# Java基础

## 1.String类为什么是final的。

答：final 修饰的类是不被能继承的，所以 final 修饰的类是不能被篡改的。

1、从设计安全)上讲，

1)、确保它们不会在子类中改变语义。String类是final类，这意味着不允许任何人定义String的子类。

换言之，

如果有一个String的引用，它引用的一定是一个String对象，而不可能是其他类的对象。

2)、String 一旦被创建是不能被修改的，

3)、不可变对象不能被写，所以线程安全

4)、不可变性支持字符串常量池，这样在大量使用字符串的情况下，可以节省内存空间，提高效率。

## 2．HashMap的源码，实现原理，底层结构。

答：HashMap的链表元素对应的是一个静态内部类Entry，Entry主要包含key，value，next三个元素

主要有put和get方法，put的原理是，通过hash%Entry.length计算index，此时记作Entry[index]=该元素。如果index相同，就是新入的元素放置到Entry[index]，原先的元素记作Entry[index].next，get就比较简单了，先遍历数组，再遍历链表元素。

null key总是放在Entry数组的第一个元素

解决hash冲突的方法：链地址法

再散列rehash的过程：确定容量超过目前哈希表的容量，重新调整table 的容量大小，当超过容量的最大值时，取Integer.Maxvalue

## 3．说说你知道的几个Java集合类：list、set、queue、map

答：

1.List 用于描述一个有序集合，关心的是索引，与其他集合相比，List特有的就是和索引相关的一些方法：get(int index) 、 add(int index,Object o) 、 indexOf(Object o) 。它有两种访问元素的协议：一种是用迭代器，另一种是用get和set方法随机地访问每个元素。后者不适用于链表，但对数组却很有用。

**ArrayList**封装了一个动态再分配的Object[]数组，它提供快速迭代和快速随机访问的能力。

**LinkedList** 中的元素之间是双链接的，当需要快速插入和删除时LinkedList成为List中的不二选择。使用链表的唯一理由就是尽可能地减少在列表中间插入或删除元素所付出的代价。如果列表只有少数几个元素，完全可以使用ArrayList。避免使用以整数索引表示链表中位置的所有方法。如果需要对集合进行随机访问，就使用数组或ArrayList，而不要使用链表。它具有方法addFirst()、addLast()、getFirst()、getLast()、removeFirst()、removeLast()，这些方法(没有在任何接口或基类中定义过)使得LinkedList可以当作堆栈、队列和双向队列使用。

**Vector** 是ArrayList的线程安全版本，性能比ArrayList要低，现在已经很少使用

2、Set接口

Set关心唯一性，它不允许重复。set接口没有引入新方法，所以Set就是一个Collection，只不过其行为不同。

**HashSet** 当不希望集合中有重复值，并且不关心元素之间的顺序时可以使用此类。

**LinkedHashset**当不希望集合中有重复值，并且希望按照元素的插入顺序进行迭代遍历时可采用此类。扩展HashSet，它提供了一个可以快速访问各个元素的有序集合。同时，它也增加了实现的代价，因为 哈希表元中的各个元素是通过双重链接式列表链接在一起的。

**TreeSet** 当不希望集合中有重复值，并且希望按照元素的自然顺序进行排序时可以采用此类。将一个元素添加到TreeSet要比添加到HashSet中慢。TreeSet是SortedSet接口的唯一实现类，TreeSet可以确保集合元素处于排序状态。TreeSet支持两种排序方式，自然排序 和定制排序，其中自然排序为默认的排序方式。向TreeSet中加入的应该是同一个类的对象。 TreeSet判断两个对象不相等的方式是两个对象通过equals方法返回false，或者通过CompareTo方法比较没有返回0 。

3、Map接口

Map关心的是唯一的标识符。他将唯一的键映射到某个元素。当然键和值都是对象。

**HashMap**当需要键值对表示，又不关心顺序时可采用HashMap。为了优化HashMap空间的使用，您可以调优初始容量和负载因子。

**Hashtable**注意Hashtable中的t是小写的，它是HashMap的线程安全版本，现在已经很少使用。

**LinkedHashMap** 当需要键值对，并且关心插入顺序时可采用它。扩展HashMap，以插入顺序将关键字/值对添加进链接哈希映像中。象LinkedHashSet一样，LinkedHashMap内部也采用双重链接式列表。

**TreeMap** 当需要键值对，并关心元素的自然排序时可采用它。TreeMap没有调优选项，因为该树总处于平衡状态。

**WeakHashMap** 是Map的一个特殊实现，它使用WeakReference(弱引用)来存放哈希表关键字。使用这种方式时，当映射的键在 WeakHashMap 的外部不再被引用时，垃圾收集器会将它回收，但它将把到达该对象的弱引用纳入一个队列。WeakHashMap的运行将定期检查该队列，以便找出新到达的 弱应用。当一个弱引用到达该队列时，就表示关键字不再被任何人使用，并且它已经被收集起来。然后WeakHashMap便删除相关的映射。

**IdentityHashMap** 也是Map的一个特殊实现。在这个类中，关键字的哈希码不应该由hashCode()方法来计算，而应该由 System.identityHashCode方法进行计算(即使已经重新定义了hashCode方法)。这是Object.hashCode根据对象 的内存地址来计算哈希码时使用的方法。另外，为了对各个对象进行比较，IdentityHashMap将使用==，而不使用equals方法。

4、Queue接口

**Queue**用于保存将要执行的任务列表。

**LinkedList**同样实现了Queue接口，可以实现先进先出的队列。

**PriorityQueue**用来创建自然排序的优先级队列。

## 4. 描述一下ArrayList和LinkedList各自实现和区别

答：ArrayList和LinkedList的大致区别如下:

1.ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。

2.对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。(aryylist尾部插入直接给数组末尾赋值，其他remove和insert都是复制数组

system.arrycopy方法)

1. 对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据。

## 5. Java中的队列都有哪些，有什么区别

答：。Java提供的线程安全的Queue可以分为阻塞队列和非阻塞队列，其中阻塞队列的典型例子是BlockingQueue，非阻塞队列的典型例子是ConcurrentLinkedQueue，在实际应用中要根据实际需要选用阻塞队列或者非阻塞队列。

**LinkedBlockingQueue**

由于LinkedBlockingQueue实现是线程安全的，实现了先进先出等特性，是作为生产者消费者的首选，LinkedBlockingQueue 可以指定容量，也可以不指定，不指定的话，默认最大是Integer.MAX\_VALUE，其中主要用到put和take方法，put方法在队列满的时候会阻塞直到有队列成员被消费，take方法在队列空的时候会阻塞，直到有队列成员被放进来。

**ConcurrentLinkedQueue**

ConcurrentLinkedQueue是Queue的一个安全实现．Queue中元素按FIFO原则进行排序．采用CAS操作，来保证元素的一致性。

LinkedBlockingQueue是一个线程安全的阻塞队列，它实现了BlockingQueue接口，BlockingQueue接口继承自java.util.Queue接口，并在这个接口的基础上增加了take和put方法，这两个方法正是队列操作的阻塞版本

## 6. 反射中，Class.forName和classloader的区别

java中class.forName()和classLoader都可用来对类进行加载。

class.forName()前者除了将类的.class文件加载到jvm中之外，还会对类进行解释，执行类中的static块。

而classLoader只干一件事情，就是将.class文件加载到jvm中，不会执行static中的内容,只有在newInstance才会去执行static块。

Class.forName(name, initialize, loader)带参函数也可控制是否加载static块。并且只有调用了newInstance()方法采用调用构造函数，创建类的对象

## 7.Java内存泄露的问题调查定位：jmap，jstack的使用等等

1.jps主要用来输出JVM中运行的进程状态信息。语法格式如下：

jps [options] [hostid]：

-q 不输出类名、Jar名和传入main方法的参数

-m 输出传入main方法的参数

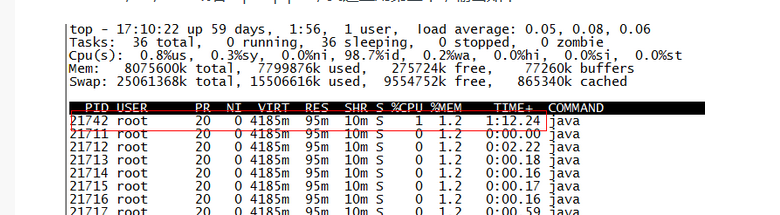
-l 输出main类或Jar的全限名

-v 输出传入JVM的参数

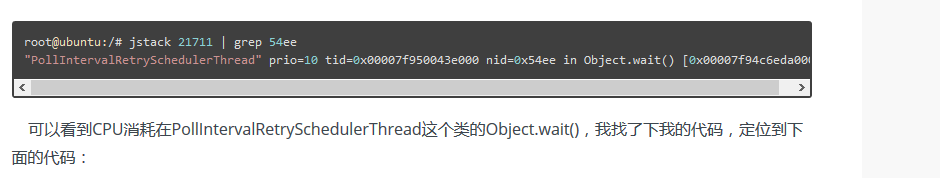
2. jstack可以定位到线程堆栈，根据堆栈信息我们可以定位到具体代码，所以它在JVM性能调优中使用得非常多。下面我们来一个实例找出某个Java进程中最耗费CPU的Java线程并定位堆栈信息，用到的命令有ps、top、printf、jstack、grep。

(1) 第一步先找出Java进程ID：ps -ef | grep + 应用名

(2) 找出该进程内最耗费CPU的线程：top -Hp pid ，TIME+最大的就是最耗cpu的



(3) 第二步看到id为21742的进程最耗时间，用printf "%x\n" 21742命令，得到21742的十六进制值为54ee，它用来输出进程21711的堆栈信息，然后根据线程ID的十六进制值，执行如下命令: jstack 21711 | grep 54ee



1. jmap用来查看堆内存使用状况，一般结合jhat使用。

## 8. string、stringbuilder、stringbuffer区别

答：

String 字符串常量

StringBuffer 字符串变量（线程安全）

StringBuilder 字符串变量（非线程安全）

简要的说， String 类型和 StringBuffer 类型的主要性能区别其实在于 String 是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，所以经常改变内容的字符串最好不要用 String ，因为每次生成对象都会对系统性能产生影响，特别当内存中无引用对象多了以后， JVM 的 GC 就会开始工作，那速度是一定会相当慢的。

而如果是使用 StringBuffer 类则结果就不一样了，每次结果都会对 StringBuffer 对象本身进行操作，而不是生成新的对象，再改变对象引用。所以在一般情况下我们推荐使用 StringBuffer ，特别是字符串对象经常改变的情况下。

java.lang.StringBuilder一个可变的字符序列是5.0新增的。此类提供一个与 StringBuffer 兼容的 API，但不保证同步。该类被设计用作 StringBuffer 的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）。如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。两者的方法基本相同。

## 9 .hashtable和hashmap的区别

答： HashMap和Hashtable都实现了Map接口，但决定用哪一个之前先要弄清楚它们之间的分别。主要的区别有：线程安全性，同步(synchronization)，以及速度。

1. HashMap几乎可以等价于Hashtable，除了HashMap是非synchronized的，并可以接受null(HashMap可以接受为null的键值(key)和值(value)，而Hashtable则不行)。

2. HashMap是非synchronized，而Hashtable是synchronized，这意味着Hashtable是线程安全的，多个线程可以共享一个Hashtable；而如果没有正确的同步的话，多个线程是不能共享HashMap的。Java 5提供了ConcurrentHashMap，它是HashTable的替代，比HashTable的扩展性更好。

3. 另一个区别是HashMap的迭代器(Iterator)是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。所以当有其它线程改变了HashMap的结构（增加或者移除元素），将会抛出ConcurrentModificationException，但迭代器本身的remove()方法移除元素则不会抛出ConcurrentModificationException异常。但这并不是一个一定发生的行为，要看JVM。这条同样也是Enumeration和Iterator的区别。

4. 由于Hashtable是线程安全的也是synchronized，所以在单线程环境下它比HashMap要慢。如果你不需要同步，只需要单一线程，那么使用HashMap性能要好过Hashtable。

HashMap不能保证随着时间的推移Map中的元素次序是不变的。

## 10. 异常的结构，运行时异常和非运行时异常，各举个例子

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。

ArithmeticException - 算术运算异常

ArrayStoreException - 向数组中存放与声明类型不兼容对象异常

IndexOutOfBoundsException - 下标越界异常

NegativeArraySizeException - 创建一个大小为负数的数组错误异常

NumberFormatException - 数字格式异常

SecurityException - 安全异常

UnsupportedOperationException - 不支持的操作异常

1、Thorwable类

Thorwable类所有异常和错误的超类，有两个子类Error和Exception，分别表示错误和异常。其中异常类Exception又分为运行时异常(RuntimeException)和非运行时异常，这两种异常有很大的区别，也称之为不检查异常（Unchecked Exception）和检查异常（Checked Exception）。

2.Error（错误）：

一般是指java虚拟机相关的问题，如系统崩溃、虚拟机出错误、动态链接失败等，这种错误无法恢复或不可能捕获，将导致应用程序中断，通常应用程序无法处理这些错误，因此应用程序不应该捕获Error对象，也无须在其throws子句中声明该方法抛出任何Error或其子类。

3.Exception：

Exception类及其子类是Throwable的一种形式，它指出了合理的应用程序想要捕获的条件

（1）SQLException：该异常提供关于数据库访问错误或其他错误的信息。

（2）RuntimeException 是那些可能在 Java 虚拟机正常运行期间抛出的异常的超类

（3）IOException：此类为异常的通用类，它是由失败的或中断的 I/O 操作生成的。

4、运行时异常和非运行时异常

（1）运行时异常都是RuntimeException类及其子类异常，如NullPointerException、IndexOutOfBoundsException等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引起的，程序应该从逻辑角度尽可能避免这类异常的发生。

（2）非运行时异常是RuntimeException以外的异常，类型上都属于Exception类及其子类。从程序语法角度讲是必须进行处理的异常，如果不处理，程序就不能编译通过。如IOException、SQLException等以及用户自定义的Exception异常，一般情况下不自定义检查异常。

5、异常的捕获和处理

Java异常的捕获和处理是一个不容易把握的事情，如果处理不当，不但会让程序代码的可读性大大降低，而且导致系统性能低下，甚至引发一些难以发现的错。

Java异常处理涉及到五个关键字，分别是：try、catch、finally、throw、throws。下面将骤一介绍，通过认识这五个关键字，掌握基本异常处理知识。

（1）、try：它里面放置可能引发异常的代码

（2）、catch：后面对应异常类型和一个代码块，用于表明该catch块用于处理这种类型的代码块，可以有多个catch块。

（3）、finally：主要用于回收在try块里打开的物力资源（如数据库连接、网络连接和磁盘文件），异常机制总是保证finally块总是被执行。只有finally块，执行完成之后，才会回来执行try或者catch块中的return或者throw语句，如果finally中使用了return或者throw等终止方法的语句，则就不会跳回执行，直接停止。

(对于这个说的顺序，大师兄我做了个测试，可以看下，链接：亲自测试下try catch finally 的执行顺序)

（4）、throw：用于抛出一个实际的异常，可以单独作为语句使用，抛出一个具体的异常对象。

（5）、throws：用在方法签名中，用于声明该方法可能抛出的异常。

6、throw与throws关键字的区别

（1）throw关键字是用于方法体内部，用来抛出一个Throwable类型的异常。如果抛出了检查异常，则还应该在方法头部声明方法可能抛出的异常类型。该 方法的调用者也必须检查处理抛出的异常。如果所有方法都层层上抛获取的异常，最终JVM会进行处理，处理也很简单，就是打印异常消息和堆栈信息。如果抛出 的是Error或RuntimeException，则该方法的调用者可选择处理该异常。

（2）throws关键字用于方法体外部的方法声明部分，用来声明方法可能会抛出某些异常。仅当抛出了检查异常，该方法的调用者才必须处理或者重新抛出该异常。当方法的调用者无力处理该异常的时候，应该继续抛出，而不是囫囵吞枣一般在catch块中打印一下堆栈信息做个勉强处理。

## 11.String a= “abc” String b = “abc” String c = new String(“abc”) String d = “ab” + “c” .他们之间用 == 比较的结果

答：public static void main(String[] args) {  
 String a ="abc";  
 String b="abc";  
 String c= new String( "abc");  
 String d = "ab"+"c";  
 System.*out*.println(a==b);  
 System.*out*.println(a==c);  
 System.*out*.println(a==d);  
 System.*out*.println(b==c);  
 System.*out*.println(b==d);  
 System.*out*.println(c==d);  
}

结果：

true

false

true

false

true

false

## 12．String 类的常用方法

构造方法：

1、public String()

无参构造方法，用来创建空字符串的String对象。

1 String str1 = new String();

2、public String(String value)

用已知的字符串value创建一个String对象。

1 String str2 = new String("asdf"); 2 String str3 = new String(str2);

3、public String(char[] value)

用字符数组value创建一个String对象。

1 char[] value = {"a","b","c","d"};

2 String str4 = new String(value);//相当于String str4 = new String("abcd");

4、public String(char chars[], int startIndex, int numChars)

用字符数组chars的startIndex开始的numChars个字符创建一个String对象。

1 char[] value = {"a","b","c","d"};

2 String str5 = new String(value, 1, 2);//相当于String str5 = new String("bc");

5、public String(byte[] values)

用比特数组values创建一个String对象。

1 byte[] strb = new byte[]{65,66};

2 String str6 = new String(strb);//相当于String str6 = new String("AB");

常用方法:

1. public int length()//返回该字符串的长度

2. public char charAt(int index)//返回字符串中指定位置的字符；注意字符串中第一个字符索引是0，最后一个是length()-1。

3、提取子串

用String类的substring方法可以提取字符串中的子串，该方法有两种常用参数:

1)public String substring(int beginIndex)//该方法从beginIndex位置起，从当前字符串中取出剩余的字符作为一个新的字符串返回。

2)public String substring(int beginIndex, int endIndex)//该方法从beginIndex位置起，从当前字符串中取出到endIndex-1位置的字符作为一个新的字符串返回

4、字符串比较

1)public int compareTo(String anotherString)//该方法是对字符串内容按字典顺序进行大小比较，通过返回的整数值指明当前字符串与参数字符串的大小关系。若当前对象比参数大则返回正整数，反之返回负整数，相等返回0。

2)public int compareToIgnore(String anotherString)//与compareTo方法相似，但忽略大小写。

3)public boolean equals(Object anotherObject)//比较当前字符串和参数字符串，在两个字符串相等的时候返回true，否则返回false。

4)public boolean equalsIgnoreCase(String anotherString)//与equals方法相似，但忽略大小写

5、字符串连接：public String concat(String str)//将参数中的字符串str连接到当前字符串的后面，效果等价于"+"。

6. 字符串中单个字符查找

1)public int indexOf(int ch/String str)//用于查找当前字符串中字符或子串，返回字符或子串在当前字符串中从左边起首次出现的位置，若没有出现则返回-1。

7.字符串中字符的大小写转换

1)public String toLowerCase()//返回将当前字符串中所有字符转换成小写后的新串

2)public String toUpperCase()//返回将当前字符串中所有字符转换成大写后的新串

8. 字符串中字符的替换

1)public String replace(char oldChar, char newChar)//用字符newChar替换当前字符串中所有的oldChar字符，并返回一个新的字符串。

2)public String replaceFirst(String regex, String replacement)//该方法用字符replacement的内容替换当前字符串中遇到的第一个和字符串regex相匹配的子串，应将新的字符串返回。

3)public String replaceAll(String regex, String replacement)//该方法用字符replacement的内容替换当前字符串中遇到的所有和字符串regex相匹配的子串，应将新的字符串返回。

其他类方法

1. String trim()//截去字符串两端的空格，但对于中间的空格不处理。
2. split(String str))//将str作为分隔符进行字符串分解，分解后的字字符串在字符串数组中返回
3. contains(String str)//判断参数s是否被包含在字符串中，并返回一个布尔类型的值。
4. statWith(String prefix)或boolean endWith(String suffix)//用来比较当前字符串的起始字符或子字符串prefix和终止字符或子字符串suffix是否和当前字符串相同，重载方法中同时还可以指定比较的开始位置offset。

## 13. Java 的引用类型有哪几种

答：有这样一类对象：当内存空间还足够，则可保留在内存中；如果内存空间在gc之后还是非常紧张，则可抛弃这些对象。很多系统的缓存功能适合这样的场景，所以jdk1.2以后

java将引用分为了强引用、软引用、弱引用、虚引用四种，引用强度一次减弱。

强引用:类似Object a=new Object()这类,永远不会被回收。例: Persion p = new Persion();

软引用:SoftReference,当系统快要发生内存溢出异常时，将会把这些对象列入回收范围进行二次回收，如果这次回收还是没有足够内存，则抛出内存溢出异常。

例如: Person jack = new Person("Jack");

SoftReference<Person> personSoftReference = new SoftReference<>(jack);

弱引用：比软引用更弱，活不过下一次gc。无论当前内存是否足够，下一次gc都会被回收掉。

Person jack = new Person("Jack");

WeakReference<Person> personSoftReference = new WeakReference<Person>(jack);

虚引用：又叫幻引用，最弱，一个对象时候有虚引用的存在，不会对它的生存时间构成影响，唯一目的就是能在这对象被回收以后收到一个系统通知。

## 14. 抽象类和接口的区别

答：1、抽象类和接口都不能直接实例化，如果要实例化，抽象类变量必须指向实现所有抽象方法的子类对象，接口变量必须指向实现所有接口方法的类对象。

2、抽象类要被子类继承，接口要被类实现。

3、接口只能做方法申明，抽象类中可以做方法申明，也可以做方法实现

4、接口里定义的变量只能是公共的静态的常量，抽象类中的变量是普通变量。

5、抽象类里的抽象方法必须全部被子类所实现，如果子类不能全部实现父类抽象方法，那么该子类只能是抽象类。同样，一个实现接口的时候，如不能全部实现接口方法，那么该类也只能为抽象类。

6、抽象方法只能申明，不能实现，接口是设计的结果 ，抽象类是重构的结果

7、抽象类里可以没有抽象方法

8、如果一个类里有抽象方法，那么这个类只能是抽象类

9、抽象方法要被实现，所以不能是静态的，也不能是私有的。

10、接口可继承接口，并可多继承接口，但类只能单根继承。

1.抽象类 和 接口 都是用来抽象具体对象的. 但是接口的抽象级别最高

2.抽象类可以有具体的方法 和属性, 接口只能有抽象方法和不可变常量

3.抽象类主要用来抽象类别,接口主要用来抽象功能.

4、抽象类中，且不包含任何实现，派生类必须覆盖它们。接口中所有方法都必须是未实现的。

## 15. java的基础类型和字节大小。

Int: 4 字节

Short: 2字节

Long: 8字节

Byte: 1字节

Character: 2字节

Float: 4字节

Double: 8字节

Boolean: 没有size

## 16.refresh

在创建hashMAP的时候可以设置来个参数，一般默认

初始化容量：创建hash表时桶的数量

负载因子：负载因子=map的size/初始化容量

当hash表中的负载因子达到负载极限的时候，hash表会自动成倍的增加容量（桶的数量），并将原有的对象

重新的分配并加入新的桶内，这称为rehash。这个过程是十分耗性能的，一般不要

一般建议设置比较大的初始化容量，防止rehash，但是也不能设置过大，初始化容量过大 浪费空间

## 17. hashCode() 与 equals() 生成算法、方法怎么重写

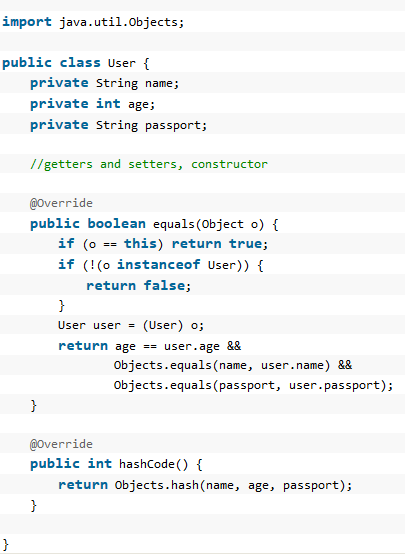
1.经典方式

这种17和31散列码



2.JDK 7

对于JDK7及更新版本，你可以是使用java.util.Objects 来重写 equals 和 hashCode 方法，代码如下



3.Apache Commons Lang

或者,您可以使用Apache Commons LangEqualsBuilder 和HashCodeBuilder 方法

