# JAVA

## 1.反射

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法；这种动态获取的以及动态调用对象的方法的功能称为Java的反射机制。

三种方式获取class对象：

/第一种方式 通过对象getClass方法

Person person = new Person();

Class<?> class1 = person.getClass();

//第二种方式 通过类的class属性

class1 = Person.class;

try {

//第三种方式 通过Class类的静态方法——forName()来实现

class1 = Class.forName("com.whoislcj.reflectdemo.Person");

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

Field[] allFields = class1.getDeclaredFields();//获取class对象的所有属性

Field[] publicFields = class1.getFields();//获取class对象的public属性

class对象动态生成

//第一种方式 Class对象调用newInstance()方法生成

Object obj = class1.newInstance();

//第二种方式 对象获得对应的Constructor对象，再通过该Constructor对象的newInstance()方法生成

Constructor<?> constructor = class1.getDeclaredConstructor(new Class[]{String.class});//获取指定声明构造函数

obj = constructor.newInstance(new Object[]{"lcj"});

应用：

动态代理

Spring的AOP思想

## 2.类加载

Java虚拟机把描述类的数据从Class文件加载到内存，并对数据进行校验、转换解析和初始化，最终形成可以被虚拟机直接使用的Java类型，这就是虚拟机的加载机制。

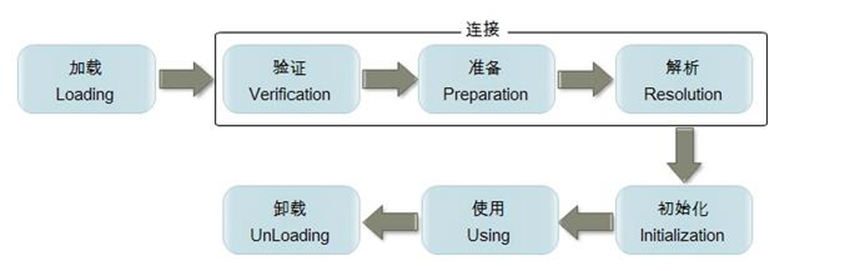
它的整个生命周期包括了：加载（Loading）、验证（Verification）、准备（Preparation）、解析（Resolution）、初始化（Initialization）、使用（using）、和卸载（Unloading）七个阶段。其中验证、准备和解析三个部分统称为连接（Linking）

三个类加载器：

引导（Bootstrap）类加载器：负责将 <JAVA\_HOME>/lib路径下的核心类库或-Xbootclasspath参数指定的路径下的jar包加载到内存中

扩展（Extension）类加载器：负责加载<JAVA\_HOME>/lib/ext目录下或者由系统变量-Djava.ext.dir指定位路径中的类库

系统（System）类加载器（也称应用类加载器）：负责加载系统类路径java -classpath或-D java.class.path 指定路径下的类库



## 3.JAVA8新特性

### 1. 函数式接口和Lambda表达式

函数式接口是只包含一个方法的抽象接口，通过@FunctionalInterface注解（非必须），将一个接口标注为函数式接口。函数式接口是为Java 8中的lambda而设计的，lambda表达式的方法体其实就是函数接口的实现。

Lambda表达式语法结构如下：

(parameters) -> expression 或者 (parameters) -> {statements;}

lambda表达式使代码变得简单，代替了匿名内部类

new Thread(new Runnable(){

@Override

public void run() {

for (int i = 0; i < 10; i++)

System.out.println("Without Lambda Expression");

}}).start();

### 2.接口的默认方法

（default修饰）：接口里也可以写方法体了，实现该接口的类不再强制实现该方法，只需要在方法签名增加default签名并实现方法体；

接口可以定义静态方法，通过接口调用。实现类不需实现，也无法在实现类中直接调用。

### 3.方法引用

构造器引用。语法是Class::new，或者更一般的Class< T >::new，要求构造器方法是没有参数；

静态方法引用。语法是Class::static\_method，要求接受一个Class类型的参数；

特定类的任意对象方法引用。它的语法是Class::method。要求方法是没有参数的；

特定对象的方法引用，它的语法是instance::method。要求方法接受一个参数，与3不同的地方在于，3是在列表元素上分别调用方法，而4是在某个对象上调用方法，将列表元素作为参数传入；

### 4.重复注解

相同的注解在同一地方也可以声明多次。重复注解机制本身需要用@Repeatable注解。

### 5.扩展注解的支持

Java 8扩展了注解的上下文，几乎可以为任何东西添加注解，包括局部变量、泛型类、父类与接口的实现，连方法的异常也能添加注解

### 6.引入optional类

Java 8引入Optional类来防止空指针异常，Optional类最先是由Google的Guava项目引入的。Optional类实际上是个容器：它可以保存类型T的值，或者保存null。使用Optional类我们就不用显式进行空指针检查了

### 7.Stream API

Stream API是把真正的函数式编程风格引入到Java中。

### 8.Date/Time API

Java 8新的Date-Time API (JSR 310)受Joda-Time的影响，提供了新的java.time包，可以用来替代 java.util.Date和java.util.Calendar。一般会用到Clock、LocaleDate、LocalTime、LocaleDateTime、ZonedDateTime、Duration这些类，对于时间日期的改进还是非常不错的。

### 9.引入Javascript引擎Nashorn

Nashorn允许在JVM上开发运行JavaScript应用，允许Java与JavaScript相互调用。

### 10.Base64

对Base64编码的支持已经被加入到Java 8官方库中，这样不需要使用第三方库就可以进行Base64编码

其他

更好的类型推测机制：Java 8在类型推测方面有了很大的提高，这就使代码更整洁，不需要太多的强制类型转换了。

编译器优化：Java 8将方法的参数名加入了字节码中，这样在运行时通过反射就能获取到参数名，只需要在编译时使用-parameters参数。

并行（parallel）数组：支持对数组进行并行处理，主要是parallelSort()方法，它可以在多核机器上极大提高数组排序的速度。

并发（Concurrency）：在新增Stream机制与Lambda的基础之上，加入了一些新方法来支持聚集操作。

Nashorn引擎jjs：基于Nashorn引擎的命令行工具。它接受一些JavaScript源代码为参数，并且执行这些源代码。

类依赖分析器jdeps：可以显示Java类的包级别或类级别的依赖。

JVM的PermGen空间被移除：取代它的是Metaspace（JEP 122）

## 4.面向对象的特性

### 1）抽象性 ：

忽略与主题有关，只专注与目标本身有关的方面，

其中包括数据抽象：类的属性，如鸟的翅膀

过程抽象：类的方法，如鸟的飞行

### 2）封装性：

把数据和过程实行保护，通过private和set,get

### 3）继承性：

是一种联结类的层次模型，子类继承父类，是一种表达共性的方法

### 4）多态性：

允许不同类的对象对同一种消息做出响应

## 5. Java 中覆盖和重载是什么意思？

覆盖（Override）是指子类对父类方法的一种重写，覆盖方法返回值，函数名，参数列表都要相同，只能比父类抛出更少的异常，访问权限不能比父类的小。被覆盖的方法不能是 private 的，否则只是在子类中重新定义了一个方法；

重载（Overload）表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同。参数列表指的是参数个数，类型，顺序。

面试题：

函数的返回值不同可以构成重载吗？为什么？

不可以，因为 Java 中调用函数并不需要强制赋值。

## 6．抽象类和接口的区别有哪些？

1）抽象类中可以没有抽象方法；接口中的方法必须是抽象方法；

2）抽象类中可以有普通的成员变量；接口中的变量必须是 static final 类型的，必须被初始化 , 接口中只有常量，没有变量。

3）抽象类只能单继承，接口可以继承多个父接口；

4）Java8 中接口中会有 default 方法，即方法可以被实现。

## 7.java和C++的区别？

都是面向对象的语言，都支持封装、继承和多态；

指针：Java 不提供指针来直接访问内存，程序更加安全；

继承： Java 的类是单继承的，C++ 支持多重继承； Java 通过一个类实现多个接口来实现 C++ 中的多重继承； Java 中类不可以多继承，但是！！！接口可以多继承；

内存： Java 有自动内存管理机制，不需要程序员手动释放无用内存。

## 8.java常用的包

java.lang、java.util、java.io、java.net、java.sql。

## 9. JDK，JRE 和 JVM 的联系和区别：

JDK 是 java 开发工具包，是 java 开发环境的核心组件，并提供编译、调试和运行一个 java 程序所需要的所有工具，可执行文件和二进制文件，是一个平台特定的软件。

JRE 是 java 运行时环境，是 JVM 的实施实现，提供了运行 java 程序的平台。JRE 包含了 JVM，但是不包含 java 编译器 / 调试器之类的开发工具。

JVM 是 java 虚拟机，当我们运行一个程序时，JVM 负责将字节码转换为特定机器代码，JVM 提供了内存管理 / 垃圾回收和安全机制等。

这种独立于硬件和操作系统，正是 java 程序可以一次编写多处执行的原因。

区别：

JDK 用于开发，JRE 用于运行 java 程序；

JDK 和 JRE 中都包含 JVM；

JVM 是 java 编程语言的核心并且具有平台独立性。

## 10.StringBuffer和StringBuilder的区别

1、StringBuffer 与 StringBuilder 中的方法和功能完全是等价的，

2、只是StringBuffer 中的方法大都采用了 synchronized 关键字进行修饰，因此是线程安全的，

而 StringBuilder 没有这个修饰，可以被认为是线程不安全的。

3、在单线程程序下，StringBuilder效率更快，因为它不需要加锁，不具备多线程安全

而StringBuffer则每次都需要判断锁，效率相对更低

底层原理：初始化大小默认16，每次扩容为大小\*2+2，如果还不够就直接扩容到所需大小。

## 11.Java序列化

序列化是一种对象持久化的手段，把对象转化为二进制序列形式，普遍应用在网络传输，远程方法调用的场景。

类实现Serializable接口即可实现序列化

注意：

1. 当一个父类实现序列化，子类自动实现序列化，不需要显式实现Serializable接口；

2. 当属性声明为static 或者transient(临时)时，该属性不能被序列化

## 12．Java泛型

### 1）. Java中的泛型是什么 ? 使用泛型的好处是什么?

泛型是一种参数化类型的机制。它可以使得代码适用于各种类型，从而编写更加通用的代码，例如集合框架。

泛型是一种编译时类型确认机制。它提供了编译期的类型安全，确保在泛型类型（通常为泛型集合）上只能使用正确类型的对象，避免了在运行时出现ClassCastException。

### 2）.Java的泛型是如何工作的 ? 什么是类型擦除 ?

泛型的正常工作是依赖编译器在编译源码的时候，先进行类型检查，然后进行类型擦除并且在类型参数出现的地方插入强制转换的相关指令实现的。

编译器在编译时擦除了所有类型相关的信息，所以在运行时不存在任何类型相关的信息。例如List<String>在运行时仅用一个List类型来表示。为什么要进行擦除呢？这是为了避免类型膨胀。

### 3). 什么是泛型中的限定通配符和非限定通配符 ?

限定通配符对类型进行了限制。有两种限定通配符，一种是<? extends T>它通过确保类型必须是T的子类来设定类型的上界，另一种是<? super T>它通过确保类型必须是T的父类来设定类型的下界。泛型类型必须用限定内的类型来进行初始化，否则会导致编译错误。另一方面<?>表示了非限定通配符，因为<?>可以用任意类型来替代

## 13.java集合（重点）

### 1)概念

（1） Collection(单列集合)

List(有序,可重复)

ArrayList

底层数据结构是数组,查询快,增删慢

线程不安全,效率高

Vector

底层数据结构是数组,查询快,增删慢

线程安全,效率低

LinkedList

底层数据结构是链表,查询慢,增删快

线程不安全,效率高

Set(无序,唯一)

HashSet

底层数据结构是哈希表。

哈希表依赖两个方法：hashCode()和equals()

执行顺序：

首先判断hashCode()值是否相同

是：继续执行equals(),看其返回值

是true:说明元素重复，不添加

是false:就直接添加到集合

否：就直接添加到集合

最终：

自动生成hashCode()和equals()即可

LinkedHashSet

底层数据结构由链表和哈希表组成。

由链表保证元素有序。

由哈希表保证元素唯一。

TreeSet

存储的对象需要实现comparable接口

底层数据结构是红黑树。(是一种自平衡的二叉树)

如何保证元素唯一性呢?

调用对象的compareTo方法 根据比较的返回值是否是0来决定

如何保证元素的排序呢?

两种方式

自然排序(元素具备比较性)

让元素所属的类实现Comparable接口

比较器排序(集合具备比较性)

让集合接收一个Comparator的实现类对象

（2）Map(双列集合)

A:Map集合的数据结构仅仅针对键有效，与值无关。

B:存储的是键值对形式的元素，键唯一，值可重复。

HashMap

底层数据结构是哈希表。线程不安全，效率高

哈希表依赖两个方法：hashCode()和equals()

执行顺序：

首先判断hashCode()值是否相同

是：继续执行equals(),看其返回值

是true:说明元素重复，不添加

是false:就直接添加到集合

否：就直接添加到集合

最终：

自动生成hashCode()和equals()即可

LinkedHashMap

底层数据结构由链表和哈希表组成。

由链表保证元素有序。

由哈希表保证元素唯一。

Hashtable

底层数据结构是哈希表。线程安全，效率低

哈希表依赖两个方法：hashCode()和equals()

执行顺序：

首先判断hashCode()值是否相同

是：继续执行equals(),看其返回值

是true:说明元素重复，不添加

是false:就直接添加到集合

否：就直接添加到集合

最终：

自动生成hashCode()和equals()即可

TreeMap

底层数据结构是红黑树。(是一种自平衡的二叉树)

如何保证元素唯一性呢?

根据比较的返回值是否是0来决定

如何保证元素的排序呢?

两种方式

自然排序(元素具备比较性)

让元素所属的类实现Comparable接口

比较器排序(集合具备比较性)

让集合接收一个Comparator的实现类对象

2:到底使用那种集合：

看需求。

是否是键值对象形式:

是：Map

键是否需要排序:

是：TreeMap

否：HashMap

不知道，就使用HashMap。

否：Collection

元素是否唯一:

是：Set

元素是否需要排序:

是：TreeSet

否：HashSet

不知道，就使用HashSet

否：List

要安全吗:

是：Vector(其实我们也不用它,后面我们讲解了多线程以后，我在给你回顾用谁)

否：ArrayList或者LinkedList

增删多：LinkedList

查询多：ArrayList

不知道，就使用ArrayList

不知道，就使用ArrayList

3:集合的常见方法及遍历方式

Collection:

add()

remove()

contains()

iterator()

size()

遍历方式：

增强for

迭代器

|--List

get()

遍历：

普通for

|--Set

Map:

put()

remove()

containskey(),containsValue()

keySet()

get()

value()

entrySet()

size()

遍历方式：

根据键找值

根据键值对对象分别找键和值。

### 2）底层原理

#### a)HashMap

##### 1-两个重要参数

容量(Capacity)和负载因子(Load factor)：

Capacity就是bucket的大小，Load factor就是bucket填满程度的最大比例。当bucket中的entries的数目大于capacity\*load factor时就需要调整bucket的大小为当前的2倍。

##### 2-Put函数的实现

put函数大致的思路为：

1.对key的hashCode()做hash，然后再计算index;

2.如果没碰撞直接放到bucket里；

3.如果碰撞了，以链表的形式存在buckets后；

4.如果碰撞导致链表过长(大于等于TREEIFY\_THRESHOLD)，就把链表转换成红黑树；

5.如果节点已经存在就替换old value(保证key的唯一性)

6.如果bucket满了(超过load factor\*current capacity)，就要resize。

##### 3-get函数的实现

1.bucket里的第一个节点，直接命中；

2.如果有冲突，则通过key.equals(k)去查找对应的entry

若为树，则在树中通过key.equals(k)查找，O(logn)；

若为链表，则在链表中通过key.equals(k)查找，O(n)。

在Java 8之前的实现中是用链表解决冲突的，在产生碰撞的情况下，进行get时，两步的时间复杂度是O(1)+O(n)。因此，当碰撞很厉害的时候n很大，O(n)的速度显然是影响速度的。

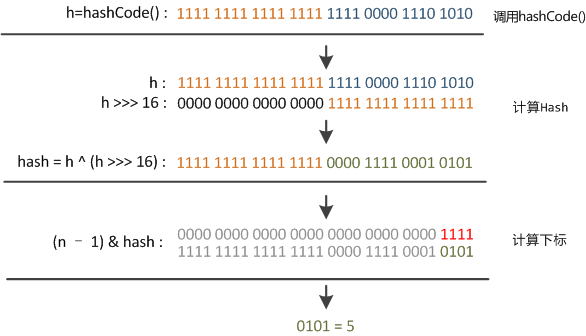
因此在Java 8中，利用红黑树替换链表，这样复杂度就变成了O(1)+O(logn)了

默认碰撞数是8就替换

##### 4-hash()方法

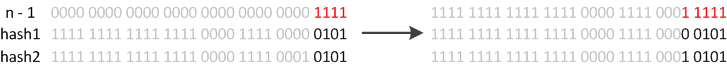
对键值作hashcode（），得到的值进行高16bit不变，低16bit和高16bit做了一个异或。（为了使1尽量分布均匀）

##### 5-计算下标方法

(n-1)&hash

##### 6-resize()方法

当超过限制的时候会resize，然而又因为我们使用的是2次幂的扩展(指长度扩为原来2倍)，所以，元素的位置要么是在原位置，要么是在原位置再移动2次幂的位置。主要是



#### b)ConcurrentHashMap

##### 1-概念

HashTable性能差主要是由于所有操作需要竞争同一把锁，而如果容器中有多把锁，每一把锁锁一段数据，这样在多线程访问时不同段的数据时，就不会存在锁竞争了，这样便可以有效地提高并发效率。这就是ConcurrentHashMap所采用的"分段锁"思想。

ConcurrentHashMap的主干是个Segment数组。

一个Segment就是一个子哈希表，Segment里维护了一个HashEntry数组，并发环境下，对于不同Segment的数据进行操作是不用考虑锁竞争的。（就按默认的ConcurrentLeve为16来讲，理论上就允许16个线程并发执行，有木有很酷）

所以，对于同一个Segment的操作才需考虑线程同步，不同的Segment则无需考虑。

get方法无需加锁，由于其中涉及到的共享变量都使用volatile修饰，volatile可以保证内存可见性，所以不会读取到过期数据。

##### 2-结构

##### 3-HashEntry

每个HashEntry代表Hash表中的一个节点，在其定义的结构中可以看到，除了value值没有定义final，其余的都定义为final类型。Value被volatile修饰

##### 4-Segment

Segment 类继承于 ReentrantLock 类，从而使得 Segment 对象能充当锁的角色。每个 Segment 对象用来守护其（成员对象 table 中）包含的若干个桶。

##### 5-java7与java8

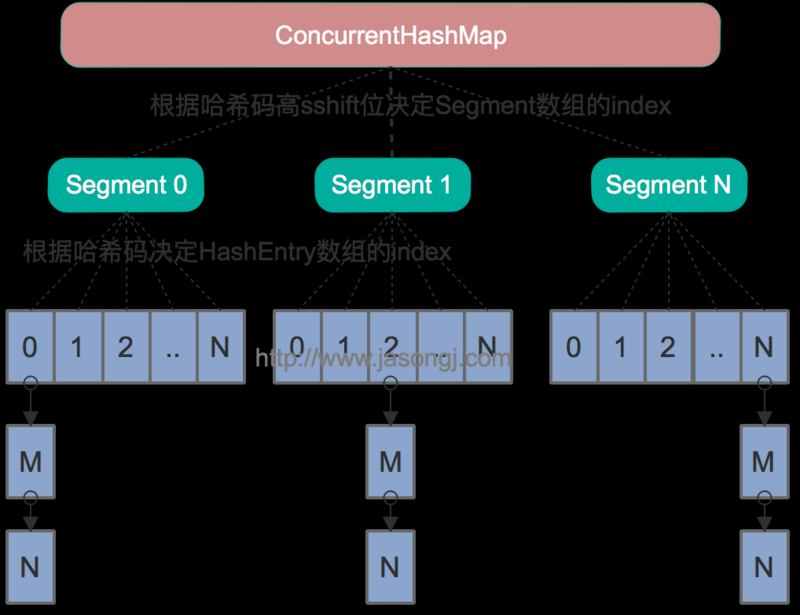
注意：

Java7

基于分段锁

采用的是对segment进行加锁，整段节点数组。

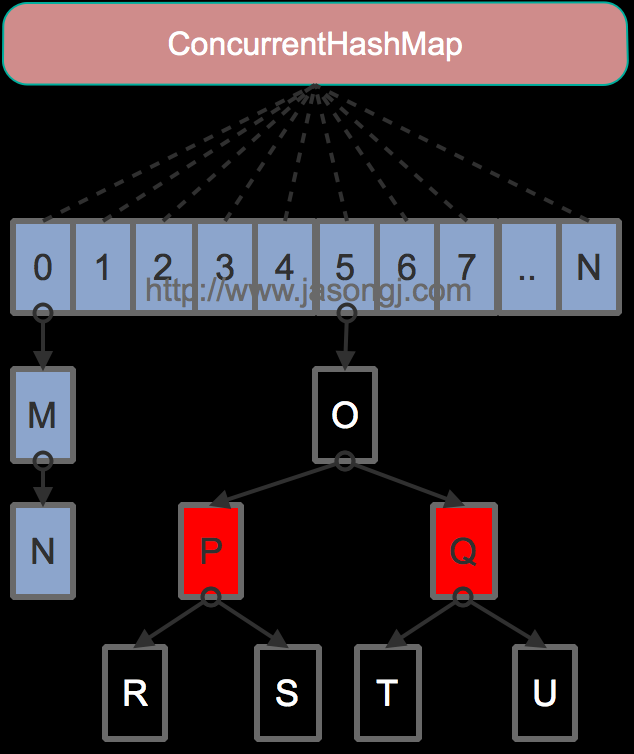
同步方式是：Segment继承自RentranLock,直接在判断属于哪段segment之后调用put，在这个方法里加lock() 与unlock()



Java8

进行了改进，基于CAS无锁技术，compare and swap, CAS有3个操作数，内存值V，旧的预期值A，要修改的新值B。当且仅当预期值A和内存值V相同时，将内存值V修改为B，否则什么都不做。

变成对节点数组的元素进行加锁，也就是每行数据进行加锁，



### 3）面试题

##### 1-常用集合类有哪些?

答：Map 接口和 Collection 接口是所有集合框架的父接口：

a)Collection 接口的子接口包括：Set 接口和 List 接口；

b)Map 接口的实现类主要有：HashMap、TreeMap、Hashtable、ConcurrentHashMap 以及 Properties 等；

c)Set 接口的实现类主要有：HashSet、TreeSet、LinkedHashSet 等；

d)List 接口的实现类主要有：ArrayList、LinkedList、Stack 以及 Vector 等。

##### 2-HashMap 和 Hashtable 的区别有哪些？（必问）

a)HashMap 没有考虑同步，是线程不安全的；Hashtable所有相关操作都是synchronized的，是线程安全的；

b)前者允许 null 作为 Key；后者不允许 null 作为 Key。

##### 3-HashMap 的底层实现你知道吗？

在 Java8 之前，其底层实现是数组 + 链表实现，

Java8 使用了数组 + 链表 + 红黑树实现。当碰撞不多时结构是数组+链表，当碰撞超过默认8个时，结构转换为数组+红黑树。查找速度从O(1)+O(N)变为O(1)+O(logN)

##### 4-ConcurrentHashMap 和 Hashtable 的区别？ （必问）

ConcurrentHashMap 结合了 HashMap 和 HashTable 二者的优势。HashMap 没有考虑同步，hashtable 考虑了同步的问题。但是 hashtable 在每次同步执行时都要锁住整个结构。 ConcurrentHashMap 锁的方式是稍微细粒度的。 ConcurrentHashMap 将 hash 表分为 16 个桶（默认值），诸如 get,put,remove 等常用操作只锁当前需要用到的桶。

##### 5-你了解ConcurrentHashMap的底层实现吗？

Java8之前：

1.ConcurrentHashMap类包含两个静态内部类 HashEntry 和 Segment；前者用来封装映射表的键值对，后者用来充当锁的角色；

2.Segment 是一种可重入的锁 ReentrantLock，每个 Segment 守护一个 HashEntry 数组里得元素，当对 HashEntry 数组的数据进行修改时，必须首先获得对应的 Segment 锁

3.底层采用数据+链表

Java8之后：

1. ConcurrentHashMap采用静态内部类Node数组来封装映射表，对每个数组元素采用synchronized方式进行同步。每个数组元素就是一个锁
2. 底层采用数组+链表+红黑树

##### 6- HashMap 的长度为什么是 2 的幂次方？

1. 在计算数组下标时，为了考虑效率问题，采用与操作，而不用取模运算
2. 为了尽可能的分布均匀 减少碰撞，就采用2的幂次方，这样length-1再进行与操作，保证都是1111..，都可以与到。
3. 即使用户初始化的长度不为2的幂次方，也会计算出比输入稍大的2的幂次方树

##### 7- List 和 Set 的区别是啥？

List元素可以重复，按添加顺序排序

Set元素唯一，是无序的

##### 8- List、Set 和 Map 的初始容量和加载因子

1. List

ArrayList 的初始容量是 10；加载因子为 0.5； 扩容增量：原容量的 0.5 倍 +1；一次扩容后长度为 16。

Vector 初始容量为 10，加载因子是 1。扩容增量：原容量的 1 倍，如 Vector 的容量为 10，一次扩容后是容量为 20。

2. Set

HashSet，初始容量为 16，加载因子为 0.75； 扩容增量：原容量的 1 倍； 如 HashSet 的容量为 16，一次扩容后容量为 32

3. Map

HashMap，初始容量 16，加载因子为 0.75； 扩容增量：原容量的 1 倍； 如 HashMap 的容量为 16，一次扩容后容量为 32

##### 9- Comparable 接口和 Comparator 接口有什么区别？

1. Comparable实现比较简单，直接在所在类实现此接口并覆盖compare方法，但是会修改源码
2. Comparator接口不需要修改源码，自定义一个比较器类并实现该接口覆盖compare方法，并且可以实现更复杂逻辑，如引入接口多态

##### 10- Java 集合的快速失败机制 “fail-fast”

它是 java 集合的一种错误检测机制，当多个线程对集合进行结构上的改变的操作时，有可能会产生 fail-fast 机制。读的时候又修改了数据就会抛出ConcurrentModificationException异常。

解决：

List:

new CopyOnWriteArrayList(list) //读写分离思想，写时复制的方法，当有新元素添加到CopyOnWriteArrayList时，先从原有的数组中拷贝一份出来，然后在新的数组做写操作，写完之后，再将原来的数组引用指向到新数组。（添加操作加锁，读操作不加锁）

Map:

1. 使用Collections.synchronizedMap()（类似于装饰者模式）
2. 使用ConcurrentHashMap

## 14.transient关键字

java 的transient关键字为我们提供了便利，你只需要实现Serilizable接口，将不需要序列化的属性前添加关键字transient，序列化对象的时候，这个属性就不会序列化到指定的目的

地中。

## 15.多线程

### 1）线程与进程的区别?

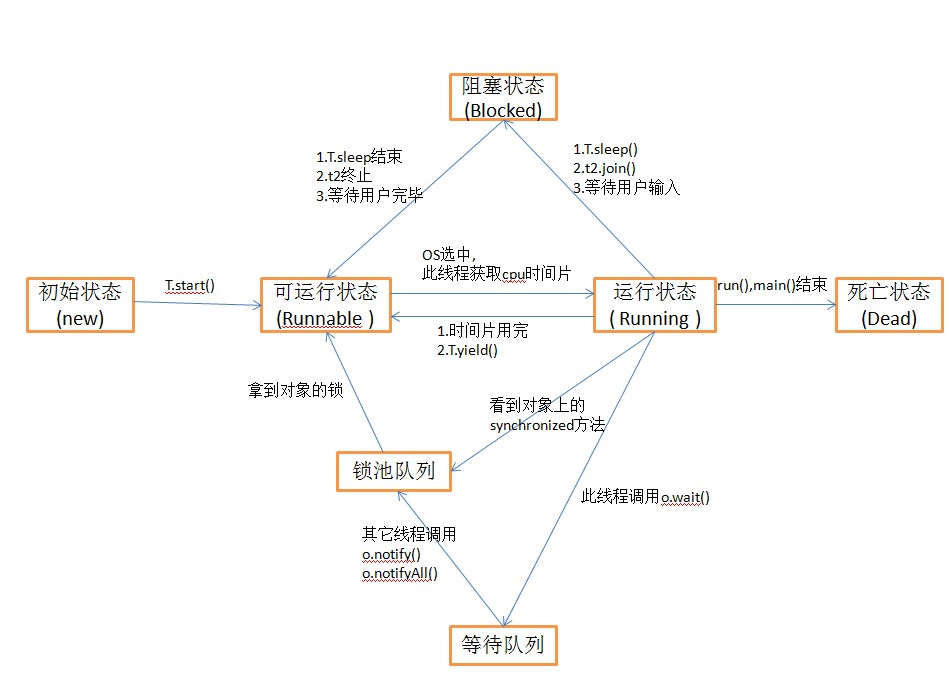
进程是指资源的分配和调度的最小单元，是执行的程序；

线程是程序执行的最小单元，一个进程可以拥有多个线程。

使用多线程的好处：

1. 减少程序响应的时间
2. 与进程相比，创建与切换线程的开销更小

### 2）线程的状态（五种）



1. 新建(NEW)：

新创建了一个线程对象。

2. 可运行(RUNNABLE)：

线程对象创建后，其他线程(比如main线程）调用了该对象的start()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获取cpu 的使用权 。

1. 运行(RUNNING)：

可运行状态(runnable)的线程获得了cpu 时间片（timeslice） ，执行程序代码。

1. 阻塞(BLOCKED)：

阻塞状态是指线程因为某种原因放弃了cpu 使用权，也即让出了cpu timeslice，暂时停止运行。直到线程进入可运行(runnable)状态，才有机会再次获得cpu timeslice 转到运行(running)状态。阻塞的情况分三种：

(一). 等待阻塞：运行(running)的线程执行o.wait()方法，JVM会把该线程放入等待队列(waitting queue)中。

(二). 同步阻塞：运行(running)的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池(lock pool)中。

(三). 其他阻塞：运行(running)的线程执行Thread.sleep(long ms)或t.join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入可运行(runnable)状态。

5. 死亡(DEAD)：

线程run()、main() 方法执行结束，或者因异常退出了run()方法，则该线程结束生命周期。死亡的线程不可再次复生。

### 3）多线程产生死锁的 4 个必要条件？

1.互斥条件：一个资源每次只能被一个线程使用；

2.请求与保持条件：一个线程因请求资源而阻塞时，对已获得的资源保持不放；

3.不剥夺条件：线程已经获得的资源，在未使用完之前，不能强行剥夺；

4.循环等待条件：若干线程之间形成一种头尾相接的循环等待资源关系。

### 4）如何避免死锁？

1.加锁顺序（线程按照一定的顺序加锁）

2.加锁时限（线程尝试获取锁的时候加上一定的时限，超过时限则放弃对该锁的请求，并释放自己占有的锁）

3.死锁检测

### 5）sleep和wait的区别？

1，这两个方法来自不同的类分别是Thread和Object

2，最主要是sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法。

3，wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用

synchronized(x){

x.notify()

//或者wait()

}

4,sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常

# 前端

# 后台Web

# 数据库

知识点：

面试题：

答案：

# 网络协议

# 数据结构与算法

## 1八大排序算法



### a)直接插入

思想：在待排序数据中，默认前边数据是排好序，从第n个往前边n-1插入

### b)希尔排序（增量递减排序）

思想：将数据按增量分为组，初始增量d为(n/2)，然后增量每次减半d/2，直到等于0终止。组内数据再进行直接出入排序。

### c)选择排序

思想：依次取数据与其后边所有数据比较，较小就交换，使得最小在前边。

### d)堆排序

思想：首先建立最大堆，然后把数组尾部与堆顶交换，使最大数在最后。循环数据完成排序

建立最大堆思想，把数组数据当作完全二叉树顺序存储。则非叶子节点在第n/2个，从这里往前判断 每个节点与其叶子节点的大小，若小于其较大子节点，则交换。直到判断完所有，则最大堆建成。

### e)冒泡排序

思想：待排数据两两比较，较大的往后移，一次循环之后，最大值出现在末尾。

### f)快速排序

思想：确定一个基准数（一般选择第一个），从数组两端往内比较，有前后两个指针，来回与基准数比较，最终找到基准数的位置。然后再递归分开两边的数据。

### g)归并排序

思想：采用分治思想，把有序子序列合并成新序列，最终实现排序。把数据一直取mid，然后递归分开左右两边，终止条件left<right。借助辅助数组，临时存储两个子序列，轮询较小值先存入辅助数组。

### h)基数排序

思想：按照最大数据的位数进行循环比较。从最低位开始，每次判断，都把数据按照当前位数大小存储到指定位数位置，然后再重新读入数组中。直到最高为为止，则实现排序。