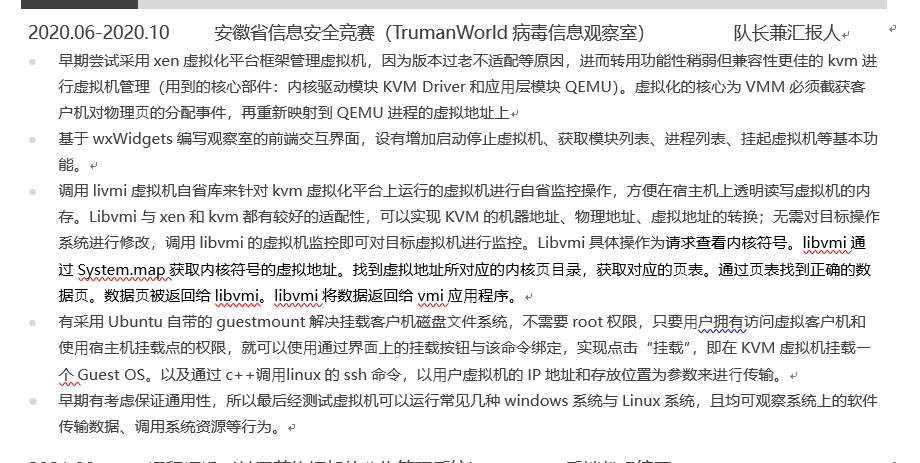
static boolean isEqual(byte[] digesta, byte[] digestb) 比较两个摘要的平等性。

void reset() 重置摘要以供进一步使用。

String toString() 返回此消息摘要对象的字符串表示形式。

void update(byte[] input) 使用指定的字节数更新摘要。

3os (得回顾trumanworld2文档)



补:

Libvmi获取进程列表：先挂起虚拟机获取内存映像，再通过system\_map拿到当前进程链表current\_task；然后找到进程链表头节点，从头节点开始遍历进程链表，最后打印进程信息

负责队内各部分工作的衔接以及最终的成果汇报

**KVM虚拟机管理平台，基于wxWidgets编写前端交互界面，有增加启动停止虚拟机，获取模块、进程列表，挂起虚拟机等功能，调用livmi来对KVM进行自省监控操作，guestmount解决挂载客户机磁盘⽂件系统，并且在常见几种windows系统与Linux系统进行了通用性测试**

1. 硬盘重新分配事件计数（不太清楚了）

重新分配的扇区计数，当硬盘发现一个读取/写入/校验错误时他将这个扇区标识为“重新分配”，并且将数据传输到一个特殊的保留区（空闲区）。这个过程也称为“重定向”，这个重新分配的扇区叫做“重新映射”。

1. 虚拟地址

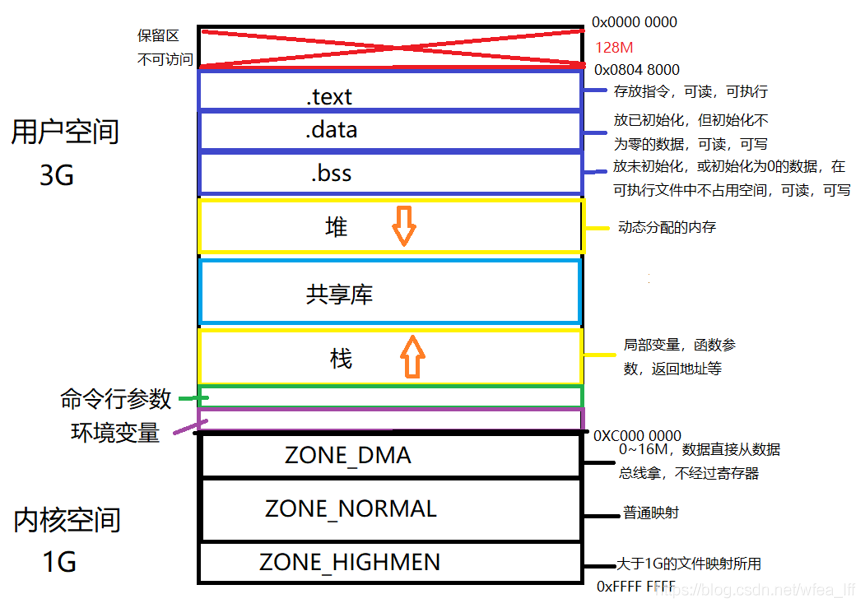
程序运行在保护模式下，访问存储器所使用的逻辑地址就称为虚拟地址。

保护模式中，程序从硬盘加载进内存的中间加了个中间层，即虚拟地址，在程序编译，连链接的时候先映射进虚拟地址，在运行的时候会再映射进物理地址。

优势是：乙程序不管如何通过怎么样的偏移都会在虚拟地址中，最后映射进物理地址也在b程序分配到的地址内，不会影响其他程序，起到了进程隔离，保护了其他进程。

（虚拟地址空间）32位OS中大小为4G；在虚拟地址里用户空间和内核空间的比例划分为

Windows—2：2；Linux—3：1



1. qemu 对虚机的地址空间管理

[qemu 对虚机的地址空间管理 - Jessica程序猿 - 博客园 (cnblogs.com)](https://www.cnblogs.com/wuchanming/p/4732604.html)

1. wxwidgets
2. libvmi

[Libvmi实现分析 - ccxikka - 博客园 (cnblogs.com)](https://www.cnblogs.com/ccxikka/p/9694888.html)

1. 自省库
2. KVM

[虚拟化技术：KVM与XEN的技术分析 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/115108777)

[KVM 虚拟化详解 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/105499858)

[KVM是什么？一文带你快速了解 Linux KVM 虚拟化 (redhat.com)](https://www.redhat.com/zh/topics/virtualization/what-is-KVM)

h.

4基于若依框架的货物管理系统

负责部分：后端的service、controller、model等的编写；还有所需对象的实体化；调用md5库对密码加密、加入验证码这些功能

（调用加密函数实现把用户密码加密成十六进制密文形式，即使是自己在数据库里查看也只能看到加密后的密码）

项目分层：

①domain层（模型层）

对应数据库中的表，数据库中有几个表就对应有几个类（货物类

②dao层（数据持久层）（我没写

数据库访问对象，主要用于与数据库进行交互，获取数据或者写入数据（增删查改），对应每一个表有自己单独的接口和接口实现类（放在dao下的impl文件中）

1. 接口对应：规范这个访问对象能做什么（增加/删除）通常方法query、queryAll、update、save、delete
2. 接口的实现：重写与数据库相关接口中的（增删改查）方法，

③service层（业务逻辑层）

分两部分：

1. 专门的接口定义

定义一些方法（例如判断存在、返回满足要求的对象

1. 接口的实现类（impl）

实现接口，重写具体方法

④controller层（控制器层）

⑤views视图层

界面显示相关，调用service层对象

调用的接口：用了加密算法md5用于保护用户的密码信息，用了kaptcha图片验证码，避免用户恶意破解密码、防止爬虫占用用户资源

5（救命）项目过程中遇到什么问题，怎么解决

部分细节操作或者需要的功能不会使用与实现

已停止更新的虚拟机平台XEN版本和电脑不兼容