**一、Java基础**

1、面向对象和面向过程的区别

**面向过程**：结构化程序设计是用计算机的思维方式去处理问题，将数据结构和算法分离。数据结构描述待解决问题的数据的组织形式，而算法描述具体的解决步骤，我们用函数把这些步骤一步一步的实现，使用的时候一个一个的依次调用就可以了。

**优点**：性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源。

**缺点**：没有面向对象易维护、易复用、易扩展的优点。

**面向对象**：面向对象是把构成问题的各个要素抽象成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个要素在解决整个问题的步骤中的行为。

**优点**：易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护。

**缺点**：性能比面向过程低。

面向对象的三大特征及特点

**封装**：指的是将对象的属性和行为封装在类的内部，这些实现细节不能被外界直接访问，在类的内部会提供一系列接口来使用对象。由于封装，程序并不需要关心对象内部的具体算法或者实现细节，只需要操作对象对外暴露的接口，在一定程度上使代码更加容易维护。另外，由于封装，程序不能直接访问对象内部的私有数据，提高了安全性。

**继承**：指的是子类对象可以继承父类对象的属性和行为，因为在面向对象的编程中，并不是每一种对象都需要定义各自独有的属性和行为，有些对象的属性和行为可以从别的对象那里直接得到，从而提高代码的复用率。

**多态**：多态又分为静态多态和动态多态，静态多态通过重载体现，动态多态通过重写体现，同时动态的多态与继承具有密切的关系，在继承体系中，多个子类继承自同一个父类，并且各自重写了父类中的方法，通过父类引用调用子类对象的方法可以最大程度的利用接口，实现了一种行为，多种效果。

**总的来说，封装是为了将数据私有化并且隐藏实现细节，使得代码模块化并且提高安全系数，继承是为了扩展已有的模块，实现代码重用，多态的目的是保证类在继承或者派生的过程中能够正确的调用各自的方法。**

2、JDK和JRE的区别

**JDK**：(Java Development Kit) java开发工具包，提供了Java的开发环境和运行环境。

**JRE**：(Java Runtime Environment) java运行环境，为Java程序的运行提供所需环境。

JDK中包含了JRE，同时还有java程序的编译器Javac，还包含了Java的调试和分析工具。如果你需要运行Java程序，只需要安装JRE，如果你要编写Java程序，需要安装JDK。

3、==和equals的区别

**==**：对于基本类型和引用类型，==的效果是不一样的

**基本类型**：比较的是值是否一样

**引用类型**：比较的是引用是否相同

**equals**：本质上是对==的包装，但是在String，Integer等类在内部重写了equals方法，使它变成了值比较。

4、重载和重写的区别

**重载**：指在同一个类中的多个方法具有相同的名字,但这些方法具有不同的参数列表,即参数的数量或参数类型不能完全相同，重载是**静态多态性**的一种表现。调用方法时通过传递给它们的不同参数个数和参数类型来决定具体使用哪个方法，发生在编译时。

**重写**：父类与子类之间的多态性。如果子类中具有与父类中相同名称和参数的方法，就说这个方法在子类中被重写了，当子类继承了父类的方法但是却想对父类中的方法做一定修改时就要用到重写，重写是**动态多态性**的一种表现。调用方法时程序会通过对象的实际类型去匹配对应的方法，发生在运行时。

5、String和StringBuffer、StringBuilder的区别是什么？String为什么是不可变的？

**String对象不可变**：String对象内部是一个用final修饰的char类型数组，private final char value[]，所以只能初始化一次，并且不能再被修改。每次对String 类型进行改变的时候，都会生成一个新的String对象，然后将指针指向新的String 对象。

**线程安全性**：String中的对象是不可变的，也就可以理解为常量，是线程安全的。AbstractStringBuilder是StringBuilder与StringBuffer的父类，定义了一些字符串的基本操作。StringBuffer对方法加了同步锁，所以是线程安全的。StringBuilder并没有对方法进行加同步锁，所以是非线程安全的。但是StringBuilder的性能却略高于StringBuffer，所以一般在单线程环境下使用StringBuilder，多线程环境下使用StringBuffer。

6、static关键字是什么意思？static修饰的方法是否可以被重写？在static环境中是否可以访问非static变量？

Static修饰的变量或者方法称之为静态成员或者静态方法，它们随着类的加载就已经被编译了，所以不需要通过类的实例对象去访问，可以直接通过类名去访问。

Static修饰的方法在编译期间就已经静态绑定了，但是方法的重写执行的是运行时绑定，所以，对static方法重写是错误的。

Static变量或者方法时属于java类的，在所有类的对象中都是一样的，在jvm加载java类的时候，它会对类中的static变量进行编译，这个时候类的对象还没有被创建，变量也不存在，所以在static环境中使用非static变量是错误的。

7、什么是构造函数，构造函数可以重载吗？

构造函数的作用是对类进行初始化，当要创建类的实例对象时，构造函数就会被调用。每个类都有一个构造函数，如果没有显式的给出构造函数，编译器会给一个默认的构造函数。

构造函数可以像普通类方法一样重载，通过不同的参数列表，可以以不同的方式创建对象。

8、抽象类和接口

抽象类和接口的设计都是为了支持**多态**，

**抽象类**：包含抽象方法的类或者是用abstract修饰的类都是抽象类，类是对事物的抽象，既包括对属性的抽象，又包括对行为的抽象，但是对于某些行为，它无法给出具体的抽象过程，所以它就是一个抽象类。因此抽象类包含了一组未定义的方法，具体实现细节只能由子类去实现。

**接口**：接口是比抽象类更加抽象的抽象类，抽象类里面可以有非抽象方法，但是接口里面的方法全都是抽象方法，接口里面的方法自动声明为public，接口里面可以有成员变量，但是自动声明为public static final类型。

**区别**：

(1) 抽象类可以拥有自己的成员变量，可以拥有抽象方法和非抽象方法，但是接口只能拥有静态变量和抽象方法；

(2) 抽象类是对类抽象，包括事物的属性、行为，而接口是对行为的抽象，是对类局部（行为）进行抽象。

(3) 一个类只能继承一个抽象类，但是可以继承多个接口。抽象类和接口都不能实例化，但是抽象类可以有构造函数。由于抽象类的构造函数在实例化派生类之前发生，所以，可以在这个阶段初始化抽象类字段或执行其它与子类相关的代码。

6、Java中package的概念及作用

package是一种名称空间命名机制，它为java中的文件提供了一种管理方式，它采用树形结构把一组功能相似和相关的类文件或者接口文件组织到一起，不同包内的文件通过包名加以区别，避免了命名冲突。

Java中Exception和Error有什么区别

* Exception：是程序正常运行中，可以预料的意外情况，而且可以被捕获并进行相应处理，分为运行时异常(RuntimeException)和编译时异常。一般指应用程序级别的错误。
* Exception又分为可检查(checked)异常和不可检查(unchecked)异常。可检查异常在源代码里必须显式的进行捕获处理，这是编译期检查的一部分。不可检查异常是指运行时异常，不需要在方法上声明。
* Error：是指正常情况下，不大可能出现的情况，绝大部分的Error都会导致程序处于不可控、非正常的、不可恢复的状态。一般表示系统级别或者偏底层的错误。

常见的运行时异常：

NullPointerException - 空指针引用异常

ClassCastException - 类型强制转换异常。

IllegalArgumentException - 传递非法参数异常。

NumberFormatException - 数字格式异常

ArithmeticException - 算术运算异常

IndexOutOfBoundsException - 下标越界异常

throw和throws有什么区别

* throw是在程序中明确地抛出一个异常，一般是在代码块的内部，当程序可能出现某种逻辑错误时由程序员主动抛出某种特定类型的异常。
* throws用来在方法声明上抛出异常，当方法内部可能出现某种异常，在方法内部又处理不了这个异常，就在方法声明上显式的抛出这个异常，让方法的调用者捕捉异常并处理。

final, finally, finalize的区别

* final:修饰关键字，当final修饰class时，这个类不能被继承，当final修饰变量时，这个变量必须在定义的同时就赋予初始值，以后也只能读取使用，而不能修改；当final修饰方法时，这个方法不能被重写。
* finally：用来修饰代码块，在异常处理时，无论异常是否发生，finally代码块总是会被执行，在finally代码块里面主要进行一些资源的释放操作。
* finalize：是一个方法名称，它是在 Object 类中定义的，因此所有的类都继承了它。一般情况下，不会用到finalize()方法，在一些特殊情况下，当对象被回收之前需要释放一些资源，这个时候就需要实现finalize()方法，用来在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。一个对象的finalize()方法只会被调用一次。

什么是java序列化，如何实现java序列化？或者请解释Serializable接口的作用。

我们有时候将一个java对象变成字节流的形式传出去或者从一个字节流中恢复成一个java对象，例如，要将java对象存储到硬盘或者传送给网络上的其他计算机，这个过程我们可以自己写代码去把一个java对象变成某个格式的字节流再传输，但是，jre本身就提供了这种支持，我们可以调用OutputStream的writeObject方法来做，如果要让java 帮我们做，要被传输的对象必须实现serializable接口，这样，javac编译时就会进行特殊处理，编译的类才可以被writeObject方法操作，这就是所谓的序列化。需要被序列化的类必须实现Serializable接口，该接口是一个mini接口，其中没有需要实现的方法，implements Serializable只是为了标注该对象是可被序列化的。

例如，在web开发中，如果对象被保存在了Session中，tomcat在重启时要把Session对象序列化到硬盘，这个对象就必须实现Serializable接口。如果对象要经过分布式系统进行网络传输或通过rmi等远程调用，这就需要在网络上传输对象，被传输的对象就必须实现Serializable接口。

**二、Java集合**

Collection

├ List

│├ LinkedList

│├ ArrayList

│└ Vector

│ 　 └ Stack

├ Set

│ ├ HashSet

│ │ │ └ LinkedHashSet

│ │ └ TreeSet

└ Queue

* List特点：能够维护元素具有特定的顺序，允许有相同元素，用户可以控制元素插入的位置，也可以按照下标获取元素。
* LinkidList：底层是双向循环链表，元素有序，可重复，插入删除速度快
* ArrayList：底层实现是动态数组，元素排列有序，可重复，查找快，插入删除比较慢
* Vector：底层是动态数组，是线程安全的，因为它内部的方法加了同步
  + Stack：先进后出的(LIFO)数据结构
* Set特点：并不关心元素的特定顺序，只关心是否具有相同元素。
* HashSet：底层是基于哈希算法的哈希表，元素排列无序，不可重复，线程不安全
  + LinkedHashSet：底层是LinkedHashMap，并使用双向链表维护元素顺序
* TreeSet：底层是TreeMap，使用元素的自然顺序进行排序，或者是根据创建set时所选用的comparator方法进行排序
* Queue：先进先出的(FIFO)的数据结构

Map

├ Hashtable

├ HashMap

│├ LinkedHashMap

│└ WeakHashMap

├ TreeMap

└ IdentifyHashMap

Map特点：用来存储一组键值对，提供从key到value的映射

* HashMap：底层是基于哈希表的map接口实现。key不可重复，value可以，允许使用null值和null键，线程不安全。
* LinkedHashMap：可以认为是**HashMap+LinkedList，**既使用了HaspMap的数据结构，又使用了双向链表来维护插入元素的顺序。
* WeakHashMap：继承AbstractMap类，使用弱密钥的哈希表。
* HashTable：线程安全，不可以使用null值，其它方法跟HashMap类似。
* TreeMap：底层使用红黑二叉树，根据键的自然顺序进行排序，或者根据创建map时提供的 Comparator 进行排序，具体取决于使用的构造方法。

Collection和Collections有什么区别

Collection：是一个集合的接口，它提供了操作集合对象的通用方法。

Collections：是一个集合的工具类，封装了操作集合的一系列静态方法，不可以被实例化。

什么是迭代器(Iterator)，Iterator和ListIterator的区别是什么？

* Iterator接口提供了对集合元素进行迭代方法。每一个集合类都包含返回迭代器实例的方法，用来遍历集合中的元素。
* Iterator可以用来遍历set和list，而且只能对集合进行前向遍历，但是ListIterator只能遍历list，而且既可以向前也可以向后遍历。ListIterator实现了Iterator接口，并且增加了一些方法，增加、删除或者替换，获取前一个或者后一个元素。

hashCode()和equals()方法的区别

HashMap的工作原理是什么？

HashMap以键值对的形式存储元素，它的底层是基于数组和链表来实现的。在HashMap内部有一个hash函数，它使用hashCode()和equals()方法在集合中添加或者检索元素，当调用put()方法时，HashMap会先计算键值的hash值，然后把键值对放入到合适的索引上，如果key值已经存在了， value值会被更新。如果出现hash冲突，也就是两个键值的哈希值一样，HashMap通过链表来解决hash冲突，当链表的长度超过8时，使用红黑树来代替链表，因为红黑树的插入、删除、查找的最坏复杂度是O(logN)。当红黑树节点个数少于6时，又会转换为链表。因为HashMap每个桶中的节点个数遵循泊松分布，节点个数达到8的概率非常小，几乎是不可能事件，所以这个阈值选择为8. 之所以一开始不使用红黑树，是因为TreeNode节点是普通Node节点的两倍。

HashMap的重要特征是:

**容量(capacity)**: 容量是哈希表中桶的数量，初始容量只是哈希表在创建时的容量，默认是16。

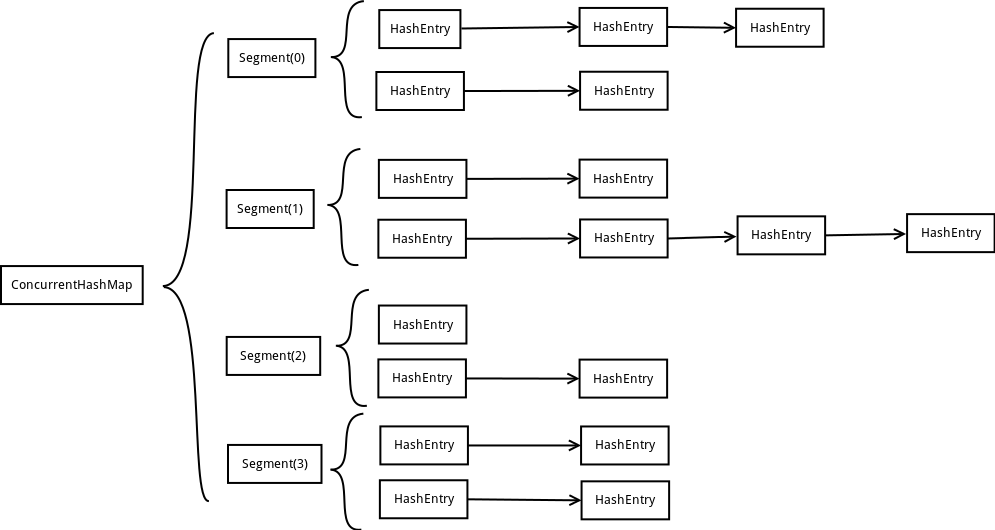
**负载因子(load factor)**: 是哈希表在其容量自动增加之前可以达到多满的一种尺度。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行 rehash 操作（即重建内部数据结构），从而哈希表将具有大约两倍的桶数。默认加载因子 (.75) 是时间和空间成本上的一种折衷。加载因子如果过高虽然减少了空间开销，但同时也增加了查询成本。加载因子如果太小，则空间不能充分利用。

HashMap和HashTable有什么区别

* Hashtable是线程安全的，多个线程可以共享一个Hashtable；但是HashMap不是线程安全的。
* HashMap可以接受为null的键值(key)和值(value)，而Hashtable则不行
* HashTable中的散列是：hash%length，这种方法基本能保证元素在哈希表中散列的比较均匀，但是取模会用到除法运算，效率很低。HashMap中的散列是hash&(length-1)，因为哈希表的容量是2的n次方，所以hash&(length-1)= hash%length，但效率要高很多，这也是HashMap对Hashtable的一个改进。

HashTable和ConcurrentHashMap的区别

HashTable的默认长度是11，负载因子是0.75，当达到阈值后，扩容公式为new=old\*2+1。它的底层是数组加链表，它内部的方法都是同步的，所以它是线程安全的，当一个线程正在操作HashTable时，它会锁住整个数组，这意味着其它线程只能等待当前线程操作完以后再继续操作HashTable，在高并发的情况下这种方式的效率时非常差的。但是ConcurrentHashMap不一样，它使用了更加细粒化的锁来提高并发效率，ConcurrentHashMap默认将Hash表分为16个Segement，Segement内部是一个链表数组(类似于桶，每一个桶可以被看作是一个HashTable），ConcurrentHashMap定位一个元素需要两次hash散列，第一次是找到对应的Segement，也就是对应的桶，第二次是找到元素所在链表的头部，比起HashMap或者HashTable，虽然元素定位的时间变长了，但是好处却是操作元素只在当前Segement进行，所以也只需要对当前Segement加锁就可以，其它的线程依然可以其它的Segement，所以，多个线程可以同时访问Map，提高了并发效率。只有用到个别方法如size()时，才需要对所有的Segement加锁。



ConcurrentHashMap的结构

ArrayList和LinkedList的区别是什么？

它们都实现了 List 接口，不同点如下：

* ArrayList：是基于索引的数据接口，它的底层是**动态数组**。它可以以 O(1)时间复杂度对元素进行随机访问。
* LinkedList：是以**双向链表**的形式存储数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是 O(n)。
* 相对于 ArrayList，LinkedList 的插入，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。
* LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

ArrayList和Vector的区别是什么？

* Vector使用了线程同步，所以是线程安全的，而Arraylist是线程不安全的，
* 两者都会根据实际需求动态扩容，但是Vector每次扩容会增加1倍，但是Arraylist只会扩充50%。

Comparable和Comparator 接口有什么区别

Comparable接口里面只有一个compareTo()方法，该方法只有一个参数，用来比较当前对象和指定对象的顺序，返回结果负数、0和正数分别代表当前对象小于、等于和大于指定对象。

Comparator接口里面有两个方法,compare(T o1,T o2)和equals(),compare()方法接收两个参数，用来比较这两个参数的顺序。

Comparable:内部比较器，给对象一个默认的比较规则，让它具有比较的能力，如果要使用Collections或者Arrays的sort()方法，就必须实现Comparable接口。

Comparator:外部比较器，如果不想让对象按照默认的规则进行排序，或者是对象没有比较规则，就可以使用Comparator接口，它不需要改变类原有的结构。

**三、Java多线程**

1、进程、线程和多线程的概念

**并行**：多个处理器同时执行多个任务。

**并发**：多个任务在一个CPU上执行，CPU通过时间的划分交替轮流地执行各个任务。

**进程**：进程指的是程序的一次执行过程，是操作系统最基本的分配单元，也是运行程序的基本单元。每个进程都有自己独立的内存空间，一般包括文本区，代码区和它内部的线程可访问的各种资源。

**线程**：CPU进行资源调度的最小单元，是一种轻量级的进程，一个进程可以有多个线程并且至少有一个主线程，线程的执行依赖进程内部的资源。

**多线程**：指的是在一个进程内部有多个线程在几乎同时工作，但它们并不是真正的同时工作，因为单核CPU在一个时刻只能执行一个计算任务，但是CPU可以把时间分成很多个细小的片段，让每个线程在不同的时间片段上执行计算任务，因为每个时间片段都很短暂，所以宏观上看起来就像是这些线程在同时执行一样。

2、为什么要使用多线程？

使用多线程不是为了提高程序的运行速度，而是更加高效地使用程序。因为在多线程程序中，多个线程被并发的执行以提高程序的效率，耗时长的任务可以被放到后台去处理，CPU不会因为某个线程需要等待资源而进入空闲状态。

多线程安全问题

原因：当多个线程同时操作共享数据时，就会导致错误的执行结果。

解决方法：使用同步方法，将操作共享数据的语句在某一时刻只让一个线程执行，在当前线程操作共享数据的过程中，其它线程只能处于阻塞状态，不能操作共享数据。同步方法如下：

⑴ 同步代码块：通过锁定一个指定的对象，来对同步块中包含的代码进行同步

synchronized(对象){ //任意对象都可以，这个对象就是锁

同步代码;

}

有两个或者更多的线程的时候才需要同步，而且多个线程必须使用同一把锁。

⑵ 同步函数：对整个方法体里面的代码进行同步。

将同步关键字定义在函数上，让函数具备了同步性。类的普通成员函数拥有自己所属的对象的this，所以同步函数的锁就是this对象，静态函数在加载时是属于类的，这时候还没有对象被创建，但是这个类的class对象已经加载进内存了，所以静态函数的锁是类的class对象。

3、创建线程有哪几种方式？

* 继承Thread类重写run()方法
* 实现Runnable接口创建线程
* 使用Callable和FutureTask创建线程

**继承Thread和实现Runnable接口的区别**

* 实现Runnable接口避免多继承局限，线程类只是实现了Runnable接口或Callable接口，还可以继承其他类。
* 实现Runnable()可以更好的体现共享的概念，在这种方式下，多个线程可以共享同一个target对象，所以非常适合多个相同线程(Thread)来处理同一份资源(子类)的情况，体现了面向对象的思想。

Runnable和Callable的区别

Runnable没有返回值，Callable可以拿到返回值，call()方法可以抛出异常，但是run()方法却不可以，Callable可以看作是对Runnable的补充。

4、Java中线程有哪些状态？

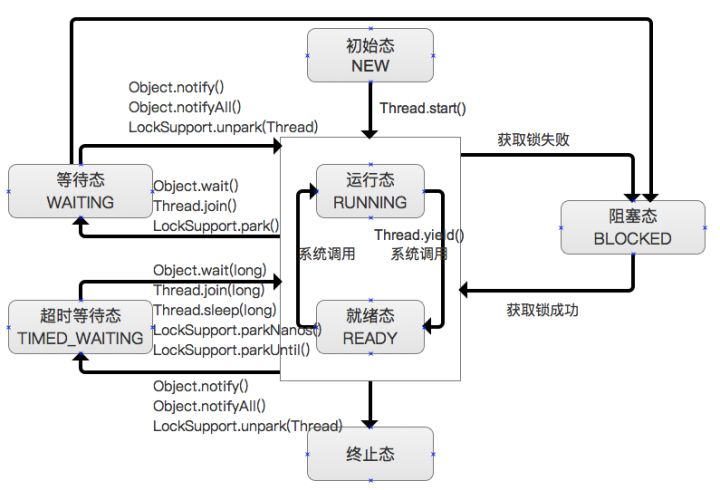
* **NEW(初始化状态)**：新创建了一个线程，但是还没有调用start()方法启动线程。
* **RUNNABLE (就绪状态、运行状态)**
* **就绪状态**

1. 该状态下的线程已经获得执行所需的所有资源，只要CPU分配执行权就能运行。
2. 所有就绪态的线程存放在就绪队列中。

* **运行状态**

1. 处于就绪状态的线程一旦获得CPU执行权，就处于运行状态，表示正在执行的线程。
2. 由于一个CPU同一时刻只能执行一条线程，因此每个CPU每个时刻只有一条运行状态态的线程。

* **BLOKED (阻塞状态)**
* 当一条正在执行的线程在请求某一资源失败时，就会进入阻塞状态。
* 有一个阻塞队列存放所有处于阻塞状态的线程。
* 处于阻塞态的线程会不断请求资源，一旦请求成功，就会进入就绪队列，等待执行。
* **WAITING (等待状态)**
* 当前线程中如果调用wait()、join()、park()方法时，当前线程就会主动进入等待状态。
* 有一个等待队列存放所有等待状态的线程。
* 线程处于等待状态表示它需要等待其他线程的指示才能继续运行。
* 进入等待状态的线程会释放CPU执行权，并释放资源（如：锁）
* **TIMED\_WAITING (定时等待状态)**
* 和等待状态一样，并不是因为请求不到资源，而是主动进入，并且进入后需要其他线程唤醒；
* 进入该状态后释放CPU执行权和占有的资源。
* 与等待状态的区别：到了超时时间后自动进入阻塞队列，开始竞争锁。
* **DEAD(终止状态)**：线程执行结束以后的状态。



注意

* wait()方法是Object类的方法，调用此方法会使当前线程释放CPU使用权和占有的锁对象，当前线程进入等待状态，当调用notify()方法或者notifyAll()方法时，才能唤醒该线程，并让它进入阻塞状态，准备获取锁对象进入就绪状态。
* sleep()方法是Thread类的静态方法，调用此方法会让当前线程暂停指定的时间，在这段时间内，当前线程仅释放CPU使用权，锁对象仍然占用；当前线程被放入定时等待队列，待时间结束以后，当前线程回到就绪状态，准备再次获取CPU使用权。
* yield()方法是让当前处于运行状态的线程暂停执行，回到就绪状态。目的是让具有相同优先级的线程之间能够适当的轮换执行。但有可能没有效果，因为做出让步的线程有可能被很快的再次选中执行。
* notify()：唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且与优先级无关；
* notifyAll()：唤醒 所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态；
* wait()方法和notify()必须配套使用，而且必须放在一个同步块中，并且使用同一把锁调用。

守护线程是什么

Java线程分为两种

用户线程：就是应用程序里的自定义线程。

守护线程：在程序运行时，在后台为用户线程提供服务的线程。比如垃圾回收线程就是一个守护线程。当所有用户线程结束时，程序也就终止了，进程内的守护线程也随之退出。

当一个线程进入一个对象的synchronized方法A之后，其它线程是否可进入此对象的synchronized方法B？

其它线程不能进入此对象的同步方法，但是非同步方法可以执行，因为成员方法的同步操作是以对象作为锁，当线程在执行对象的同步方法时，说明当前锁对象已经属于这个线程，其它线程由于无法获取到锁对象只能处于阻塞状态。

synchronized和Lock有什么区别

Lock是一个接口，实现Lock接口的类具有和synchronized关键字相同的功能，但是它的功能更加强大。该接口定义中定义的lock()方法用来锁定对象，unlock()方法用来释放锁对象，处于这两个方法之间的代码跟synchronized修饰的代码块的效果是一样的。

不同的是，使用synchronized关键字实现的同步，会把一个对象的所有同步方法或者同步块当做一个整体进行锁定，只要有一个同步方法被某个线程，那么其它的同步方法就不能被其它线程执行，即使这两个同步方法之间没有逻辑关系，这样的同步方式降低了程序的执行效率。如果使用Lock，可以将对象中具有逻辑关系的代码分组，为每一个组创建一个Lock对象，对对象中的同步方法分别进行锁定。这样，当一个线程在执行该对象的一个同步代码块时，别的线程可以执行该对象内部的其它同步代码块。

Synchronized可以给类、方法、代码块加锁，而且不需要手动的获取和释放锁，Lock只能给代码块加锁，需要自己加锁和释放。

Java中volatile关键字的作用

可见性：指的是线程之间的可见性，一个线程修改的状态对另外一个线程是可见的，为了保证多个内存写入操作的可见性，必须使用同步机制。

原子性：一个操作不能再被分割，就说这个操作是原子操作。例如，int a = 0；这个操作是不能再被分割的，所以是一个原子操作。再比如，a++，这个指令是a=a+1，它包含两步操作，所以它是一个非原子操作，非原子操作都存在线程安全问题，需要使用同步方法将它变为一个原子操作。

有序性（指令重排序）：

在执行程序时为了提高性能，编译器和处理器常常会对指令做重排序。重排序分三种类型：

* 编译器优化的重排序

编译器在不改变单线程程序语义的前提下（代码中不包含synchronized关键字），可以重新安排语句的执行顺序。

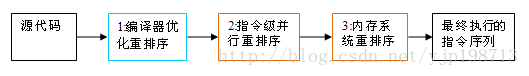
* 指令级并行的重排序

现代处理器采用了指令级并行技术（Instruction-Level Parallelism， ILP）来将多条指令重叠执行。如果不存在数据依赖性，处理器可以改变语句对应机器指令的执行顺序。

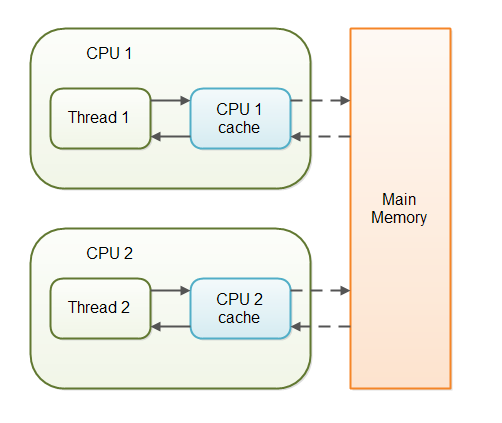
* 内存系统的重排序。

由于处理器使用缓存和读/写缓冲区，这使得加载和存储操作看上去可能是在乱序执行。

从java源代码到最终实际执行的指令序列，会分别经历下面三种重排序：



volatile原理



volatile变量提供了一种比较弱的同步机理，用来保证将变量的更新操作通知给其它线程。当一个变量被声明为volatile类型后，该变量具有可见性，也不允许线程内部缓存和重排序，也就是直接从内存读取变量或者直接在内存上修改该变量。因此在读取volatile类型的变量时总是会获得最新写入的值。当变量被定义为volatile类型后，会具有两种特性：

* 保证对所有线程的可见性：当一个线程修改了这个变量以后，这个新值被马上同步到主内存上，其它线程使用时会直接从主内存上获取这个新值。
* 禁止指令重排序优化：

线程池的使用

如果并发的线程数量特别多，并且每个线程的执行时间都特别短，这样频繁地创建和销毁线程很消耗系统资源，另外一个原因是，如果JVM里面创建的线程数量过多会导致系统内存消耗过度，因此为了防止系统资源被过度消耗，就出现了线程池。

线程池是一个线程管理器，在线程池里面维护了一定数量的活动线程，当有请求到达服务器时，服务器直接从线程池里面取出一个线程去执行请求，如果当前请求数量大于线程池中的线程数量，则会将任务添加到一个任务队列里面，当线程执行完别的请求时，它就会从任务队列里面取出请求任务继续执行。

线程池的好处：

* 降低系统的资源消耗，通过线程复用，服务器避免了频繁的创建和销毁线程，从而减小了系统的资源消耗。
* 提高了系统的运行效率，当任务请求到达服务器时，不需要等待线程创建就会被立即执行。
* 可以更好的管理多个线程，利用线程池可以对多个并发线程进行统一的管理和分配，提高了线程的可维护性。

线程池中execute()方法和submit()方法有什么区别

* execute():只能执行Runnable类型的任务，适用于不需要关注返回值的场景，只需要将线程丢到线程池中去执行就可以了
* submit():可以执行Runnable和Callable类型的任务，适用于需要关注返回值的场景

四、JVM

JVM运行时的数据区

程序计数器(Program counter Register)[线程私有]:一块很小的内存空间，里面存储了当前线程所要执行的下一条字节码指令的地址，虚拟机里面每个线程都拥有各自的程序计数器。

Java虚拟机栈(JVM Stack) [线程私有]:每个线程在调用java方法时，会为每个方法生成一个栈帧，里面存储局部变量表，操作数等信息。每一个方法的调用和结束对应一个栈帧在虚拟机栈中出栈和入栈的过程。

本地方法栈(Native Method Stack) [线程私有]:功能和Java虚拟机栈类似，用来存储线程在调用本地方法时的局部变量表等信息。

堆(Java Heap) [所有线程共享]:Java虚拟机中内存最大的一块区域，被所有的线程共享，主要用来存储对象实例，几乎所有的对象都是在这里分配的。

方法区(Method Area) [所有线程共享]:用来存储已经被JVM加载的类信息、常量、静态变量等数据。

Java中的引用类型

强引用：垃圾回收时不会被回收的对象。

软引用：有用但不是必须的对象，如果内存要溢出了，它就会被回收。

弱引用：有用但不是必须的对象，在下一次垃圾回收时被回收。

虚引用：和没有引用的效果是一样的，随时都有可能被垃圾回收器回收。它的用途是回收时会返回一个通知，用来跟踪对象的回收过程。

如何判断一个对象是否可以被回收了

有两种方法可以用来判断对象是否存活。

引用计数法：给对象中添加一个引用计数器，每当有一个地方引用它时，计数器就加1，当引用失效时，计数器就减1。任何时刻计数器都为0的对象是不可能在被使用的，代表这个对象已经死亡，是可以被回收的。

但是引用计数算法无法对象之间循环引用的问题。

可达性分析：通过一系列的“GC Roots”的对象作为起始点，向下搜索，搜索过的路径称之为“引用链”，当一个对象到“GC Roots”没有任何引用路径时，则证明这个对象时不可用的，可以被回收了。

GC Roots:JVM虚拟机栈中局部变量表总引用的对象，方法区中类的静态属性引用的对象，以及常量引用的对象。

对象的创建过程

java虚拟机在创建一个对象时，首先查看对象所属的类文件是否被加载到内存中，如果没有，jvm首先根据类的全限定名加载类文件，类的加载和初始化完成以后，就开始创建对象。

* 在堆区开辟一块内存，包括本类和父类的所有实例变量，记录内存的首地址；
* 对所有的实例变量执行默认的初始化，基本类型的变量被设为默认值，引用类型的变量被设为null；
* 调用对象的构造函数，先执行父类的构造函数，然后从上往下依次执行子类的构造函数；
* 将对象的首地址赋给栈中的引用变量，栈内的变量就指向了堆中的对象实体。

类加载过程

* 加载：根据全类名获取类的二进制字节流文件，将字节流文件从class文件的数据结构转换为JVM方法区内的数据结构。
* 验证：检验二进制class文件是否符合class文件格式的定义，检验是否符合java语言规范，检验程序的执行流程是否符合逻辑，检验符号引用是否正确。
* 准备：开始进行对类中的类变量第一次设置初始值，初始值指的是数据类型的零值，如果变量是用static final修饰的除外，它会被设置为它原有的值。
* 解析：将常量池中的符号引用转换为直接引用。符号引用指的是用来描述目标的符号，符号代表的是常量池中用来描述方法、字段的字面量。直接引用指的是直接定位目标的指针或者偏移量，也就是目标在方法区里面的入口地址。
* 初始化：执行类构造器方法的过程，按照程序员的意愿设置类变量的初值。

JVM有哪些垃圾回收算法

标记-清理算法：标记没有用的对象，然后清理回收这部分空间。缺点是：效率不高，无法清除垃圾碎片。

标记-整理算法：标记没有用的对象，让所有存活的对象都向一端移动，然后直接清除端边界以外的空间。

复制算法：将内存分成两部分区域，每次对象分配时只使用其中一块区域，垃圾回收时，将存活着的对象移动到另外一块区域，然后将原来的区域整体清理回收。缺点是：每次只使用一半内存空间，内存使用率不高。

分代算法:根据对象的存活周期将内存区域划分为几块区域，在不同的区域上使用不同的回收算法，相当于是前面几种算法的综合。

Minor GC与Full GC分别在什么时候发生

Minor GC：是新生代的垃圾回收动作，当JVM无法为一个新的对象分配内存空间，比如Eden区满了，就会触发一次Minor GC。发生Minor GC时，对象只会存在于Eden区和FromSurvivor区域，ToSurvivor区是空的，JVM会将Eden和FromSurvivor区域里的存活对象复制到ToSurvivor区域，然后清空Eden和FromSurvivor区域，To和From区域角色互换。

Full GC：老年代的对象都比较稳定，所以FullGC不会频繁执行，在进行一次MinorGC后，如果有新生代的对象进入老年代导致空间不够才会触发FullGC。如果无法找到足够大的连续空间分配给新创建的较大的对象也会触发FullGC。

双亲委派模型的原理及其优点

类加载器用来实现类的加载过程，在java中区分一个类用的是类的全限定名，而在JVM中，只有一个类的全限定名和它的类加载器才能确定它的唯一性。

原理：当一个类加载器收到一个加载请求时，它自己先不进行加载，而是把请求提交给它的父类加载器，如果父类加载器还有父类加载器就依此向上提交。因此，所有的加载请求最终都会到达顶层的启动类加载器，如果父类加载器可以完成加载请求，就成功返回，如果父类无法完成加载请求，子类加载器才会尝试去加载，这就是双亲委派模型。

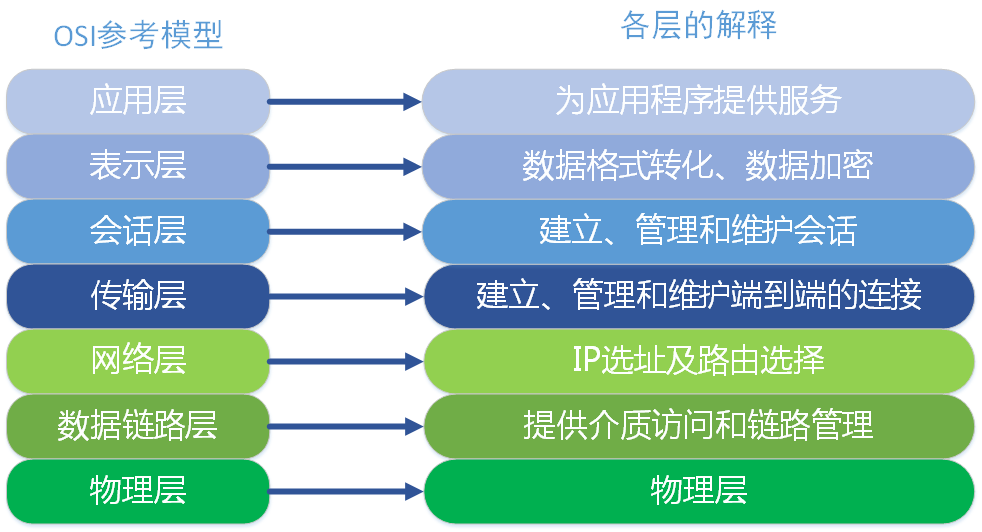
优点：双亲委派模型使得类的加载有了层次优先级，越核心的类由越顶层的类加载器去加载，通过这种层次可以确保类不会被重复加载，因为JVM中类的唯一性是由类的全限定名和它的类加载器唯一确定的。另外一个优点是防止核心类库被随意篡改，提高了安全性。

五、数据结构

六、网络通信

OSI(Open System Interconnection)模型

OSI是一种技术标准，它试图为计算机网络互连提供一个技术框架。OSI模型将计算机网络体系结构分为七层：



应用层：用户的应用程序和网络之间的接口，负责为用户直接提供各种网络服务，例如：HTTP协议。

表示层：表示层用于将应用层的数据进行编码和转换功能，

会话层：负责建立、管理和终止表示层实体之间的通信会话。

传输层：建立了主机端到端的链接，为上层协议提供端到端的数据传输服务。

网络层：通过IP寻址来建立两个节点之间的连接，选择正确的路由，将数据传输到目的端。

数据链路层：将字节数据分帧,使用链路层地址 (以太网使用MAC地址)来访问介质

物理层：通过物理介质传输比特流。

TCP/IP五层协议

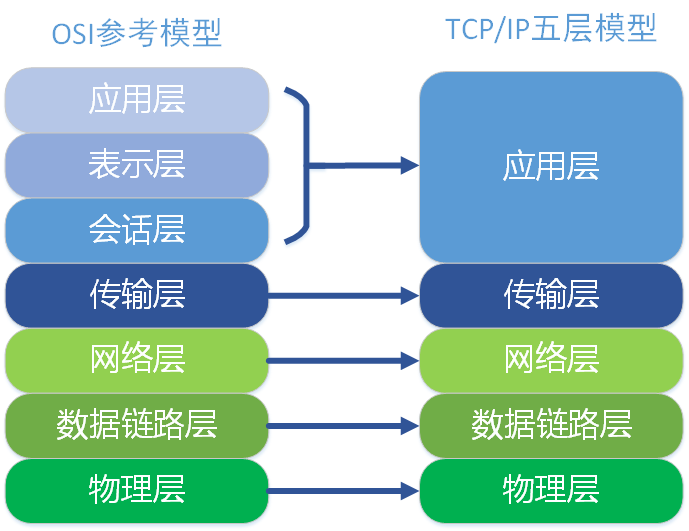
应用层：用户的应用程序和网络之间的接口，负责为用户直接提供各种网络服务，例如：HTTP协议。

传输层：建立了主机端到端的链接，为上层协议提供端到端的数据传输服务。

网络层：通过IP寻址来建立两个节点之间的连接，选择正确的路由，将数据传输到目的端。

数据链路层：将字节数据分帧,使用链路层地址 (以太网使用MAC地址)来访问介质

物理层：通过物理介质传输比特流。



TCP与UDP的区别

TCP(Transmission Control Protocol)：传输控制协议

UDP(User Data Protocol)：用户数据报协议

特点：

* TCP是面向连接的协议，在收发数据前，必须通过三次握手和对方建立可靠的连接。而UDP是无连接的，传输数据之前双方不需要建立连接。
* 使用TCP协议时，数据的传输过程是受到控制的，可以实现丢包重发，序列检测并且重新整理，实现了可靠性传输。使用UDP协议时，传输速度只与应用程序生成数据的速度，计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段，因此使用UDP有可能发生丢包问题。
* TCP是基于字节流的传输方式，UDP是基于数据报的传输方式。
* TCP头部的字节有20个，UDP只有8个。使用TCP需要的系统资源比较多。

应用场景：

UDP适用于对数据安全性没有特殊要求，要求实时性比较高的场合，例如直播、游戏之类的，因为UDP不需要在双方之间建立连接，所以在拥有大量client的时候，可以使用UDP。

TCP适用于需要可靠传输，而且对安全性要求比较高的场合。

TCP的三次握手过程：

（1）客户端向服务器发送SYN同步信号包，包里携带客户端的初识序列号seq=x，此时客户端处于等待连接状态。

（2）服务器收到客户端的SYN包以后，一方面对客户端发来的包进行确认反馈，反馈值ack=x+1，另一方面，服务器也会发出自己的syn同步信号，也有一个初始序列seq=y，这个过程是SYN+ACK。

（3）客户端收到服务器的SYN+ACK包以后，再次向服务器发出ACK确认包，反馈值ack=y+1，客户端发出这个包以后，标志双方已经建立了连接。

TCP断开时的四次握手过程：

由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这个原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。

（1）客户端A发送一个FIN，用来关闭客户A到服务器B的数据传送。

（2）服务器B收到这个FIN，它发回一个ACK，确认序号为收到的序号加1。和SYN一样，一个FIN将占用一个序号。

（3）服务器B关闭与客户端A的连接，发送一个FIN给客户端A。

（4）客户端A发回ACK报文确认，并将确认序号设置为收到序号加1。

浏览器与服务器建立了一个TCP连接后是否会在一个HTTP请求完成后断开？什么情况下会断开？

在HTTP/1.0中，一个服务器在发送完一个HTTP响应后，会断开TCP连接。但是这样每次请求都会重新建立和断开TCP连接，代价过大。所以某些服务器对Connection：keep-alive的Header进行了支持。意思是当完成这个HTTP请求之后，不要断开对应的TCP连接。这样的好处是TCP连接可以被重新使用，之后发送HTTP请求的时候不需要重新简历TCP连接。

在HTTP/1.1中，Connection头部被写进了标准，并且默认是keep-alive，就是不会断开TCP连接，如果在请求报头中声明将Connection属性设置为close，才会在请求完成后关闭连接。

一个TCP连接中HTTP请求可以同时发送吗（比如同时发送三个HTTP请求，再同时接收三个响应）？

* HTTP/1.1中，单个TCP连接在同一时刻只能处理一个请求，也就是一个HTTP请求从开始到结束的生命周期里，其它HTTP请求不能插入其中，它们的生命周期不能重叠。虽然HTTP/1.1中规定了Pipelining来试图让一个TCP连接同时处理多个HTTP请求，但是这个功能在浏览器里面默认是关闭的。
* 但是，在HTTP/2.0中，提供了Multplexing多路传输技术，使得多个HTTP请求可以在一个TCP连接中同时进行。
* 所以，在HTTP/1.1中，主要通过以下方式提高页面加载效率
* 维持和服务器已经建立的TCP连接，在同一连接上按顺序执行多个HTTP请求
* 和服务器建立多个TCP连接。（chrome可以同时建立6个TCP连接）

TCP为什么是可靠连接

* TCP协议在传输数据之前首先通过三次握手机制在双方之间建立了连接，双方一直在维护这个连接。
* TCP报头里面有一个序列号可以保证数据不丢失，不重复，按序到达。
* TCP报头里面还有一个ACK确认信号能够保证累积确认和超时重传。
* TCP具有流量控制和拥塞控制的机制以保证数据的可靠传输。

TCP中的流量控制和拥塞控制

流量控制：

接收方接收到TCP数据以后会先将数据放在缓冲区中，然后在缓冲区内将这些数据进行拼装再交给应用程序，如果数据到达缓冲区的速率大于处理这些数据的速率，就会导致缓冲区溢出，这样后续传过来的数据就会丢失。

使用流量控制是为了防止发送方的发送速度过快而导致接收方的缓冲区溢出。

TCP采用滑动窗口来控制发送方的数据发送操作，接收端发送ACK的时候会带上缓冲区的窗口大小，当应用程序从缓冲区取出数据以后，这个窗口就需要进行更新。

拥塞控制：

当传输网络比较繁忙，ACK信号的等待时间就会变长。一般来说，当对网络中的某些资源的需求超过了该资源所能提供的最大容量就会发生网络拥塞。

使用拥塞控制是为了防止过多的数据注入到网络中，这样可以使网络中的路由器或者链路不至于过载。

发送方维护一个拥塞窗口，这个窗口是动态变化的，窗口的大小取决于网络的拥塞情况，发送方的发送窗口是拥塞窗口和滑动窗口的较小值。拥塞控制主要体现在：



慢启动：主机开发发送数据报时，如果立即将大量的数据注入到网络中，可能会出现网络的拥塞。慢启动算法就是在主机刚开始发送数据报的时候先探测一下网络的状况，如果网络状况良好，发送方每发送一次文段都能正确的接受确认报文段。那么就从小到大的增加拥塞窗口的大小，即增加发送窗口的大小。

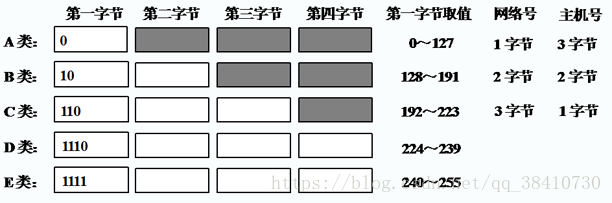
拥塞避免：为了防止拥塞窗口增加过快，设置一个慢启动开始门限，当拥塞窗口的大小超过这个门限时，拥塞窗口缓慢增加而不是成倍增加。

快重传：快重传算法要求接收方收到一个失序的报文段后就立刻发出重复确认，而不要等待自己发送数据时才进行捎带确认。

快恢复：当发送方连续接收到三个确认时，首先执行乘法减小算法，把慢启动开始门限减小一半，然后把拥塞窗口减小一半，开始执行拥塞避免控制算法。

IP地址分类

IP地址由网络号和主机号构成，网络号的长度决定了整个因特网中能容纳多少网络，主机号的长度决定每个网络能容纳多少台主机。IPv4规定，因特网地址长度为32比特（IPv6规定地址长度为128比特）。



主机号为全0时用于表示网络地址，主机号为全1时用于表示广播地址。

A: 0.0.0.0-127.255.255.255

第一个字节的最高位固定为0，另外7比特可变的网络号可以标识128个网络（0～127），0一般不用，127用作环回地址。所以共有126个可用的A类网络。三字节的主机号可以标识2^24 =1677216台主机。

其中，10.0.0.1 --- 10.255.255.254为A类私有地址。

B: 128.0.0.0-191.255.255.255

第一个字节的最高2比特固定为10，另外14比特可变的网络号可以标识2^14=16384个网络。两字节的主机号可以标识2^16 =65536台主机。

其中，172.16.0.1 --- 172.31.255.254为B类私有地址。

C: 192.0.0.0-223.255.255.255

第一个字节的最高3比特固定为110，另外21比特可变的网络号可以标识2^21=2097152个网络。8比特主机号可以标识2^8=256台主机。

其中，192.168.0.1 --- 192.168.255.254为C类私有地址。

D: 224.0.0.0-239.255.255.255

D类地址用于组播，又被称为组播地址。每个地址对应一个组，发往某一组播地址的数据将被该组中的所有成员接收。D类地址不能分配给主机。

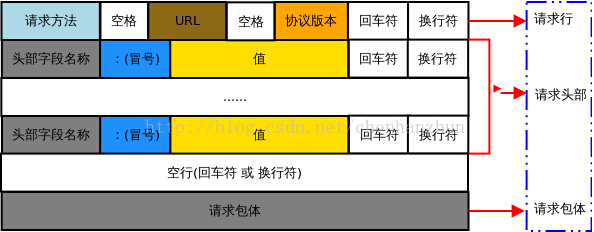
E: 240.0.0.0-255.255.255.255

E类地址为保留地址，可以用于实验目的。

从URL在浏览器被输入到页面展现的过程中发生了什么？

HTTP的报文结构

HTTP请求报文由四部分构成：



请求首行：请求首行由请求方法、URL字段和HTTP协议版本字段3个部分组成，他们之间使用空格隔开。请求方法有get、post、delete、put、head、options、trace和connect

请求头部：请求头部由一系列键值对组成，每行存储一对，保存请求时的系统和浏览器的信息。

请求空行：

请求体：get请求中没有请求空行和请求体，post请求中有请求体，请求体中用来携带请求参数，get请求参数在url地址后面已经拼接了。

post和get的区别

* get请求没有空行和请求体，因为提交的参数已经在url中拼接了；post请求有空行和请求体，post请求参数在请求体中携带
* get请求相对不安全，post相对安全
* get请求参数在url地址栏后面携带，url地址栏数据大小限制在255个字符，所以get请求有限制
* post参数没有大小限制，（只要是上传文件都采用post请求）
* post请求头中多了和参数相关的内容

Http的状态码含义

状态代码有三位数字组成，第一个数字定义了响应的类别，共分五种类别:

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

常见状态码

200 OK //客户端请求成功

400 Bad Request //客户端请求有语法错误，不能被服务器所理解

401 Unauthorized //请求未经授权，此状态代码必须和WWW-Authenticate报头域一起使用

403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

Cookie与Session的作用原理

HTTP是无状态协议，这意味着每次客户端检索网页时，都要单独打开一个服务器连接，因此服务器不会记录下先前客户端请求的任何信息。为了跟踪用户的轨迹以及保存用户的状态信息，诞生了Cookie和Session这样的会话跟踪技术。

Cookie：Cookie是客户端技术，服务器把每个用户的数据以Cookie的形式写给用户各自的浏览器。当用户使用浏览器再去访问服务器中的web资源时，就会带着各自的数据去。这样，web资源处理的就是用户各自的数据了。Cookie实际上是一小段的文本信息，客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

Session：Session是服务器端技术，在服务器端，服务器在运行时可以为每一个用户的浏览器创建一个其独享的session对象，然后把session的id号以cookie的形式写回给浏览器。由于session为用户浏览器独享，所以用户在访问服务器的web资源时，可以把各自的数据放在各自的session中，当用户再去访问服务器时，都会带着session的id号，服务器就会根据这个id号找到对应的session对象，并且从里面提取出之前的数据位用户服务。

重定向和转发的区别