# 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。

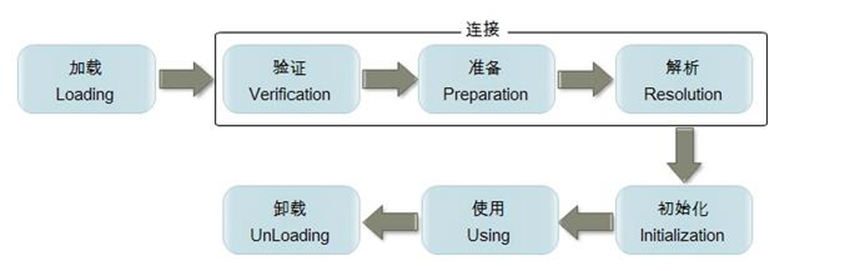
它的整个生命周期包括了：加载（Loading）、验证（Verification）、准备（Preparation）、解析（Resolution）、初始化（Initialization）、使用（using）、和卸载（Unloading）七个阶段。其中验证、准备和解析三个部分统称为连接（Linking）

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。

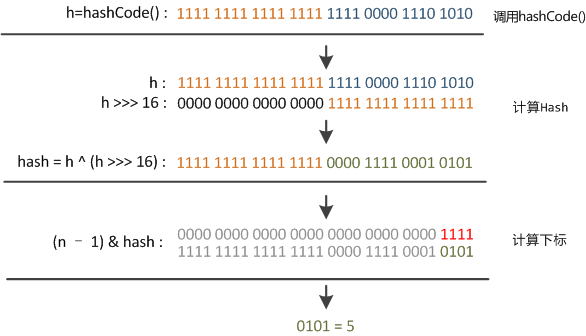
因此在Java 8中，利用红黑树替换链表，这样复杂度就变成了O(1)+O(logn)了

默认碰撞数是8就替换

##### 4-hash()方法

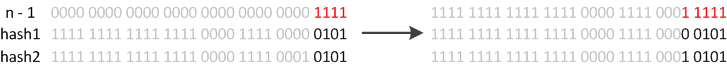
对键值作hashcode（），得到的值进行高16bit不变，低16bit和高16bit做了一个异或。（为了使1尽量分布均匀）

##### 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

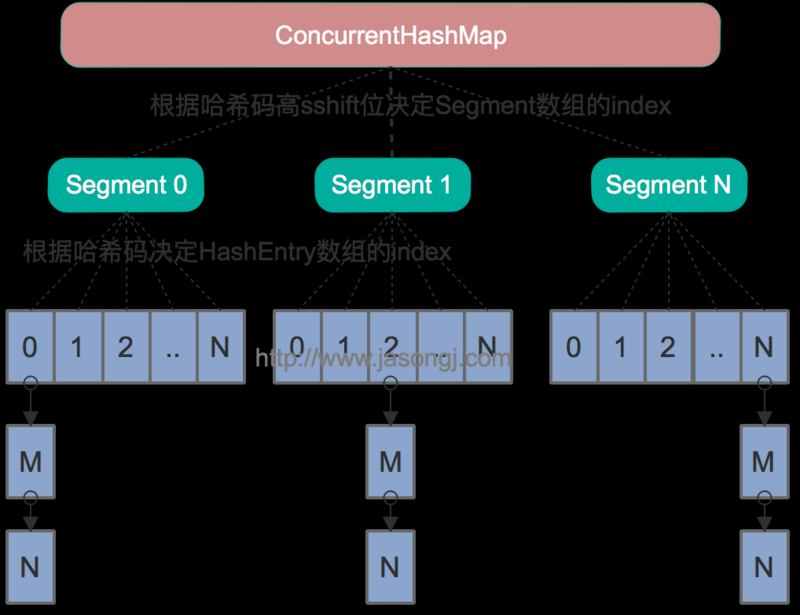
注意：

Java7

基于分段锁

采用的是对segment进行加锁，整段节点数组。

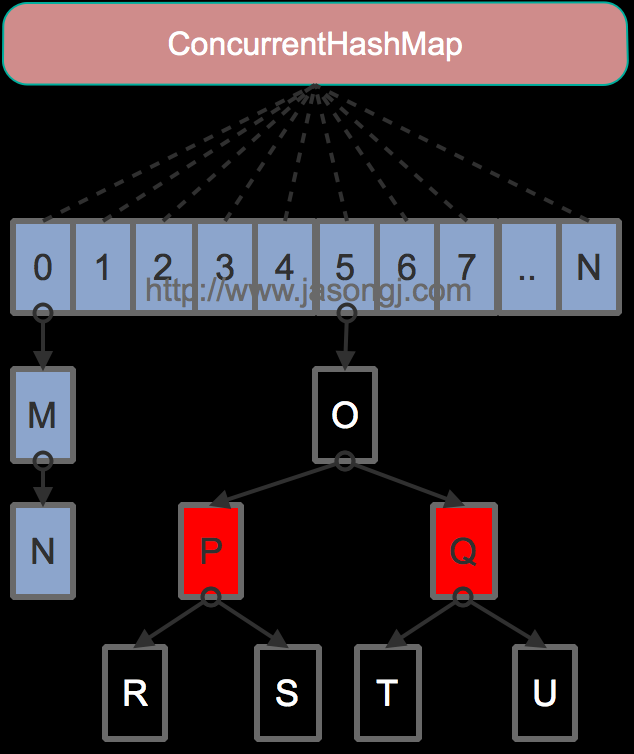
同步方式是：Segment继承自RentranLock,直接在判断属于哪段segment之后调用put，在这个方法里加lock() 与unlock()



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?

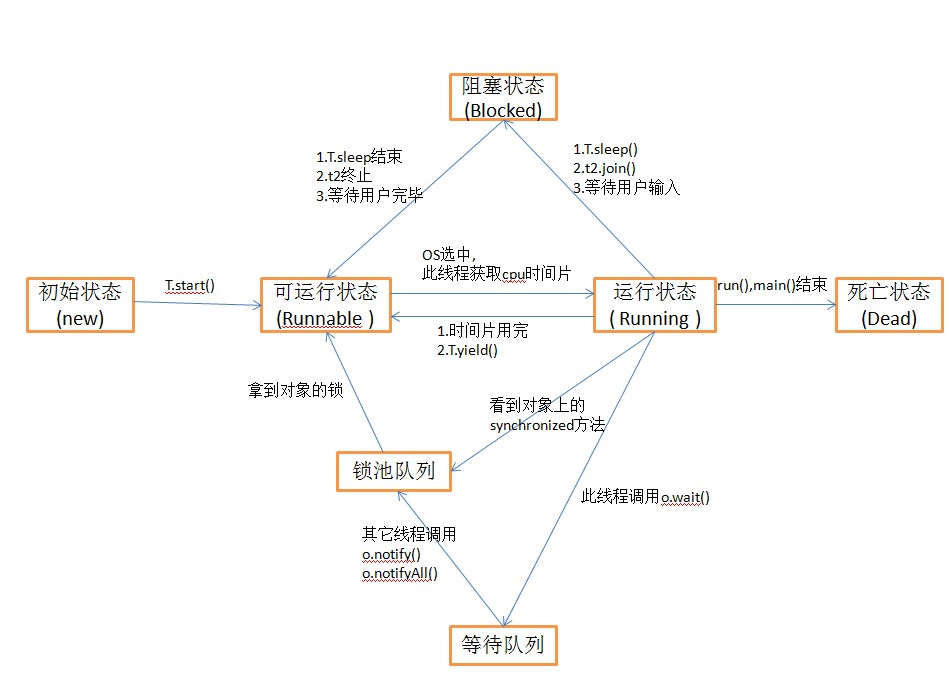
进程是指资源的分配和调度的最小单元，是执行的程序；

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

### 6）synchronized 关键字的底层实现

1.进入时，执行 monitorenter，将计数器 +1，释放锁 monitorexit 时，计数器-1；

2.当一个线程判断到计数器为 0 时，则当前锁空闲，可以占用；反之，当前线程进入等待状态。

含义：（monitor 机制）

### 7）volatile 关键字

该关键字可以保证可见性不保证原子性。

功能：

1.主内存和工作内存，直接与主内存产生交互，进行读写操作，保证可见性；

2.禁止 JVM 进行的指令重排序。

### 8）ThreadLocal关键字

当使用 ThreadLocal 维护变量时，其为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立的改变自己的副本，而不会影响其他线程对应的副本。

ThreadLocal 内部实现机制：

1.每个线程内部都会维护一个类似 HashMap 的对象，称为 ThreadLocalMap，里边会包含若干了 Entry（K-V 键值对），相应的线程被称为这些 Entry 的属主线程；

2.Entry 的 Key 是一个 ThreadLocal 实例，Value 是一个线程特有对象。Entry 的作用即是：为其属主线程建立起一个 ThreadLocal 实例与一个线程特有对象之间的对应关系；

1. Entry 对 Key 的引用是弱引用；Entry 对 Value 的引用是强引用。

### 9）Automic关键字

可以使基本数据类型以原子的方式实现自增自减等操作。Java.util.concurrent.atomic 包下的类 AtomicInteger 的使用。

### 10) 线程池有了解吗？（必考）

Java.util.concurrent.ThreadPoolExecutor

构造函数中需要传入的参数包括corePoolSize、maximumPoolSize、keepAliveTime、timeUnit和workQueue

执行线程：线程池对象的submit()方法或者execute()方法，要求线程池运行的任务（这些任务必须实现Runnable接口或者Callable接口

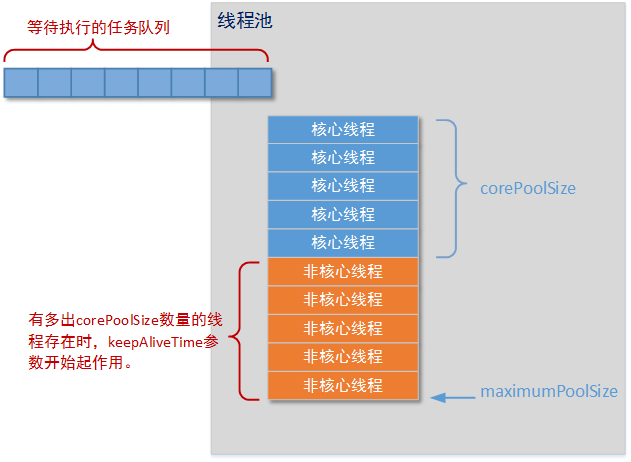
执行线程过程：

1.如果运行的线程少于 corePoolSize，则 Executor 始终首选添加新的线程，而不进行排队；

2.如果运行的线程等于或者多于 corePoolSize，则 Executor 始终首选将请求加入队列，而不是添加新线程；

3.如果无法将请求加入队列，即队列已经满了，则创建新的线程，除非创建此线程超出 maxinumPoolSize， 在这种情况下，任务将被拒绝。

如果这个任务，无法被“核心线程”直接执行，又无法加入等待队列，又无法创建“非核心线程”直接执行，且你没有为线程池设置RejectedExecutionHandler。这时线程池会抛出RejectedExecutionException异常，即线程池拒绝接受这个任务。



## 16.JVM内存管理

### 1）Java运行时数据区

#### a)程序计数器

程序计数器（Program Counter Register） 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。在虚拟机的概念模型里，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执行字节码指令。

此内存区域是唯一一个在java虚拟机规范中没有规定任何OutOfMemoryError情况的区域

#### b)Java虚拟机栈

同样是线程私有，描述Java方法执行的内存模型：每个方法在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。一个方法对应一个栈帧。

规定的异常情况有两种：1.线程请求的栈的深度大于虚拟机所允许的深度，将抛出StackOverflowError异常；2.如果虚拟机可以动态扩展，如果扩展时无法申请到足够的内存，就抛出OutOfMemoryError异常。

#### c)本地方法栈

和Java虚拟机栈很类似，不同的是本地方法栈为Native方法服务。

#### d)Java堆

是Java虚拟机所管理的内存中最大的一块。由所有线程共享，在虚拟机启动时创建。堆区唯一目的就是存放对象实例。

堆中可细分为新生代和老年代，再细分可分为Eden空间、From Survivor空间、To Survivor空间。

堆无法扩展时，抛出OutOfMemoryError异常

#### e)方法区

所有线程共享，存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

当方法区无法满足内存分配需求时，抛出OutOfMemoryError

#### f)运行时常量池

jdk6以前：

它是方法区的一部分，Class文件中除了有类的版本、字段、方法、接口等描述信息外，还有一项是常量池（Const Pool Table），用于存放编译期生成的各种字面量和符号引用。并非预置入Class文件中常量池的内容才进入方法运行时常量池，运行期间也可能将新的常量放入池中，这种特性被开发人员利用得比较多的便是String类的intern()方法。

Jdk7：运行时常量池被放到堆中

Jdk8：使用元空间取代永久代，也就是方法区，从而常量池在堆中。

##### string.intern()方法

“如果常量池中存在当前字符串, 就会直接返回当前字符串. 如果常量池中没有此字符串, 会将此字符串放入常量池中后, 再返回”。

Jdk6及以前：没有时是真的放入常量

Jdk7及以后，常量池在堆里，没有时是放入引用。

### 2）-xms,-xmx参数等

#### a)堆

Java堆溢出：通过不断的创建对象使堆溢出，java.lang.OutOfMemoryError:Java heap space;-Xms20M,-Xmx20M表示堆最小最大值为20M，避免自动扩展。

#### b)栈

虚拟机栈和本地方法栈：-Xss设置虚拟机栈，-Xoss设置本地方法栈（HotSpot的-Xoss无效）栈容量由-Xss决定。单线程下：不断调用某个方法while(true)，抛出StackOverflowError.只有在多线程下，不断创建线程，才会抛出OutOfMemoryError：unable to create native thread

#### c)方法区

方法区和运行时常量池溢出：-XX:PermSize=10M设置非堆区最小容量，-XX:MaxPermSize=10M设置最大容量。常量池溢出通过不断创建字符串常量：OutOfMemoryError:PermGen space;通过产生大量的类使方法区溢出

Jdk8以后采用-XX:MateSpaceSize=10M,…

### 3）垃圾收集算法

#### a)标记-清除算法：

标记清除的过程跟上边生存死亡一致。

当对象是不可达对象时，就被第一次标记，然后判断有没有必要执行finalize()方法，不执行的情况是没有覆盖finalize()方法，或者系统自动执行过一次（注意只一次）。对象可以在finalize()方法重新拯救对象，把它重新与任何一个对象关联起来。把this赋值给某个类变量等。不执行或者在finalize()不拯救，对象就被第二次标记，然后回收。

缺点：1.标记、清除效率都不高；2.空间问题，标记清楚后会导致内存碎片化，一旦要存储大的对象，很难有足够的连续空间

#### b)复制算法：

将可用内存按容量划分为一定比例的两块(1:1),每次只使用其中的一块。当这一块的内存用完时，就将还存活的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉。缺点：总有一块没有使用，优点：解决了空间不连续问题 （一般新生代采用这个算法）

#### c)标记-整理算法：

整理是让所有存活的对象都向一端移动，然后直接清理端边界以外的内存。

#### d)分段收集算法：

根据对象存活周期的不同将内存划分为几块。一般把java堆分为新生代和老年代。新生代采用复制算法。老年代采用标记-清除或者标记-整理。

### 4）GC Root搜索算法（可达性分析）中，root可以是？

1.虚拟机栈中的引用对象；2.方法区中类静态属性引用的对象；

3.方法区中常量引用的对象；4.本地方法栈中引用的对象

### 5）GC什么时候开始？

GC 经常发生的区域是堆区，堆区还可以细分为新生代、老年代，新生代还分为一个 Eden 区和两个 Survivor 区。

对象优先在 Eden 中分配，当 Eden 中没有足够空间时，虚拟机将发生一次 Minor GC，因为 Java 大多数对象都是朝生夕灭，所以 Minor GC 非常频繁，而且速度也很快；

Full GC，发生在老年代的 GC，当老年代没有足够的空间时即发生 Full GC，发生 Full GC 一般都会有一次 Minor GC。

大对象直接进入老年代，如很长的字符串数组，虚拟机提供一个；XX:PretenureSizeThreadhold 参数，令大于这个参数值的对象直接在老年代中分配，避免在 Eden 区和两个 Survivor 区发生大量的内存拷贝；

发生 Minor GC 时，虚拟机会检测之前每次晋升到老年代的平均大小是否大于老年代的剩余空间大小，如果大于，则进行一次 Full GC，如果小于，则查看 HandlePromotionFailure 设置是否允许担保失败，如果允许，那只会进行一次 Minor GC，如果不允许，则改为进行一次 Full GC。

### 6）内存泄漏和内存溢出

1.内存溢出指的是内存不够用了；

内存泄漏是指对象可达，但是没用了。即本该被 GC 回收的对象并没有被回收；

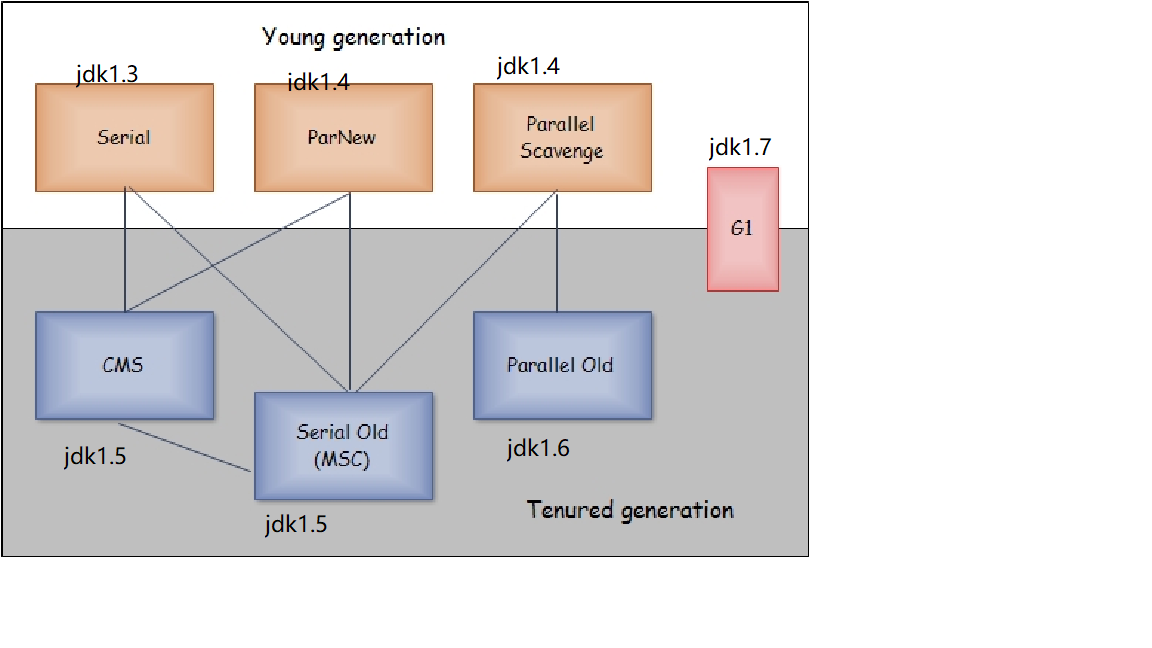
2.内存泄露是导致内存溢出的原因之一；内存泄露积累起来将导致内存溢出。

3.内存泄漏的原因分析：

长生命周期的对象引用短生命周期的对象；

没有将无用对象置为 null。

### 7）垃圾收集器



#### **Serial(年轻代）**

年轻代收集器，可以和Serial Old、CMS组合使用

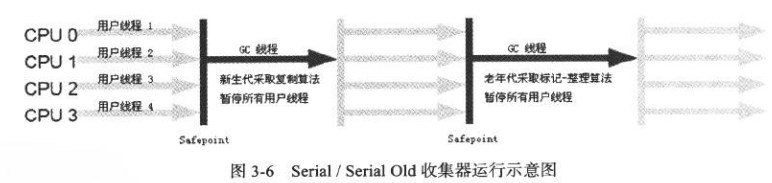
采用复制算法

使用单线程进行垃圾回收，回收时会导致Stop The World，用户进程停止

client模式年轻代默认算法

GC日志关键字：DefNew(Default New Generation)

图示（Serial+Serial Old)



#### ParNew(年轻代）

新生代收集器，可以和Serial Old、CMS组合使用

采用复制算法

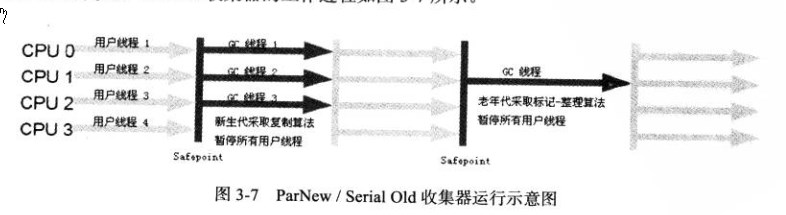
使用多线程进行垃圾回收，回收时会导致Stop The World，其它策略和Serial一样

server模式年轻代默认算法

使用-XX:ParallelGCthreads参数来限制垃圾回收的线程数

GC日志关键字：ParNew(Parallel New Generation)

图示（ParNew + Serail Old）



#### **Paralle Scavenge(年轻代）**

新生代收集器，可以和Serial Old、Parallel组合使用，不能和CMS组合使用

采用复制算法

使用多线程进行垃圾回收，回收时会导致Stop The World

关注系统吞吐量

-XX:MaxGCPauseMillis：设置大于0的毫秒数，收集器尽可能在该时间内完成垃圾回收

-XX:GCTimeRatio：大于0小于100的整数，即垃圾回收时间占总时间的比率，设置越小则希望垃圾回收所占时间越小，CPU能花更多的时间进行系统操作，提高吞吐量

-XX:UseAdaptiveSizePolicy：参数开关，启动后系统动态自适应调节各参数，如-Xmn、-XX：SurvivorRatio等参数，这是和ParNew收集器重要的区别

GC日志关键字：PSYoungGen

#### Serial Old(年老代）

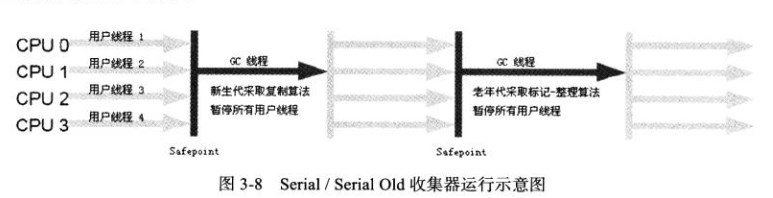
年老代收集器，可以和所有的年轻代收集器组合使用(Serial收集器的年老代版本）

采用 ”标记-整理“算法，会对垃圾回收导致的内存碎片进行整理

使用单线程进行垃圾回收，回收时会导致Stop The World，用户进程停止

GC日志关键字：Tenured

图示（Serial+Serial Old)



#### Parallel Old(年老代）

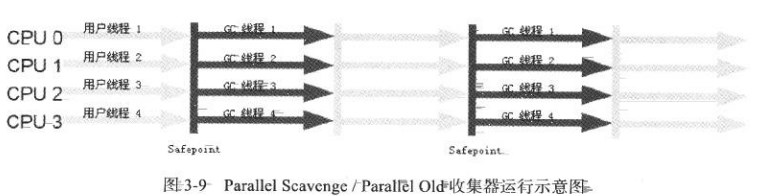
年老代收集器，只能和Parallel Scavenge组合使用(Parallel Scavenge收集器的年老代版本）

采用 ”标记-整理“算法，会对垃圾回收导致的内存碎片进行整理

关注吞吐量的系统可以将Parallel Scavenge+Parallel Old组合使用

GC日志关键字：ParOldGen

图示(Parallel Scavenge+Parallel Old)



#### CMS(Concurrent Mark Sweep年老代）

年老代收集器，可以和Serial、ParNew组合使用

采用 ”标记-清除“算法，可以通过设置参数在垃圾回收时进行内存碎片的整理

1、UserCMSCompactAtFullCollection：默认开启，FullGC时进行内存碎片整理，整理时用户进程需停止，即发生Stop The World

2、CMSFullGCsBeforeCompaction：设置执行多少次不压缩的Full GC后，执行一个带压缩的（默认为0，表示每次进入Full GC时都进行碎片整理）

CMS是并发算法，表示垃圾回收和用户进行同时进行，但是不是所有阶段都同时进行，在初始标记、重新标记阶段还是需要Stop the World。CMS垃圾回收分这四个阶段

1、初始标记（CMS Initial mark） Stop the World 仅仅标记一下GC Roots能直接关联到的对象，速度快

2、并发标记（CMS concurrent mark） 进行GC Roots Tracing，时间长，不发生用户进程停顿

3、重新标记（CMS remark） Stop the World 修正并发标记期间因用户程序继续运行导致标记变动的那一部分对象的标记记录，停顿时间较长，但远比并发标记时间短

4、并发清除（CMS concurrent sweep） 清除的同时用户进程会导致新的垃圾，时间长，不发生用户进程停顿

适合于对响应时间要求高的系统

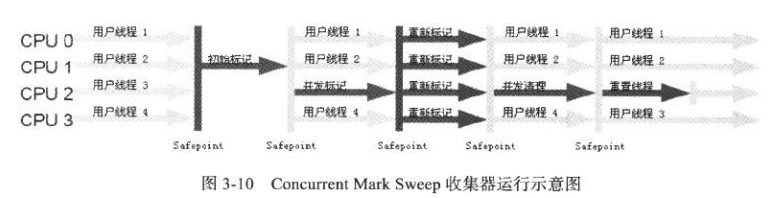
GC日志关键字：CMS-initial-mark、CMS-concurrent-mark-start、CMS-concurrent-mark、CMS-concurrent-preclean-start、CMS-concurrent-preclean、CMS-concurrent-sweep、CMS-concurrent-reset等等

缺点

1、对CPU资源非常敏感

2、CMS收集器无法处理浮动垃圾，即清除时用户进程同时产生的垃圾，只能等到下次GC时回收

3、因为是使用“标记-清除”算法，所以会产生大量碎片



# 前端

## CSS

### 概念

CSS 指层叠样式表 (Cascading Style Sheets)，是一种用来表现 HTML 文档样式的语言，样式定义如何显示 HTML 元素，是能够真正做到网页表现与结构分离的一种样式设计语言。样式通常存储在样式表中，外部样式表通常存储在 CSS 文件中，多个样式定义可层叠为一。

### 样式规则

“选择器“{属性1：属性值；属性2：属性值；…… }

选择器区分大小写，用于指定HTML元素，属性，属性值不区分大小写。

### CSS样式

#### a)行内样式/内联样式

在相关标签内使用样式（style）属性定义

<div style="width:200px;height:100px;border:1px solid black;"></div>

#### b)内部样式

需要使用 <style> 标签在文档头部<head>定义内部样式表

<head>

<style>

div{

width:200px;

height:100px;

border:1px solid black;

}

</style>

</head>

#### c)外部样式

使用 <link> 标签链接到外部样式表

<head>

<link rel="stylesheet" type="text/css" href="css/style.css">

</head>

#### d)样式优先级

内联样式 > 内部样式 > 外部样式

### CSS选择器

#### 元素选择器

选择器通常将是某个 HTML 元素，比如 body、h1、p、a，也可以是 html 本身。

#### id选择器

id 选择器可以为标有特定 id 的 HTML 元素指定特定的样式，id 属性和身份证一样具有唯一性。HTML元素以 id 属性来设置 id 选择器，CSS 中 id 选择器以 "#" 来定义。注意： id 属性不能以数字开头。

<head>

<style>

#heading{

color:red;

text-align:center;

}

</style>

</head>

#### 类选择器

class 选择器用于描述一组元素的样式，也叫做类选择器，class 选择器有别于 id 选择器，class 可以在多个元素中使用，并且一个元素也可以指定多个类名。class 选择器在 HTML 中以 class 属性表示，在 CSS 中，类选择器以一个点 "." 号来定义。可以设置所有带有指定 class 的 HTML 元素，也可以指定特定的 HTML 元素使用 class。注意：类名的第一个字符也不能使用数字。

<head>

<style>

.center{

text-align:center;

}

.col{

color:red;

}

.font{

font-size:18px;

font-family:"Microsoft YaHei";

}

</style>

</head>

#### 扩展：属性选择器

对带有指定属性的 HTML 元素设置样式。

input[type="text"]

{

width:150px;

display:block;

margin-bottom:10px;

background-color:yellow;

font-family: Verdana, Arial;

}

#### 扩展：包含选择器

h1 em {color:red;}

### CSS盒子模型

CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：边距，边框，填充，和实际内容。

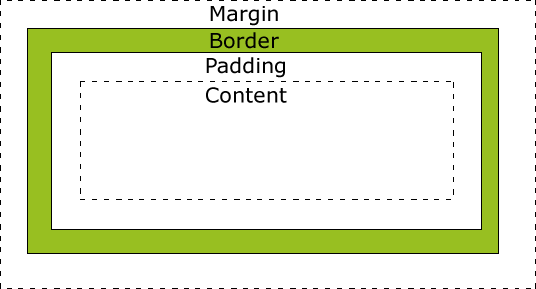
盒模型允许我们在其它元素和周围元素边框之间的空间放置元素。

Margin（外边距） - 清除边框区域。Margin没有背景颜色，它是完全透明

Border（边框） - 边框周围的填充和内容。边框是受到盒子的背景颜色影响

Padding（内边距） - 清除内容周围的区域。会受到框中填充的背景颜色影响

Content（内容） - 盒子的内容，显示文本和图像



## JavaScript

### 1）简介

JavaScript 是脚本语言

JavaScript 是一种轻量级的编程语言。

JavaScript 是可插入 HTML 页面的编程代码。

JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

### 2）引入方式

#### a)行内引入：安全度低

<div style="width: 300px;background: red; border-radius: 20px" onclick="alert('OK')">你好</div>

#### b)内嵌引入

写在script标签中

#### c)外部引入

在<script>标签加src属性

<script src="js/index.js">

alert('我是外链式标签块中的代码');//外链式标签块中不要写任何js代码，写了不会报错，肯定不执行

</script>

### 3）存放位置

HTML 中的脚本必须位于 <script> 与 </script> 标签之间。

脚本可被放置在 HTML 页面的 <body> 和 <head> 部分中。

由于Html代码是从上到下执行，如果Head中的js代码耗时严重，就会导致用户长时间无法看到页面，如果放置在body代码块底部，那么即使js代码耗时严重，也不会影响用户看到页面效果，只是js实现特效慢而已。

这里，推荐放在body体底部

### 4）输出方式

　　1、alert(要输出的内容); 在浏览器中弹出一个框，在框中有我们要输出的内容（不管最后输出什么内容，输出的都是字符串内容）

2、confirm：在浏览器中弹出一个提示确认框（confirm("确定要删除吗?")）

3、console.log(要输出的内容); 按F12在控制台中的console页卡中显示（一般用于调试，不会影响页面中的内容）

4、console.dir：他只是比.log输出的内容更加详细一些

5、console.table：他能把我们需要查看的数据在控制台中以一个表格的形式展示出来

6、document.write(我们要输出的内容); 直接显示在页面中，不停的输出

7、innerHTML/innerText 动态的向指定的元素中添加内容

### 5）组成

ECMAScript(定义了JS的基本语法,命名规范,操作语句,变量,数据类型等最基础最核心的知识)

DOM(document object model 文档对象模型) 提供了JS操作页面上元素的常用属性和方法

BOM(browser object model 浏览器对象模型) 提供了JS操作浏览器的常用属性和方法

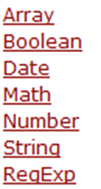
### 6)数据类型

Javascript数据类型分为原始数据类型和引用数据类型

原始数据类型：

string、number、boolean、null、undefined

引用数据类型：



## JS框架：Jquery

### 1）简介

Jquery它是javascript的一个轻量级框架，对javascript进行封装，它提供了很多方便的选择器。供你快速定位到需要操作的元素上面去。还提供了很多便捷的方法。

### 2）入门

#### a)引入Jquery

Jquery它是一个库(框架)，要想使用它，必须先引入！ 使用外联式

<script src="../../js/jquery-1.8.3.js"></script>

jquery-1.8.3.js：一般用于学习阶段。

jquery-1.8.3.min.js：用于项目使用阶段

#### b)简单入门

所有的jquery代码写在页面加载函数

$(function(){

Jquery代码

});

#### c)获取元素

JS: document.getElementById();

JQ: $(“#id”);

### 3）选择器

#### a)基本选择器

id选择器：$(“#id名称”);

元素选择器：$(“元素名称”);

类选择器：$(“.类名”);

通配符：\*

多个选择器共用(并集)

#### b)层级选择器

ancestor descendant: 在给定的祖先元素下匹配所有的后代元素(儿子、孙子、重孙子)

parent > child : 在给定的父元素下匹配所有的子元素(儿子)

prev + next: 匹配所有紧接在 prev 元素后的 next 元素(紧挨着的，同桌)

prev ~ siblings: 匹配 prev 元素之后的所有 siblings 元素(兄弟)

## 面试题

### 有哪些方式可以对一个DOM设置它的CSS样式？

外部样式表。引入一个外部CSS文件；

内部样式表。将CSS代码放在<head>标签内部；

内联样式，将CSS样式直接定义在HTML元素内部；

### CSS中可以通过哪些属性定义，使得一个DOM元素不显示在浏览器的可视范围内？

最基本的：

设置display属性为none，或者设置visiblity为hidden

技巧性：

设置高度为0， 设置透明度为0，设置z-index位置在-1000

### 盒子模型在不同浏览器上的区别

w3c盒子模型的范围包括margin、border、padding、content，并且content部分不包含其他部分

IE盒子模型的范围也包括margin、border、padding、content，和标准w3c盒子模型不同的是：ie盒子模型的content部分包含了border和padding。

### CSS属性是否区分大小写？

不区分，但选择器区分大小写

### 5)display:none与visible:hidden的区别

display:none ---不为被隐藏的对象保留其物理空间，即该对象在页面上彻底消失，通俗来说就是看不见也摸不到。

visible:hidden--- 使对象在网页上不可见，但该对象在网页上所占的空间没有改变，通俗来说就是看不见但摸得到。

他们修饰的图片依然会加载，除非是背景图片属性才不加载

### 6) 什么是JavaScript？

JavaScript是客户端和服务器端脚本语言，可以插入到HTML页面中，并且是目前较热门的Web开发语言。同时，JavaScript也是面向对象编程语言。

### 列举Java和JavaScript之间的区别？

Java是一门十分完整、成熟的编程语言。相比之下，JavaScript是一个可以被引入HTML页面的编程语言。这两种语言并不完全相互依赖，而是针对不同的意图而设计的。 Java是一种面向对象编程（OOPS）或结构化编程语言，类似的如C ++或C，而JavaScript是客户端脚本语言，它被称为非结构化编程。

### 7）什么是全局变量？这些变量如何声明，使用全局变量有哪些问题？

全局变量是整个代码长度可用的变量，也就是说这些变量没有任何作用域。var关键字用于声明局部变量或对象。如果省略var关键字，则声明一个全局变量。

例：// Declare a global globalVariable = “Test”;

使用全局变量所面临的问题是本地和全局变量名称的冲突。此外，很难调试和测试依赖于全局变量的代码。

### 解释JavaScript中定时器的工作？如果有，也可以说明使用定时器的缺点？

定时器用于在设定的时间执行一段代码，或者在给定的时间间隔内重复该代码。这通过使用函数setTimeout，setInterval和clearInterval来完成。

setTimeout（function，delay）函数用于启动在所述延迟之后调用特定功能的定时器。

setInterval（function，delay）函数用于在提到的延迟中重复执行给定的功能，只有在取消时才停止。

clearInterval（id）函数指示定时器停止

定时器在一个线程内运行，因此事件可能需要排队等待执行。

### 8）什么是===运算符？

===被称为严格等式运算符，当两个操作数具有相同的值而没有任何类型转换时，该运算符返回true。

# 后台Web

## tomcat

Web开发中常用的web应用服务器

1）weblogic：oracle公司的大型收费web服务器 支持全部javaEE规范

2）websphere：IBM公司的大型收费web服务器 支持全部的javaEE规范

3）Tomcat：Apache开源组织下的 开源免费的中小型的web应用服务器 支持 javaEE 中的 servlet 和 jsp规范

## servlet

### 简介

Servlet 运行在服务端的Java小程序，是sun公司提供一套规范（接口），用来处理客户端请求、响应给浏览器的动态资源。但servlet的实质就是java代码，通过java的API 动态的向客户端输出内容

servlet规范：包含三个技术点

1）servlet技术

2）filter技术---过滤器

3）listener技术---监听器

### 2）使用servlet

实现步骤：

1）创建类实现Servlet接口

2）覆盖尚未实现的方法---service方法

3）在web.xml进行servlet的配置

但在实际开发中，我们不会直接去实现Servlet接口，因为那样需要覆盖的方法太多， 我们一般创建类继承HttpServlet

实现步骤：

1）创建类继承HttpServlet类

2）覆盖doGet和doPost

3）在web.xml中进行servlet的配置

### 生命周期（面试题）

1）Servlet何时创建

默认第一次访问servlet时创建该对象

当在servlet的配置时 加上一个配置 <load-on-startup> servlet对象在服务器启动 时就创建

2）Servlet何时销毁

服务器关闭servlet就销毁了

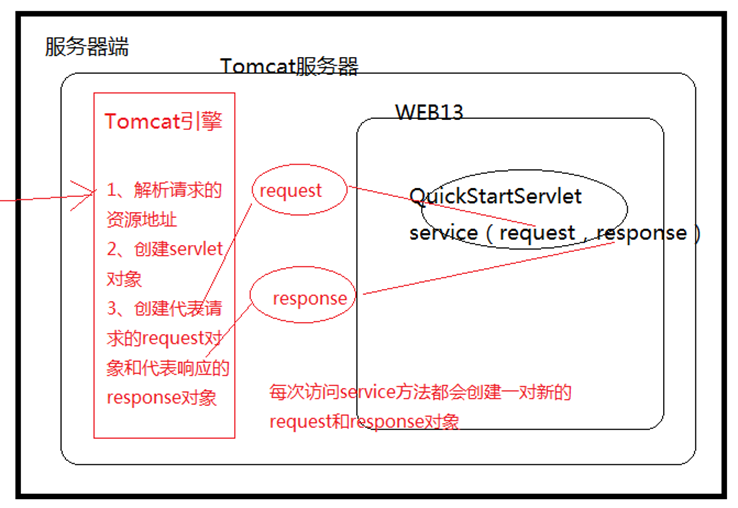
3）每次访问必然执行的方法

service(ServletRequest req, ServletResponse res)方法

对XXXServlet进行了10次访问，init()，destory()，service()，doGet()，doPost() 一共执行力多少次？request对象创建几个？response创建几个？

（1，0，10，x,x）(10,10)

### 3）访问过程



注意：servlet是**单例多线程**，系统采用多线程来处理多个Http请求

# 数据库

知识点：

面试题：

答案：

# 网络协议

## OSI，TCP/IP，五层协议的体系结构

### 结构

OSI分层（7层）：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

TCP/IP分层（4层）：网络接口层、网际层、运输层、应用层。

五层协议（5层）：物理层、数据链路层、网络层、运输层、应用层

### 2）作用

物理层：激活、维持、关闭通信端点之间的机械特性、电气特性、功能特性以及过程特性。**该层为上层协议提供了一个传输数据的物理媒体**。

数据链路层：**数据链路层在不可靠的物理介质上提供可靠的传输**。该层的作用包括：物理地址寻址、数据的成帧、流量控制、数据的检错、重发等。

网络层：**网络层负责对子网间的数据包进行路由选择**。此外，网络层还可以实现**拥塞控制**、网际互连等功能。

传输层：第一个端到端，即主机到主机的层次。**传输层负责将上层数据分段并提供端到端的、可靠的或不可靠的传输**。此外，传输层还要处理端到端的**差错控制和流量控制**问题。

会话层：**会话层管理主机之间的会话进程，即负责建立、管理、终止进程之间的会话**。会话层还利用在数据中插入校验点来实现数据的同步。

表示层：表示层对上层数据或信息进行变换以保证一个主机应用层信息可以被另一个主机的应用程序理解。**表示层的数据转换包括数据的加密、压缩、格式转换等**。

应用层：**为操作系统或网络应用程序提供访问网络服务的接口。**

### 3）层与设备

物理层：中继器（Repeater，也叫放大器），集线器。

数据链路层：网桥，交换机。

网络层：路由器。

网关：网络层以上的设备

### 4）每一层的协议

网络层：IP协议、ICMP协议、ARP协议、RARP协议。

传输层：UDP协议、TCP协议。

应用层：FTP（文件传送协议）、Telenet（远程登录协议）、DNS（域名解析协议）、SMTP（邮件传送协议），POP3协议（邮局协议），HTTP协议

## IP地址分类

### （1）网络地址

IP地址由网络号（包括子网号）和主机号组成，网络地址的主机号为全0，网络地址代表着整个网络。

### （2）广播地址

广播地址通常称为直接广播地址，是为了区分受限广播地址。

广播地址与网络地址的主机号正好相反，广播地址中，主机号为全1。当向某个网络的广播地址发送消息时，该网络内的所有主机都能收到该广播消息。

### （3）组播地址

D类地址就是组播地址。

先回忆下A，B，C，D类地址吧

A类地址以00开头，第一个字节作为网络号，地址范围为：0.0.0.0~127.255.255.255；

B类地址以10开头，前两个字节作为网络号，地址范围是：128.0.0.0~191.255.255.255;

C类地址以110开头，前三个字节作为网络号，地址范围是：192.0.0.0~223.255.255.255。

D类地址以1110开头，地址范围是224.0.0.0~239.255.255.255，D类地址作为组播地址（一对多的通信）；

E类地址以1111开头，地址范围是240.0.0.0~255.255.255.255，E类地址为保留地址，供以后使用。

Notice：只有**A,B,C有网络号和主机号之分，D类地址和E类地址没有划分网络号和主机号。**

### （4）255.255.255.255

该IP地址指的是受限的广播地址。受限广播地址与一般广播地址（直接广播地址）的区别在于，受限广播地址之只能用于本地网络，路由器不会转发以受限广播地址为目的地址的分组；一般广播地址既可在本地广播，也可跨网段广播。例如：主机192.168.1.1/30上的直接广播数据包后，另外一个网段192.168.1.5/30也能收到该数据报；若发送受限广播数据报，则不能收到。

Notice：一般的广播地址（直接广播地址）能够通过某些路由器（当然不是所有的路由器），而受限的广播地址不能通过路由器。

### （5）0.0.0.0

常用于寻找自己的IP地址，例如在我们的RARP，BOOTP和DHCP协议中，若某个未知IP地址的无盘机想要知道自己的IP地址，它就以255.255.255.255为目的地址，向本地范围（具体而言是被各个路由器屏蔽的范围内）的服务器发送IP请求分组。

### （6）回环地址

127.0.0.0/8被用作回环地址，回环地址表示本机的地址，常用于对本机的测试，用的最多的是127.0.0.1。

### （7）A、B、C类私有地址（局域网）

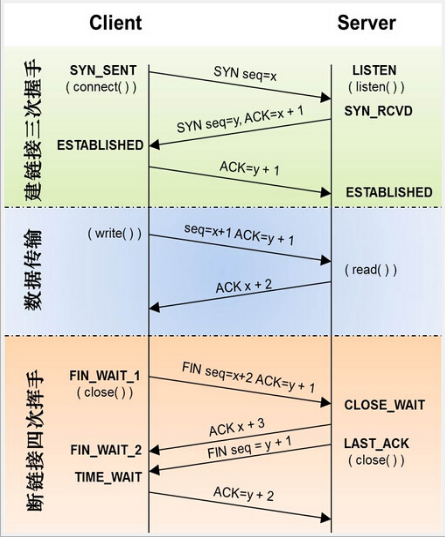
私有地址(private address)也叫专用地址，它们不会在全球使用，只具有本地意义。

A类私有地址：10.0.0.0/**8**，范围是：10.0.0.0~10.255.255.255

B类私有地址：172.16.0.0/**12**，范围是：172.16.0.0~172.31.255.255

C类私有地址：192.168.0.0/**16**，范围是：192.168.0.0~192.168.255.255

## 3.三次握手四次挥手



总共有四种状态：主动建立连接、主动断开连接、被动建立连和被动断开连接

两两组合还是 4 种组合：

主动建立连接、主动断开连接会经历的状态：

SYNC\_SENT——ESTABLISHED—-FIN\_WAIT\_1—-FIN\_WAIT\_2—-TIME\_WAIT

主动建立连接、被动断开连接会经历的状态：

SYNC\_SENT——ESTABLISHED—-CLOSE\_WAIT—-LAST\_ACK

被动建立连接、主动断开连接会经历的状态：

LISTEN—-SYN\_RCVD—-ESTABLISHED—-FIN\_WAIT\_1—-FIN\_WAIT\_2—-TIME\_WAIT

被动建立连接、被动断开连接会经历的状态：

LISTEN—-SYN\_RCVD—-ESTABLISHED—-CLOSE\_WAIT—-LAST\_ACK

# 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。