

# 1. Java += 操作符实质

---

## Java += 操作符实质

### 问题

我之前以为：  $i += j$  等同于  $i = i + j$ ；但假设有：

```
int i = 5;  
long j = 8;
```

这时  $i = i + j$  不能编译，但  $i += j$  却可以编译。这说明两者还是有差别的 这

是否意味着， $i += j$ ，实际是等同于  $i = (\text{type of } i) (i + j)$ 呢？

### 回答

这个问题，其实官方文档中已经解答了。 请看这里 [§15.26.2 Compound](#)

[Assignment Operators](#)

再照搬下官方文档的说明

对复合赋值表达式来说， $E1 \text{ op} = E2$  (诸如  $i += j$ ;  $i -= j$  等等)，其实是等同于

$E1 = (T)((E1) \text{ op } (E2))$ ，其中，T 是 E1 这个元素的类型。

举例来说，如下的代码

```
short x = 3;  
x += 4.6;
```

等同于

```
short x = 3;  
x = (short)(x + 4.6);
```

stackoverflow 链接 <http://stackoverflow.com/questions/8710619/java->

[operator](#)

## 2. 将 `InputStream` 转换为 `String`

---

### 将 `InputStream` 转换为 `String`

#### 使用 **Apache** 库

不重复造轮子。最靠谱的方法，还是用 Apache commons IOUtils 这样简单几行代码就搞定了

```
StringWriter writer = new StringWriter();
IOUtils.copy(inputStream, writer, encoding);
String theString = writer.toString();
```

或者 `String theString = IOUtils.toString(inputStream, encoding)`//这个方法其实封装了上面的方法，减少了一个参数

#### 使用原生库

如果不想引入 Apache 库，也可以这样做

```
static String convertStreamToString(java.io.InputStream is) {
    java.util.Scanner s = new java.util.Scanner(is).useDelimiter("\\A");
    return s.hasNext() ? s.next() : "";
}
```

stackoverflow 讨论地址 <http://stackoverflow.com/questions/309424/read-convert-an-inputstream-to-a-string>

## 3. 将数组转换为 `List`

---

### 将数组转换为 `List`

#### 问题

假设有数组

```
Element[] array = {new Element(1),new Element(2),new Element(3)};
```

如何将其转换为 `ArrayList<Element> arraylist = ???`

## 回答 1

```
`new ArrayList<Element>(Arrays.asList(array))`
```

## 回答 2

`Arrays.asList(array)` 或者 `Arrays.asList(new Element(1),new Element(2),new Element(3))`

不过，这样做有些坑要注意：

1. 这样做生成的 list，是定长的。也就是说，如果你对它做 add 或者 remove，都会抛 `UnsupportedOperationException`。
2. 如果修改数组的值，list 中的对应值也会改变！

**`Arrays.asList()`** 返回的是 **`Arrays`** 内部静态类，而不是 **`Java.util.ArrayList`** 的类。这个 **`java.util.Arrays.ArrayList`** 有 **`set()`**,**`get()`**,**`contains()`** 方法，但是没有任何 **`add()`** 方法，所以它是固定大小的

如果希望避免这两个坑，请改用这个方式

```
Collections.addAll(arraylist, array);
```

stackoverflow 原址：<http://stackoverflow.com/questions/157944/how-to-create-arraylist-arraylistt-from-array-t>

## 4. 如何遍历 map 对象

---

## HashMap 遍历

在 Java 中有多种遍历 HashMap 的方法。让我们回顾一下最常见的方法和它们各自的优缺点。由于所有的 Map 都实现了 Map 接口，所以接下来方法适用于所有 Map（如：HashMap，TreeMap,LinkedMap,HashTable,etc）

### 方法#1 使用 For-Each 迭代 entries

这是最常见的方法，并在大多数情况下更可取的。当你在循环中需要使用 Map 的键和值时，就可以使用这个方法

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
for(Map.Entry<Integer, Integer> entry : map.entrySet()){
    System.out.println("key = " + entry.getKey() + ", value = " +
entry.getValue())
}
```

注意：For-Each 循环是 Java5 新引入的，所以只能在 Java5 以上的版本中使用。如果你遍历的 map 是 null 的话，For-Each 循环会抛出 NullPointerException 异常，所以在遍历之前你应该判断是否为空引用。

### 方法#2 使用 For-Each 迭代 keys 和 values

如果你只需要用到 map 的 keys 或 values 时，你可以遍历 KeySet 或者 values 代替 entrySet

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();

//iterating over keys only
for (Integer key : map.keySet()) {
    System.out.println("Key = " + key);
}

//iterating over values only
```

```
for (Integer value : map.values()) {
    System.out.println("Value = " + value);
}
```

这个方法比 `entrySet` 迭代具有轻微的性能优势(大约快 10%)并且代码更简洁

## 方法#3 使用 **Iterator** 迭代

使用泛型

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
Iterator<Map.Entry<Integer, Integer>> entries = map.entrySet().iterator();
while (entries.hasNext()) {
    Map.Entry<Integer, Integer> entry = entries.next();
    System.out.println("Key = " + entry.getKey() + ", Value = " +
entry.getValue());
}
```

不使用泛型

```
Map map = new HashMap();
Iterator entries = map.entrySet().iterator();
while (entries.hasNext()) {
    Map.Entry entry = (Map.Entry) entries.next();
    Integer key = (Integer)entry.getKey();
    Integer value = (Integer)entry.getValue();
    System.out.println("Key = " + key + ", Value = " + value);
}
```

你可以使用同样的技术迭代 `keyset` 或者 `values`

这个似乎有点多余但它具有自己的优势。首先，它是遍历老 java 版本 `map` 的唯一方法。另外一个重要的特性是可以让你在迭代的时候从 `map` 中删除 `entries` 的(通过调用 `iterator.remove()`)唯一方法.如果你试图在 For-Each 迭代的时候删除 `entries`，你将会得到 `unpredictable results` 异常。

从性能方法看，这个方法等价于使用 For-Each 迭代

## 方法#4 迭代 **keys** 并搜索 **values**（低效的）

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
for (Integer key : map.keySet()) {
    Integer value = map.get(key);
    System.out.println("Key = " + key + ", Value = " + value);
}
```

这个方法看上去比方法#1 更简洁，但是实际上它更慢更低效，通过 key 得到 value 值更耗时（这个方法在所有实现 map 接口的 map 中比方法#1 慢 20%-200%）。如果你安装了 FindBugs，它将检测并警告你这是一个低效的迭代。这个方法应该避免

## 总结

如果你只需要使用 key 或者 value 使用方法#2，如果你坚持使用 java 的老版本（java 5 以前的版本）或者打算在迭代的时候移除 entries，使用方法#3。其他情况请使用#1 方法。避免使用#4 方法。

stackoverflow 链接: <http://stackoverflow.com/questions/1066589/iterate-through-a-hashmap>

## 5. public, protected, private, 不加修饰符。有什么区别呢？

**Java 修饰符: public, protected, private, 不加修饰符。有什么区别呢？**

如下表所示,Y 表示能访问(可见性)，N 表示不能访问，例如第一行的第 3 个 Y，表示类的变量/方法如果是用 public 修饰，它的子类能访问这个变量/方法

修饰符	类内部	同个包（ <b>package</b> ）	子类
public	Y	Y	Y
protected	Y	Y	Y
无修饰符	Y	Y	N or Y(见说明)
private	Y	N	N

说明： 需要特别说明“无修饰符”这个情况，子类能否访问父类中无修饰符的变量/方法，取决于子类的位置。如果子类和父类在同一个包中，那么子类可以访问父类中的无修饰符的变量/方法，否则不行。

译注：本来觉得很简单一个问题，没想记录的，但看到答案，才发现自己以前错了。我以前一直以为无修饰符和 `private` 是一样的，如果没给变量加修饰符，java 就默认为 `private`。

stackoverflow 链接： <http://stackoverflow.com/questions/215497/in-java-whats-the-difference-between-public-default-protected-and-private>

## 6. 如何测试一个数组是否包含指定的值？

如何测试一个数组是否包含指定的值

指定数组，如：

```
public static final String[] VALUES = new String[] {"AB","BC","CD","AE"};
```

现在制定一个值 `s`，有哪些比较好的方式，判断这个数组 `VALUES` 是否包含值 `s`？

## 简单且优雅的方法:

1. `Arrays.asList(...).contains(...)`
2. 使用 Apache Commons Lang 包中的 `ArrayUtils.contains`

```
String[] fieldsToInclude = { "id", "name", "location" };

if ( ArrayUtils.contains( fieldsToInclude, "id" ) ) {
    // Do some stuff.
}
```

## 自己写逻辑

问题的本质，其实是一个查找的问题，即查找一个数组是否包含某个值。对于原始类型，若是无序的数组，可以直接写一个 `for` 循环:

```
public static boolean useLoop(String[] arr, String targetValue) {
    for(String s: arr){
        if(s.equals(targetValue))
            return true;
    }
    return false;
}
```

若是有序的数组，可以考虑二分查找或者其他查找算法:

```
public static boolean useArraysBinarySearch(String[] arr, String
targetValue) {
    int a = Arrays.binarySearch(arr, targetValue);
    if(a >= 0)
        return true;
    else
        return false;
}
```



若数组里包含的是一个对象，实际上比较就是引用是否相等(String 类型是判断值是否相等)，本质就是比较 hashCode 和 equals 方法，可以考虑使用 List 或者 Set，如下

```
public static boolean useList(String[] arr, String targetValue) {
    return Arrays.asList(arr).contains(targetValue);
}
public static boolean useLoop(String[] arr, String targetValue) {
    for(String s: arr){
        if(s.equals(targetValue))
            return true;
    }
    return false;
}
```

stackoverflow 原址:<http://stackoverflow.com/questions/1128723/how-can-i-test-if-an-array-contains-a-certain-value>

## 7. 重写（Override）equals 和 hashCode 方法时应考虑的问题

### 重写（Override）equals 和 hashCode 方法时应考虑的问题

#### 理论上讲（编程语言、数学层面）

equals() 定义了对象的相等关系（自反性、对称性、传递性）（有点抽象，更详细说明，请参考 [javadoc](#)）。另外，它还具有一致性（也就是说，如果一个对象没有修改，那么对象的 equals 方法，应总是返回相同的值），此外，o.equals(null)应当总是返回 false。hashCode() ([javadoc](#))也必须具备一致性的（也就是说，如果 equals 的结果没有变，那么 hashCode()也应总是返回相同的值）

总的来说，这两个方法的关系：

假如 **a.equals(b)**，那么 **a.hashCode()** 应等于 **b.hashCode()**

## 实践上讲

如果你重写了其中一个方法，那么务必重写另外一个方法

equals()和 hashCode()所计算的属性集（set of fields）应当是一样的 如何更快地重写这两个方法呢？

1. 使用 [Apache Commons Lang library](#) 中的 [EqualsBuilder](#)、[HashCodeBuilder](#)

```
public class Person {
    private String name;
    private int age;
    // ...

    public int hashCode() {
        return new HashCodeBuilder(17, 31). // two randomly chosen prime
numbers
        // if deriving: appendSuper(super.hashCode()).
        append(name).
        append(age).
        toHashCode();
    }

    public boolean equals(Object obj) {
        if (!(obj instanceof Person))
            return false;
        if (obj == this)
            return true;

        Person rhs = (Person) obj;
        return new EqualsBuilder().
        // if deriving: appendSuper(super.equals(obj)).
        append(name, rhs.name).
        append(age, rhs.age).
        isEqual();
    }
}
```

```
}  
}
```

2. 如果你是用 eclipse, 可以在代码编辑区右键, 然后选择 Source >

Generate hashCode() and equals()

另外请记得

当你使用一些基于 Hash 的 Collection 、 Map, 例如 HashSet, HashSet, HashMap, Hashtable, 、 WeakHashMap 等。在键值对被放到集合中之后, 请确保其 key 值所对应的 hashCode()是保持不变的。比较可靠的一个办法, 是保持这些 key 是不可变的, 这也能带来不少好处

stackoverflow 链接: <http://stackoverflow.com/questions/27581/what-issues-should-be-considered-when-overriding-equals-and-hashcode-in-java>

## 8. 从一个多层嵌套循环中直接跳出

### 从一个多层嵌套循环中直接跳出

#### 问题

Java 中如何从一个多层嵌套循环中退出, 例如下面, 有两个循环, break 只能退出一个 for 循环, 不能直接跳过第二个 for 循环

```
for (Type type : types) {  
    for (Type t : types2) {  
        if (some condition) {  
            // Do something and break...  
            break; // 这样只退出了最里的 for 循环  
        }  
    }  
}
```

## 回答

可以用 `break+label` 的语法，例子如下

```
public class Test {
    public static void main(String[] args) {
        outerloop:
        for (int i=0; i < 5; i++) {
            for (int j=0; j < 5; j++) {
                if (i * j > 6) {
                    System.out.println("Breaking");
                    break outerloop;
                }
                System.out.println(i + " " + j);
            }
        }
        System.out.println("Done");
    }
}
```

首先在 `for` 循环前加标签，如例子中的 `outerloop`，然后在 `for` 循环内 `break` `label`(如本例的 `outerloop`),就会跳出该 `label` 指定的 `for` 循环。

stackoverflow 链接: <http://stackoverflow.com/questions/886955/breaking-out-of-nested-loops-in-java>

## 9. 如何将 String 转换为 Int

### 如何将 String 转换为 Int

有两种方式

```
Integer x = Integer.valueOf(str);
// or
int y = Integer.parseInt(str);
```

这两种方式有一点点不同:

- `valueOf` 返回的是 `java.lang.Integer` 的实例

- `parseInt` 返回的是基本数据类型 `int`

`Short.valueOf/parseShort`, `Long.valueOf/parseLong` 等也是有类似差别。

另外还需注意的是，在做 `int` 类型转换时，可能会抛出

`NumberFormatException`，因此要做好异常捕获

```
int foo;
String StringThatCouldBeANumberOrNot = "26263Hello"; //will throw exception
String StringThatCouldBeANumberOrNot2 = "26263"; //will not throw exception
try {
    foo = Integer.parseInt(StringThatCouldBeANumberOrNot);
} catch (NumberFormatException e) {
    //Will Throw exception!
    //do something! anything to handle the exception.
}

try {
    foo = Integer.parseInt(StringThatCouldBeANumberOrNot2);
} catch (NumberFormatException e) {
    //No problem this time but still it is good practice to care about
    exceptions.
    //Never trust user input :)
    //do something! anything to handle the exception.
}
```

stackoverflow 链接: <http://stackoverflow.com/questions/5585779/converting-string-to-int-in-java>

## 10. 如何分割（split）string 字符串

### 如何分割（split）string 字符串

使用 `String#split()` 方法  
如下所示：

```
String string = "004-034556";
String[] parts = string.split("-");
String part1 = parts[0]; // 004
```

```
String part2 = parts[1]; // 034556
```

需要注意的是，该方法的参数是个[正则表达式](#)，要注意对某些字符做转码。例如，`.`在正则表达式中表示任意字符，因此，如果你要通过`.`号做分割，需要这样写，`split("\\.")`或者 `split(Pattern.quote("."))`  
如果只是为了验证字符串中是否包含某个字符，使用 `String#contains` 方法就行。注意该方法的参数，不是正则表达式

stackoverflow 链接: <http://stackoverflow.com/questions/3481828/how-to-split-a-string-in-java>

## 11. 在 java 中如何对比 (compare) string

### 在 java 中如何对比 (compare) string

- `==`对应的是指针相等，也就是他们是否为同一个对象
- `.equals()`对应的是值相等，也就是逻辑相等

因此，如果你想检查两个字符串是否为相同值，那么应该用`.equals()`方法  
//值是相等的

```
new String("test").equals("test") // --> true
```

// ... 值相等，但不是同个对象(指向不同的地址空间)

```
new String("test") == "test" // --> false
```

// ... 同上

```
new String("test") == new String("test") // --> false
```

// 这个返回 true，是因为这种写法属于字符串字面量，编译器会维护一个常量池，相同的字面量，都会指向相同的一个对象

```
"test" == "test" // --> true
```

因此，值的对比，一般都是用 `equals` 方法。字符串字面量之间的对比，也可以用`==`（大家知其所以然即可，但没必要用`==`）

下面多举个字符串字面量的例子,下面代码中，前四个对比，返回 `true`，最后一个返回 `false`。

```

public static final String test1 = "test";
public static final String test2 = "test";

@Test
public void test() {

    String test3 = "test";
    String test = "test";

    System.out.println(test3.equals(test));
    System.out.println(test3 == test);
    System.out.println(test1.equals(test2));
    System.out.println(test1 == test2);
    System.out.println(test1 == new String("test"));
}

```

## 其他

- 如果你重写了 `equal` 方法，记得相对应地修改 `hashCode` 方法，否则将会违反这两个方法的对等关系，如果两个对象是相等（`equal`）的，那么两个对象调用 `hashCode` 必须产生相同的整数结果，即：`equal` 为 `true`，`hashCode` 必须为 `true`，`equal` 为 `false`，`hashCode` 也必须为 `false`
- 如果要忽略大小写进行对比，可以用 `equalsIgnoreCase()` 方法

## 12. Map<Key, Value>基于 Value 值排序

### Map<Key, Value>基于 Value 值排序

#### 方法 1:

使用 `TreeMap`，可以参考下面的代码

```

public class Testing {

    public static void main(String[] args) {

```

```

    HashMap<String,Double> map = new HashMap<String,Double>();
    ValueComparator bvc = new ValueComparator(map);
    TreeMap<String,Double> sorted_map = new TreeMap<String,Double>(bvc);

    map.put("A",99.5);
    map.put("B",67.4);
    map.put("C",67.4);
    map.put("D",67.3);

    System.out.println("unsorted map: "+map);

    sorted_map.putAll(map);

    System.out.println("results: "+sorted_map);
}
}

class ValueComparator implements Comparator<String> {

    Map<String, Double> base;
    public ValueComparator(Map<String, Double> base) {
        this.base = base;
    }

    // Note: this comparator imposes orderings that are inconsistent with
    equals.
    public int compare(String a, String b) {
        if (base.get(a) >= base.get(b)) {
            return -1;
        } else {
            return 1;
        } // returning 0 would merge keys
    }
}
}

```

译注：如果不自己写 Comparator,treemap 默认是用 key 来排序

## 方法 2:

先通过 linkedlist 排好序，再放到 LinkedHashMap 中

```
public class MapUtil
```



```

{
    public static <K, V extends Comparable<? super V>> Map<K, V>
        sortByValue( Map<K, V> map )
    {
        List<Map.Entry<K, V>> list =
            new LinkedList<Map.Entry<K, V>>( map.entrySet() );
        Collections.sort( list, new Comparator<Map.Entry<K, V>>()
        {
            public int compare( Map.Entry<K, V> o1, Map.Entry<K, V> o2 )
            {
                return (o1.getValue()).compareTo( o2.getValue() );
            }
        } );

        Map<K, V> result = new LinkedHashMap<K, V>();
        for (Map.Entry<K, V> entry : list)
        {
            result.put( entry.getKey(), entry.getValue() );
        }
        return result;
    }
}

```

译注：这两种方法，我简单测试了下，如果 map 的 size 在十万级别以上，两者的耗时都是几百毫秒，第二个方法会快一些。否则，第一个方法快一些。因此，如果你处理的 map，都是几十万级别以下的大小，两种方式随意使用，看个人喜欢了。

stackoverflow 链接：<http://stackoverflow.com/questions/109383/how-to-sort-a-mapkey-value-on-the-values-in-java>

## 13. `HashMap 和 Hashtable 的区别

### HashMap 和 Hashtable 的区别

#### 问题

在 Java 中 HashMap 和 Hashtable 的区别？ 哪一个对于多线程应用程序更好？

## 回答

1. Hashtable 是同步的，加了 synchronized 锁，而 HashMap 不是。没有加 synchronized 锁的对象，性能通常比加了 synchronized 锁的对象要更好一些，因此，如果是非多线程程序，不需要考虑锁、同步等问题，那么使用 HashMap 更好。
2. Hashtable 不允许有空的键或值。HashMap 允许空键和空值。
3. HashMap 有一个子类 [LinkedHashMap](#)对这个类对象进行迭代时，它的顺序是有序的（按插入顺序排序）。如有需要，你也能轻易的从 LinkedHashMap 转化成 HashMap。Hashtable 就没那么简单了，

总之，如果你无需关心同步（synchronized）问题，我会建议用 HashMap。反之，你可以考虑使用 [ConcurrentHashMap](#)

## stackoverflow 链接:

<http://stackoverflow.com/questions/40471/differences-between-hashmap-and-hashtable>

## 相关推荐:

import new: [HashMap 和 Hashtable 的区别](#)

## [14. 如何便捷地将两个数组合到一起](#)

---

如何便捷地将两个数组合到一起

## 一行代码搞定

Apache Commons Lang library `ArrayUtils.addAll(T[], T...)`就是专门干这事的  
代码：

```
String[] both = ArrayUtils.addAll(first, second);
```

## 不借助依赖包

### 非泛型

把下面的 `Foo` 替换成你自己的类名

```
public Foo[] concat(Foo[] a, Foo[] b) {  
    int aLen = a.length;  
    int bLen = b.length;  
    Foo[] c = new Foo[aLen+bLen];  
    System.arraycopy(a, 0, c, 0, aLen);  
    System.arraycopy(b, 0, c, aLen, bLen);  
    return c;  
}
```

### 泛型

```
public <T> T[] concatenate (T[] a, T[] b) {  
    int aLen = a.length;  
    int bLen = b.length;  
  
    @SuppressWarnings("unchecked")  
    T[] c = (T[]) Array.newInstance(a.getClass().getComponentType(),  
aLen+bLen);  
    System.arraycopy(a, 0, c, 0, aLen);  
    System.arraycopy(b, 0, c, aLen, bLen);  
  
    return c;  
}
```

注意，泛型的方案不适用于基本数据类型（int，boolean.....）

## 15. Java 是否支持默认的参数值

---

## Java 是否支持默认的参数值？

在 c++ 中，常见到如下的方法定义(param3 默认为 false)：

```
void MyParameterizedFunction(String param1, int param2, bool param3=false);
```

那在 java 中，是否也支持这样的定义方式？

答案是否定的，不过我们可以通过多种方式处理这种参数默认值的情况。

## 创建者模式

使用创建者模式，你可以设定部分参数是有默认值，部分参数是可选的。如：

```
Student s1 = new StudentBuilder().name("Eli").buildStudent();
Student s2 = new StudentBuilder()
    .name("Spicoli")
    .age(16)
    .motto("Aloha, Mr Hand")
    .buildStudent();
```

## 方法（构造函数）重载

如：

```
void foo(String a, Integer b) {
    //...
}

void foo(String a) {
    foo(a, 0); // here, 0 is a default value for b
}

foo("a", 2);
foo("a");
```

构造函数重载，对于参数比较少的情況下，比较适合；当参数相对多的时候，可以考虑使用静态工厂方法，或添加一个参数辅助对象。

如果是常规方法重载，可以考虑使用 参数辅助对象，或者重命名多种情况（比如说，有多个开银行卡的重载方法，可以根据需要重命名为 开交行卡，开招行卡 等多种方法）。

## null 的传递

当有多个默认参数时，可以考虑传递 null，当参数为 null 时，将参数设为默认值。如：

```
void foo(String a, Integer b, Integer c) {  
    b = b != null ? b : 0;  
    c = c != null ? c : 0;  
    //...  
}  
  
foo("a", null, 2);
```

## 多参数方式

当有多个参数，且某些参数可以忽略不设置的情况下，可以考虑使用多参数方式。

- 可选的参数类型的一致

```
void foo(String a, Integer... b) {  
    Integer b1 = b.length > 0 ? b[0] : 0;  
    Integer b2 = b.length > 1 ? b[1] : 0;  
    //...  
}  
  
foo("a");  
foo("a", 1, 2);
```

- 可选参数类型不一致

```
void foo(String a, Object... b) {  
    Integer b1 = 0;  
    String b2 = "";  
    if (b.length > 0) {  
        if (!(b[0] instanceof Integer)) {  
            throw new IllegalArgumentException("...");  
        }  
        b1 = (Integer)b[0];  
    }  
    if (b.length > 1) {  
        if (!(b[1] instanceof String)) {  
            throw new IllegalArgumentException("...");  
        }  
        b2 = (String)b[1];  
    }  
}
```

```

        throw new IllegalArgumentException("...");
    }
    b2 = (String)b[1];
    //...
}
//...
}

foo("a");
foo("a", 1);
foo("a", 1, "b2");

```

## 使用 **Map** 作为方法中的参数

当参数很多，且大部分参数都会使用默认值的情况，可以使用 **Map** 作为方法中的参数。

```

void foo(Map<String, Object> parameters) {
    String a = "";
    Integer b = 0;
    if (parameters.containsKey("a")) {
        if (!(parameters.get("a") instanceof Integer)) {
            throw new IllegalArgumentException("...");
        }
        a = (String)parameters.get("a");
    }
    if (parameters.containsKey("b")) {
        //...
    }
    //...
}

foo(ImmutableMap.<String, Object>of(
    "a", "a",
    "b", 2,
    "d", "value"));

```

stackoverflow 原址: <https://stackoverflow.com/questions/997482/does-java-support-default-parameter-values>

## 16. Java 产生指定范围的随机数

### java 产生指定范围的随机数

问题，如何使用 java 产生 0 到 10 之间的随机数？

#### Math.random()

Math.random() 可以产生一个 大于等于 0 且 小于 1 的双精度伪随机数，假设需要产生 “0 ≤ 随机数 ≤ 10” 的随机数，可以这样做：

```
int num =(int)(Math.random() * 11);
```

那如何产生 “5 ≤ 随机数 ≤ 10” 的随机数呢？

```
int num = 5 + (int)(Math.random() * 6);
```

生成 “min ≤ 随机数 ≤ max” 的随机数

```
int num = min + (int)(Math.random() * (max-min+1));
```

#### java.util.Random

Random 是 java 提供的一个伪随机数生成器。

生成 “min ≤ 随机数 ≤ max” 的随机数：

```
import java.util.Random;

/**
 * Returns a pseudo-random number between min and max, inclusive.
 * The difference between min and max can be at most
 * <code>Integer.MAX_VALUE - 1</code>.
```

```

*
* @param min Minimum value
* @param max Maximum value. Must be greater than min.
* @return Integer between min and max, inclusive.
* @see java.util.Random#nextInt(int)
*/
public static int randInt(int min, int max) {

    // NOTE: Usually this should be a field rather than a method
    // variable so that it is not re-seeded every call.
    Random rand = new Random();

    // nextInt is normally exclusive of the top value,
    // so add 1 to make it inclusive
    int randomNum = rand.nextInt((max - min) + 1) + min;

    return randomNum;
}

```

## 标准库

在实际使用中，没有必要重新写一次这些随机数的生成规则，可以借助一些标准库完成。如 [commons-lang](#).

`org.apache.commons.lang3.RandomUtils` 提供了如下产生指定范围的随机数方法:

```

// 产生 start <= 随机数 < end 的随机整数
public static int nextInt(final int startInclusive, final int
endExclusive);
// 产生 start <= 随机数 < end 的随机长整数
public static long nextLong(final long startInclusive, final long
endExclusive);
// 产生 start <= 随机数 < end 的随机双精度数
public static double nextDouble(final double startInclusive, final double
endInclusive);
// 产生 start <= 随机数 < end 的随机浮点数
public static float nextFloat(final float startInclusive, final float
endInclusive);

```



org.apache.commons.lang3.RandomStringUtils 提供了生成随机字符串的方法，简单介绍一下：

```
// 生成指定个数的随机数字串
public static String randomNumeric(final int count);
// 生成指定个数的随机字母串
public static String randomAlphabetic(final int count);
// 生成指定个数的随机字母数字串
public static String randomAlphanumeric(final int count);
```

stackoverflow 原址：<http://stackoverflow.com/questions/363681/generating-random-integers-in-a-range-with-java> 文章若有写得不正确或不通顺的地方，恳请你指出，谢谢。

## 17. JavaBean 到底是什么

---

### JavaBean 到底是什么？

#### 问题

按照我的理解：“Bean” 是一个带有属性和 getters/setter 方法的 Java 类。它

是不是和 C 的结构体是相似的呢，对吗？ 一个“Bean”类与普通的类相比是不是语法的不同呢？还是有特殊的定义和接口？ 为什么会出现这个术语呢，这让我很困惑？ 如果你很好心告诉我一些关于 Serializable 接口的信息，对于你的答案那到底是什么意思，我会非常感谢你的。

#### 回答

JavaBean 只是一个[标准](#)

1. 所有的属性是私有的（通过 [getters/setters](#) 处理属性）

2. 一个公有的无参数的构造器

3. 实现了[序列化 \(Serializable\)](#)

就这些，它只是一个规范。但是很多的类库都是依赖于这些预定。

对于 `Serializable`, 看一下 [API 文档的解释](#)

实现 `java.io.Serializable` 接口的类能串行化。

不实现此接口的类不会有任何状态的序列化和反序列化。

可序列化类的所有子类型本身都是可序列化。

序列化接口没有方法或字段，仅用于标识的可序列化的语义。

换句话说，序列化的对象可以被写入流，文件，对象数据库等。

另外，一个 `JavaBean` 类和一个普通的类没有语法区别，如果遵循上面的标准的话，一个类可以认为成 `JavaBean` 类。

之所以需要 `JavaBean`，是因为这样预定义了一种类的格式，一些库能依据这个约定的格式，来做一些自动化处理。举个例子，如果一个类库需要通过流来处理你传递的任何对象，它知道它可以正常处理，因为这个对象是可序列化的。

（假设这个类库要求你的对象是 `JavaBeans`）

**stackoverflow 链接:** <http://stackoverflow.com/questions/3295496/what-is-a-javabean-exactly>

## 关于序列化相关博客

1. [我对 Java Serializable（序列化）的理解和总结](#)
2. [理解 Java 对象序列化](#)

## 18. wait()和 sleep()的区别

---

### wait()和 sleep()的区别

## 问题：

在线程里 `wait()` 和 `sleep()` 的区别？

我的理解是执行 `wait()` 语句后，该线程仍是运行态，并且会占用 CPU，但是执行 `sleep()` 后，该线程则不会占用 CPU，对吗？

为什么需要 `sleep()` 和 `wait()` 两条语句：他们底层是如何实现的？

## 回答：

线程在 `wait` 后，可以被另一个拥有相同 `synchronized` 对象的线程，通过调用 `notify` 唤醒，而 `sleep` 不行。`wait` 和 `notify` 能正常执行的条件是（否则会抛异常）：多个线程的代码，都包在 `synchronized` 块中，并且 `synchronized` 锁的对象需要是同一个。如下所示：

```
Object mon = ...;
synchronized (mon) {
    mon.wait();
}
```

上面这个线程调用了 `wait` 后，会进入等待状态。这时另外一个线程可以这样做：

```
synchronized (mon) { mon.notify(); }
```

可以看到，`synchronized` 锁对象，都是 `mon`。因此，当第二个线程调用了 `notify()` 方法，第一个线程就会唤醒（假设有且仅有一个线程是被包在 `synchronized (mon)` 中且处于等待状态）。

如果有多个线程在等待（且 `synchronized` 锁对象是同一个，如上例中的 `mon`），则可以调用 `notifyAll` 来唤醒。但是，只有其中一个线程能抢到锁并继续执行（因为 `wait` 的线程都是在 `synchronized` 块内，需要争夺 `synchronized` 锁）。其他的线程会被锁住，直到他们依次获得锁。

再补充几点：

- `wait` 方法由 `Object` 对象调用（例如：你可以让 `synchronized` 锁对象调用 `wait` ,如上面例子的 `mon.wait()`）,而 `sleep` 则由线程调用。
- `wait` 之后，可能会伪唤醒（`spurious wakeups`）（正在 `waiting` 的线程,无故就被唤醒了，如遇到 `interrupted`, `timing out` 等情况）。因此，你需要多设置一些检查，如果不满足实际的运行条件，则继续等待，如下：

```
synchronized {
    while (!condition) { mon.wait(); }
}
```

- 当线程调用 `sleep` 时，并没有释放对象锁，而 `wait` 则释放了对象锁：

```
synchronized(LOCK) {
    Thread.sleep(1000); // LOCK is held
}
synchronized(LOCK) {
    LOCK.wait(); // LOCK is not held
}
```

最后，再小结一下：

- `sleep()`：“我已经完成了一个时间片，在 **n 微秒**前，请不要再给我一个时间片”。这时操作系统不会让这个线程做任何事情，直到 `sleep` 时间结束。
- `wait()`：“我已经完成了一个时间片，在**其他线程调用 `notify()`**前，请不要再给我一个时间片）。这时操作系统不会安排这个线程继续运行，直到有人调用了 `notify()`

## stackoverflow 链

接：<http://stackoverflow.com/questions/1036754/difference-between-wait-and-sleep>

## 相关问题及链接：

1. [Java: notify\(\) vs. notifyAll\(\) all over again](#)
2. [线程通信](#)
3. [最简实例说明 wait、notify、notifyAll 的使用方法](#)

## 19. 能否在一个构造器( constructor )中调用另一个构造器

能否在一个构造器中调用另一个构造器

### 问题

能否在一个构造器中调用另一个构造器（在同一个类中，不是子类）？如果可以，怎么做？ 调用另一个构造器的最好方法是什么（如果有几种方法可以选择的话）？

### 回答

可以这样做：

```
public class Foo
{
    private int x;

    public Foo()
    {
        this(1);
    }

    public Foo(int x)
    {
        this.x = x;
    }
}
```

如果你想调用一个特定的父类构造器，而不是本类的构造器，应该使用 `super`，而不是 `this`。请注意，在构造器中，你只能调用一次其他的构造器。并且调用其他构造器的语句，必须是这个构造器的第一个语句。

stackoverflow 原址: <http://stackoverflow.com/questions/285177/how-do-i-call-one-constructor-from-another-in-java>

## 20. finally 代码块总会被执行么

---

### 问题

有一个 `try/catch` 代码块，其中包含一个打印语句。`finally` 代码块总会被执行么？

示例：

```
try {
    something();
    return success;
}
catch (Exception e) {
    return failure;
}
finally {
    System.out.println("i don't know if this will get printed out.");
}
```

### 回答

1. `finally` 将会被调用。

只有以下情况 `finally` 不会被调用：

- 当你使用 `System.exit()` 后
- 其他线程干扰了现在运行的线程（通过 `interrupt` 方法）

- JVM 崩溃( crash )了

Answered by [Jodonnell](#), edited by [jpaugh](#).

## 2.示例代码

```
class Test
{
    public static void main(String args[])
    {
        System.out.println(Test.test());
    }

    public static int test()
    {
        try {
            return 0;
        }
        finally {
            System.out.println("finally trumps return.");
        }
    }
}
```

输出:

```
finally trumps return.
0
```

Answered by [Kevin](#)

---

原文链接: <http://stackoverflow.com/questions/65035/does-finally-always-execute-in-java?page=1&tab=votes#tab-top>

## 21. 如何将 String 转换为 enum

---

## 如何将 **String** 转换为 **enum**

### 问题

假设定义了如下的 **enum**（枚举）：

```
public enum Blah {  
    A, B, C, D  
}
```

已知枚举对应的 **String** 值，希望得到对应的枚举值。例如，已知"A"，希望得到对应的枚举——**Blah.A**，应该怎么做？

**Enum.valueOf()**是否能实现以上目的，如果是，那我如何使用？

### 答案

是的，**Blah.valueOf("A")** 将会得到 **Blah.A**

静态方法 **valueOf()** 和 **values()** 不存在于源码中，而是在编译时创建，我们也可以在 **JavaDoc** 查看到它们，比如 [Dialog.ModalityType](#) 就中出现这两个方法。

### 其他答案

我有一个挺赞的工具方法：

```
/**  
 * A common method for all enums since they can't have another base class  
 * @param <T> Enum type  
 * @param c enum type. All enums must be all caps.  
 * @param string case insensitive  
 * @return corresponding enum, or null  
 */  
public static <T extends Enum<T>> T getEnumFromString(Class<T> c, String  
string) {  
    if( c != null && string != null ) {
```



```
        try {  
            return Enum.valueOf(c, string.trim().toUpperCase());  
        } catch (IllegalArgumentException ex) {  
        }  
    }  
    return null;  
}
```

你可以这么使用：

```
public static MyEnum fromString(String name) {  
    return getEnumFromString(MyEnum.class, name);  
}
```

stackoverflow 链接: <http://stackoverflow.com/questions/604424/convert-a-string-to-an-enum-in-java>

## 22. 在 Java 中声明数组

---

### 在 java 中声明数组

#### 问题描述：

你是如何在 Java 中声明数组的。

#### 回答：

你可以直接用数组声明，或者通过数组的字面常量（array literal）声明

对于原始类型（primitive types）：

```
int[] myIntArray = new int[3];  
int[] myIntArray = {1, 2, 3};  
int[] myIntArray = new int[]{1, 2, 3};
```

对于其他类，比如 String 类，也是相同的：

```
String[] myStringArray = new String[3];
```

```
String[] myStringArray = {"a", "b", "c"};  
String[] myStringArray = new String[]{"a", "b", "c"};
```

[stackoverflow 链接: Declare array in Java?](#)

## 23. 反射是什么及其用途

### 反射（**reflection**）是什么及其用途？

#### 问题描述

反射是什么，为什么它是有用的？ 我特别感兴趣的是 java，但我认为任何语言的原理都是相同的。

#### 回答

反射的概念，主要是指程序可以访问、检测和修改它本身状态或行为的一种能力。在 java 中，通过反射，能够在"运行态"动态获得任意一个类的所有属性和方法，动态地调用对象的方法。

举个例子，假设你有一个不知道具体类的对象，并且你想调用它的 "dosomething" 方法（如果存在的话）。java 的静态类型系统只能调用一个已知类对象对应的已知接口，在未指定对象类型时，无法调用它的方法。但是通过反射，你的代码能检查这个未知类对象，并试图找出这个 dosomething 方法。如果存在这个方法，你可以通过反射调用这个方法。

为了进一步说明，请看下面的例子（下面的对象 foo，就是上文提到的，我们不知道它对应的类是什么）：

```
Method method = foo.getClass().getMethod("dosomething",null);  
method.invoke(foo,null); //调用 foo 的 dosomething 方法
```

反射这个特性，经常会用于各种注解中(annotations)。举个例子，JUnit4 将使用反射来遍历你的代码，查找所有加了 @test 注解的类方法，之后运行测试单元时就调用这些方法。

[有很多好的反射例子，可以用来入门](#)

最后，其概念在其他支持反射的静态类型语言中也是非常相似的。在动态语言中，无需用到上面说的第一种用法场景——调用未知类的方法（因为动态语言编允许任意对象调用任意方法，如果不存在对应方法，在运行时就会失败），但是第二种情况，查找做了指定标记的方法，这种场景还是很常见的

[stackoverflow 链接: What is reflection, and why is it useful?](#)

## 24. 为什么不能用 `string` 类型进行 `switch` 判断

---

### 为什么不能用 `string` 类型进行 `switch` 判断

#### 问题描述

为什么不能用 `string` 类型进行 `switch` 判断？在 java 的后续版本中，是否会增加这个新特性？有人能给我一篇文章，解释一下为什么不能这样做，或者进一步说明 java 中 `switch` 语句的运行方式？

#### 回答

在 `switch` 语句中用 `string` 作为 `case`，这个特性已经在 java SE7 中被实现了，距离 [这个'bug'](#) 被提出至少也有 16 年了。为何迟迟不提供这个特性，原因不明。但可以推测，可能跟性能有关。

#### Implementtation in JDK 7

在 JDK7 中，这个特性已经实现了。在编译阶段，以 `string` 作为 `case` 值的代码，会按照特定的模式，被转换为更加复杂的代码。最终的执行代码将是一些使用了 JVM 指令的代码。

究竟是如何转换的呢？我们直接看看源码及编译后的代码。源代码：

```
public class StringInSwitchCase {
    public static void main(String[] args) {
        String mode = args[0];
        switch (mode) {
            case "ACTIVE":
                System.out.println("Application is running on Active
mode");
                break;
            case "PASSIVE":
                System.out.println("Application is running on Passive
mode");
                break;
            case "SAFE":
                System.out.println("Application is running on Safe
mode");
        }
    }
}
```

编译后再反编译的代码：

```
import java.io.PrintStream;

public class StringInSwitchCase{
    public StringInSwitchCase() { }

    public static void main(string args[]) {
        String mode = args[0];
        String s; switch ((s = mode).hashCode()) {
            default: break;
            case -74056953:
                if (s.equals("PASSIVE")) {
                    System.out.println("Application is running
on Passive mode");
                }
                break;
            case 2537357:
                if (s.equals("SAFE")) {
```

```

        System.out.println("Application is running on
Safe mode");
    }
    break;
case 1925346054:
    if (s.equals("ACTIVE")) {
        System.out.println("Application is running on
Active mode");
    }
    break;
}
}
}

```

包含 case string 的 switch 语句，在编译时会转为为嵌套代码（switch+if）。

第一个 switch 将 case 中的 string 转为唯一的 integer 值。这个 integer 值就是原先 string 的 hashCode 值。在 case 的逻辑中，会加入 if 语句，这个 if 语句用于进一步检查 string 值是否跟原先的 case string 匹配。这样可以防止 hash 碰撞，确保代码的健壮。这本质上是一种语法糖，既支持了 string 作为 case 值这一特性，又能确保逻辑正确性。

## Switches in the JVM

switch 的更多深层技术实现，可以参考 JVM 规范，[compilation of switch statements](#)。简单概括说，根据使用的常量的多寡，switch 会对应到两种不同的 JVM 指令。JVM 指令有所不同，归根结底都是为了代码的效率。

如果常量很多，会将 case 的 int 值去掉最低位后作为索引，放到一个指针表中——也就是所谓的 `tablewitch` 指令

如果常量相对较少，那么可用二分查找来找到正确的 case--也就是所谓的 `lookupswitch` 指令

这两种指令，都要求在编译时确保 case 的对应值是 integer 常量。在运行时，虽然 `tableswitchO(1)` 的性能通常要好于 `lookupswitchO(log(n))` 的性能。但是前者需要更多的空间开销，因此需要兼顾空间及时间综合考虑性价比。Bill Venners 的文章 [a great article](#) 有更多深入的分析。

## Before JDK 7

在 JDK 之前，可以用枚举来实现类似的需求。它和在 case 中使用 string 有异曲同工之妙。例如如下：

```
Pill p = Pill.valueOf(str);
switch(p) {
    case RED: pop(); break;
    case BLUE: push(); break;
}
```

[stackoverflow 原链接: Why can't I switch on a String](#)

[可参考中文文章《Java 中字符串 switch 的实现细节》](#)

## 25. 比较 java 枚举成员使用 equal 还是 ==

### 比较 java 枚举成员使用 equal 还是 ==

#### 问题

我知道 Java 枚举会被编译成一个包含私有构造参数和一堆静态方法的类，当去比较两个枚举的时候，总是使用 `equals()` 方法，例如：

```
public useEnums(SomeEnum a)
{
    if(a.equals(SomeEnum.SOME_ENUM_VALUE))
    {
        ...
    }
}
```

```
}  
...  
}
```

除此之外，我也可以使用 `==` 替代 `equals()` 方法

```
public useEnums2(SomeEnum a)  
{  
    if(a == SomeEnum.SOME_ENUM_VALUE)  
    {  
        ...  
    }  
    ...  
}
```

我有 5 年以上的 java 编程经验，并且我想我也懂得 `==` 和 `equals()` 之间的区别，但是我仍然觉得很困惑，哪一个操作符才是我该使用的。

## 答案

二者皆对，如果你看过枚举的源码，你会发现在源码中，`equals` 也仅仅非常简单的 `==`。我使用 `==`，因为无论如何，这个左值是可以为 `null` 的

译者补充 `java.lang.Enum` 中 `Equals` 代码：

```
public final boolean equals(Object other) {  
    return this==other;  
}
```

## 额外答案

能在枚举中使用 `==` 进行判断？

答案是肯定的，因为枚举有着严格的实例化控制，所以你可以用 `==` 去做比较符，这个用法，在官方文档中也有明确的说明。

JLS 8.9 Enums 一个枚举类型除了定义的那些枚举常量外没有其他实例了。 试

图明确地说明一种枚举类型是会导致编译期异常。在枚举中 `final clone` 方法确保枚举常量从不会被克隆，而且序列化机制会确保从不会因为反序列化而创建复制的实例。枚举类型的反射实例化也是被禁止的。总之，以上内容确保了除了定义的枚举常量之外，没有枚举类型实例。

因为每个枚举常量只有一个实例，所以如果在比较两个参考值，至少有一个涉及到枚举常量时，允许使用“==”代替 `equals()`。（`equals()`方法在枚举类中是一个 `final` 方法，在参数和返回结果时，很少调用父类的 `equals()`方法，因此是一种恒等的比较。）

什么时候 `==` 和 `equals` 不一样？

As a reminder, it needs to be said that generally, `==` is NOT a viable alternative to `equals`. When it is, however (such as with enum), there are two important differences to consider: 通常来说 `==` 不是一个 `equals` 的一个备选方案，无论如何有 2 个重要的不同处需要考虑：

`==` 不会抛出 `NullPointerException`

```
enum Color { BLACK, WHITE };  
  
Color nothing = null;  
if (nothing == Color.BLACK); // runs fine  
if (nothing.equals(Color.BLACK)); // throws NullPointerException
```

`==` 在编译期检测类型兼容性

```
enum Color { BLACK, WHITE };  
enum Chiral { LEFT, RIGHT };  
  
if (Color.BLACK.equals(Chiral.LEFT)); // compiles fine  
if (Color.BLACK == Chiral.LEFT); // DOESN'T COMPILE!!! Incompatible  
types!
```

什么时候使用 `==` ？



Bloch specifically mentions that immutable classes that have proper control over their instances can guarantee to their clients that `==` is usable. `enum` is specifically mentioned to exemplify. 具体来说，那些提供恰当实例控制的不可变类能够保证 `==` 是可用的，枚举刚好符合这个条件。

考虑静态工厂方法代替构造器 它使得不可变的类可以确保不会存在两个相等的实例，即当且仅当 `a==b` 的时候才有 `a.equals(b)` 为 `true`。如果类保证了这一点，它的客户端可以使用 `==` 操作符来代替 `equals (Object)` 方法，这样可以提升性能。枚举类型保证了这一点

总而言之，在枚举比较上使用 `==` ， 因为：

1. 能正常工作
2. 更快
3. 运行时是安全的
4. 编译期也是安全的

stackoverflow 链接：

<http://stackoverflow.com/questions/1750435/comparing-java-enum-members-or-equals>

## 26. 用 java 怎么创建一个文件并向该文件写文本内容

---

用 **java** 怎么创建一个文件并向该文件写文本内容

问：在 **java** 里最简单的创建文件写文件的方法是什么

## 最佳答案:

创建一个文本文件（注意：如果该文件存在，则会覆盖该文件）

```
PrintWriter writer = new PrintWriter("the-file-name.txt", "UTF-8");
writer.println("The first line");
writer.println("The second line");
writer.close();
```

创建一个二进制文件（同样会覆盖这文件）

```
byte data[] = ...
FileOutputStream out = new FileOutputStream("the-file-name");
out.write(data);
out.close();
```

Java 7+ 用户可以用 **File** 类来写文件 创建一个文本文件

```
List<String> lines = Arrays.asList("The first line", "The second line");
Path file = Paths.get("the-file-name.txt");
Files.write(file, lines, Charset.forName("UTF-8"));
//Files.write(file, lines, Charset.forName("UTF-8"),
StandardOpenOption.APPEND);
```

创建一个二进制文件

```
byte data[] = ...
Path file = Paths.get("the-file-name");
Files.write(file, data);
//Files.write(file, data, StandardOpenOption.APPEND);
```

## 其他的答案（1）：

在 Java 7+ 中

```
try (Writer writer = new BufferedWriter(new OutputStreamWriter(
    new FileOutputStream("filename.txt"), "utf-8"))) {
    writer.write("something");
}
```

还有一些实用的方法如下：

- **FileUtils.writeStringToFile(..)** 来自于 commons-io 包

- `Files.write(..)` 来自于 guava

Note also that you can use a `FileWriter`, but it uses the default encoding, which is often a bad idea - it's best to specify the encoding explicitly. 还要注意可以使用 `FileWriter`, 但是它使用的是默认编码, 这不是很好的方法, 最好是明确指定编码

下面是来自于 prior-to-java-7 的原始方法

```
Writer writer = null;

try {
    writer = new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream("filename.txt"), "utf-8"));
    writer.write("Something");
} catch (IOException ex) {
    // report
} finally {
    try {writer.close();} catch (Exception ex) {/*ignore*/}
}
```

可以看 [Reading, Writing, and Creating Files](#)(包含 NIO2)

## 其他答案（2）：

```
public class Program {
    public static void main(String[] args) {
        String text = "Hello world";
        BufferedWriter output = null;
        try {
            File file = new File("example.txt");
            output = new BufferedWriter(new FileWriter(file));
            output.write(text);
        } catch ( IOException e ) {
            e.printStackTrace();
        } finally {
            if ( output != null ) output.close();
        }
    }
}
```

## 其他答案（3）：

如果已经有想要写到文件中的内容，`java.nio.file.Files` 作为 Java 7 附加部分的 native I/O，提供了简单高效的方法来实现你的目标。基本上创建文件，写文件只需要一行，而且是只需一个方法调用！下面的例子创建并且写了 6 个不同的文件来展示是怎么使用的

```
Charset utf8 = StandardCharsets.UTF_8;
List<String> lines = Arrays.asList("1st line", "2nd line");
byte[] data = {1, 2, 3, 4, 5};

try {
    Files.write(Paths.get("file1.bin"), data);
    Files.write(Paths.get("file2.bin"), data,
        StandardOpenOption.CREATE, StandardOpenOption.APPEND);
    Files.write(Paths.get("file3.txt"), "content".getBytes());
    Files.write(Paths.get("file4.txt"), "content".getBytes(utf8));
    Files.write(Paths.get("file5.txt"), lines, utf8);
    Files.write(Paths.get("file6.txt"), lines, utf8,
        StandardOpenOption.CREATE, StandardOpenOption.APPEND);
} catch (IOException e) {
    e.printStackTrace();
}
```

## 其他答案（4）：

下面是一个小程序来创建和写文件。该版本的代码比较长，但是可以容易理解

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;

public class writer {
    public void writing() {
        try {
            //Whatever the file path is.
```

```

        File statText = new
File("E:/Java/Reference/bin/images/statsTest.txt");
        FileOutputStream is = new FileOutputStream(statText);
        OutputStreamWriter osw = new OutputStreamWriter(is);
        Writer w = new BufferedWriter(osw);
        w.write("POTATO!!!");
        w.close();
    } catch (IOException e) {
        System.err.println("Problem writing to the file statsTest.txt");
    }
}

    public static void main(String[] args) {
        writer write = new writer();
        write.writing();
    }
}

```

stackoverflow 链接: <http://stackoverflow.com/questions/2885173/how-to-create-a-file-and-write-to-a-file-in-java>

## 27. serialVersionUID 有什么作用？该如何使用？

**serialVersionUID** 有什么作用？该如何使用？

### 问题

当一个对象实现 `Serializable` 接口时，多数 ide 会提示声明一个静态常量 `serialVersionUID`(版本标识)，那 `serialVersionUID` 到底有什么作用呢？应该如何使用 `serialVersionUID` ？

### 回答

`serialVersionUID` 是实现 `Serializable` 接口而来的，而 `Serializable` 则是应用于 Java 对象序列化/反序列化。对象的序列化主要有两种用途：

- 把对象序列化成字节码，保存到指定介质上(如磁盘等)
- 用于网络传输

现在反过来说就是，`serialVersionUID` 会影响到上述所提到的两种行为。那到底会造成什么影响呢？

[java.io.Serializable](#) doc 文档，给出了一个相对详细解释：

`serialVersionUID` 是 Java 为每个序列化类产生的版本标识，可用来保证在反序列时，发送方发送的和接受方接收的是可兼容的对象。如果接收方接收的类的 `serialVersionUID` 与发送方发送的 `serialVersionUID` 不一致，进行反序列时会抛出 `InvalidClassException`。序列化的类可显式声明 `serialVersionUID` 的值，如下：

```
...
ANY-ACCESS-MODIFIER static final long serialVersionUID = 1L;
...
```

当显式定义 `serialVersionUID` 的值时，Java 根据类的多个方面(具体可参考 Java 序列化规范)动态生成一个默认的 `serialVersionUID`。尽管这样，还是建议你在每一个序列化的类中显式指定 `serialVersionUID` 的值，因为不同的 `jdk` 编译很可能会生成不同的 `serialVersionUID` 默认值，进而导致在反序列化时抛出 `InvalidClassExceptions` 异常。所以，为了保证在不同的 `jdk` 编译实现中，其 `serialVersionUID` 的值也一致，可序列化的类必须显式指定 `serialVersionUID` 的值。另外，`serialVersionUID` 的修饰符最好是 `private`，因为 `serialVersionUID` 不能被继承，所以建议使用 `private` 修饰 `serialVersionUID`。

举例说明如下：现在尝试通过将一个类 `Person` 序列化到磁盘和反序列化来说明 `serialVersionUID` 的作用：`Person` 类如下：

```
public class Person implements Serializable {

    private static final long serialVersionUID = 1L;

    private String name;
    private Integer age;
    private String address;

    public Person() {
```

```

    }

    public Person(String name, Integer age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            ", address='" + address + '\'' +
            '}';
    }
}

```

简单的测试一下：

```

@Test
public void testversion1L() throws Exception {
    File file = new File("person.out");
    // 序列化
    ObjectOutputStream oout = new ObjectOutputStream(new
    FileOutputStream(file));
    Person person = new Person("John", 21, "广州");
    oout.writeObject(person);
    oout.close();
    // 反序列化
    ObjectInputStream oin = new ObjectInputStream(new
    FileInputStream(file));
    Object newPerson = oin.readObject();
    oin.close();
    System.out.println(newPerson);
}

```

测试发现没有什么问题。有一天，因发展需要，需要在 `Person` 中增加了一个字段 `email`，如下：

```

public class Person implements Serializable {

    private static final long serialVersionUID = 1L;

```

```

private String name;
private Integer age;
private String address;
private String email;

public Person() {
}

public Person(String name, Integer age, String address) {
    this.name = name;
    this.age = age;
    this.address = address;
}

public Person(String name, Integer age, String address,String email) {
    this.name = name;
    this.age = age;
    this.address = address;
    this.email = email;
}

@Override
public String toString() {
    return "Person{" +
        "name='" + name + '\'' +
        ", age=" + age +
        ", address='" + address + '\'' +
        ", email='" + email + '\'' +
        '}';
}
}

```

这时我们假设和之前序列化到磁盘的 `Person` 类是兼容的，便不修改版本标识 `serialVersionUID`。再次测试如下

```

@Test
public void testversion1LWithExtraEmail() throws Exception {
    File file = new File("person.out");
    ObjectInputStream oin = new ObjectInputStream(new
FileInputStream(file));
    Object newPerson = oin.readObject();
    oin.close();
    System.out.println(newPerson);
}

```



将以前序列化到磁盘的旧 `Person` 反序列化到新 `Person` 类时，没有任何问题。

可当我们增加 `email` 字段后，不作向后兼容。即放弃原来序列化到磁盘的 `Person` 类，这时我们可以将版本标识提高，如下：

```
private static final long serialVersionUID = 2L;
```

再次进行反序列化，则会报错，如下：

```
java.io.InvalidClassException:Person local class incompatible: stream  
classdesc serialVersionUID = 1, local class serialVersionUID = 2
```

谈到这里，我们大概可以清楚，`serialVersionUID` 就是控制版本是否兼容的，若我们认为修改的 `Person` 是向后兼容的，则不修改 `serialVersionUID`；反之，则提高 `serialVersionUID` 的值。再回到一开始的问题，为什么 `ide` 会提示声明 `serialVersionUID` 的值呢？

因为若不显式定义 `serialVersionUID` 的值，`Java` 会根据类细节自动生成 `serialVersionUID` 的值，如果对类的源代码作了修改，再重新编译，新生成的类文件的 `serialVersionUID` 的取值有可能也会发生变化。类的 `serialVersionUID` 的默认值完全依赖于 `Java` 编译器的实现，对于同一个类，用不同的 `Java` 编译器编译，也有可能就会导致不同的 `serialVersionUID`。所以 `ide` 才会提示声明 `serialVersionUID` 的值。

附录拓展：

- [深入理解 Java 对象序列化](#)
- [对象的序列化和反序列化](#)

stackoverflow 原址: <http://stackoverflow.com/questions/285793/what-is-a-serialversionuid-and-why-should-i-use-it>

## 28. 为什么 Java 的 `vector` 类被认为是过时的或者废弃的

### 为什么 Java 的 `vector` 类被认为是过时的或者废弃的

#### 问题

为什么 java `vector` 类被认为是一个遗留的, 过时的或废弃的类? 在并发操作时, 使用它是无效的吗?

如果我不想手动对对象实现同步, 只想用一个线程安全的集合而无需创建底层数组的全新副本(如 `CopyOnWriteArrayList` 一样)。这种情况下, 我使用

`Vector` 合理吗?

然后就是关于栈的问题, 它是 `Vector` 的一个子类, 我应该用什么代替它?

#### 回答

`Vector` 中对每一个独立操作都实现了同步, 这通常不是我们想要的做法。对单一操作实现同步通常不是线程安全的(举个例子, 比如你想遍历一个 `Vector` 实例。你仍然需要申明一个锁来防止其他线程在同一时刻修改这个 `Vector` 实例。如果不添加锁的话

通常会在遍历实例的这个线程中导致一个 `ConcurrentModificationException`) 同时这个操作也是十分慢的(在创建了一个锁就已经足够的前提下, 为什么还需要重复的创建锁)

当然, 即使你不需要同步, `Vector` 也是有锁的资源开销的。

总的来说，在大多数情况下，这种同步方法是存在很大缺陷的。正如 Mr Brain Henk 指出，你可以通过调用 `Collections.synchronizedList` 来装饰一个集合 - 事实上 `Vector` 将“可变数组”的集合实现与“同步每一个方法”结合起来的做法是另一个糟糕的设计；各个装饰方法能够更明确的指示其关注的功能实现。

对于 `Stack` 这个类-我更乐于使用 `Deque/ArrayDeque` 来实现

stackoverflow 讨论地址: <http://stackoverflow.com/questions/1386275/why-is-java-vector-class-considered-obsolete-or-deprecated>

## 29. Java 的 foreach 循环是如何工作的

### Java 的 foreach 循环是如何工作的？

#### 问题

```
List<String> someList = new ArrayList<String>();  
// add "monkey", "donkey", "skeleton key" to someList  
for (String item : someList) {  
    System.out.println(item);  
}
```

如果不用 for each 语法，等价的循环语句是什么样的？

#### 回答

```
for(Iterator<String> i = someList.iterator(); i.hasNext(); ) {  
    String item = i.next();  
    System.out.println(item);  
}
```

记住，如果需要在循环中使用 `i.remove` 或者以某种方式获取实际的 `iterator`，你不能使用 `for(;)语法`，因为实际的 `Iterator` 很难被推断出来。正如 Denis

Bueno 写的那样，这种代码对任何实现了 `Iterable` 接口的对象都奏效。此外，如果 `for(·)` 句法中右侧是一个数组而不是一个可迭代对象，那么内部代码用一个 `int` 型的计数器来防止数组越界。详见 `Java Language`

Specification: <http://docs.oracle.com/javase/specs/jls/se8/html/jls-14.html#jls-14.14.2>

stackoverflow 链接: <http://stackoverflow.com/questions/85190/how-does-the-java-for-each-loop-work>

## 30. 为什么相减这两个时间（1927 年）会得到奇怪的结果

为什么这两个时间（1927 年）相减会得到一个奇怪的结果？

### 问题描述

如果我运行如下的程序，将两个相距一秒的日期解析成字符串并比较他们。

```
public static void main(String[] args) throws ParseException {
    SimpleDateFormat sf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String str3 = "1927-12-31 23:54:07";
    String str4 = "1927-12-31 23:54:08";
    Date sDt3 = sf.parse(str3);
    Date sDt4 = sf.parse(str4);
    long ld3 = sDt3.getTime() / 1000;
    long ld4 = sDt4.getTime() / 1000;
    System.out.println(ld4 - ld3);
}
```

输出结果为:

```
353
```

为什么 `ld4 - ld3` 不是 1（正如我所期望的那样），而是 353？

如果我把时间改变为之后的一秒:

```
String str3 = "1927-12-31 23:54:08";  
String str4 = "1927-12-31 23:54:09";
```

这时, `ld4-ld3` 的结果为 1.

java 版本:

```
java version "1.6.0_22"  
Java(TM) SE Runtime Environment (build 1.6.0_22-b04)  
Dynamic Code Evolution Client VM (build 0.2-b02-internal, 19.0-b04-  
internal, mixed mode)
```

时区:

```
sun.util.calendar.ZoneInfo[id="Asia/Shanghai",  
offset=28800000,dstSavings=0,  
useDaylight=false,  
transitions=19,  
lastRule=null]
```

```
Locale(Locale.getDefault()): zh_CN
```

## 问题回答

这是因为 1927 年 11 月 31 日上海的时区改变了。 观看[此页](#)获得更多关于上海 1927 年的细节。 这个问题主要是由于在 1927 年 12 月 31 日的午夜, 时钟回调了 5 分钟零 52 秒。 所以"1927-12-31 23:54:08"这个时间实际上发生了两次, 看上去 java 将这个时间解析为之后的那个瞬间。 因此出现了这种差别。

这只是美好但奇怪的世界时区中的一个插曲。

stackoverflow 链接: [Why is subtracting these two times \(in 1927\) giving a strange result?](#)

## [31. Java 中如何将 String 转换为 enum](#)

---

## Java 中如何将 String 转换为 enum

### 问题

### 我有一个 enum 类

```
public enum Blah {  
    A, B, C, D  
}
```

我想要找到一个 String 对应的 enum 值。例如, "A" 将是 Blah.A.如何做到?

我需要使用 Enum.valueOf() 方法吗? 如果是该如何使用?

---

### A1

是的, Blah.valueOf("A") 将会给你 Blah.A.

静态方法 valueOf() 和 values() 在编译时期被插入,并不存在于源码中。但是在 Javadoc 中;例如,Dialog.ModalityType 中显示了这两个方法。

### A2

另一个解答,如果文本和 enumeration 值不一致

```
public enum Blah {  
    A("text1"),  
    B("text2"),  
    C("text3"),  
    D("text4");  
  
    private String text;  
  
    Blah(String text) {  
        this.text = text;  
    }  
}
```

```

    }

    public String getText() {
        return this.text;
    }

    public static Blah fromString(String text) {
        if (text != null) {
            for (Blah b : Blah.values()) {
                if (text.equalsIgnoreCase(b.text)) {
                    return b;
                }
            }
        }
        return null;
    }
}

```

评论区在讨论是应该返回 `null` 还是抛出异常,个人认为视具体情况,允许转换失败就返回 `null`,不允许就抛出异常,或许创建一个特殊的空对象是个好的选择 - 译者注

## A3

这是我使用的一个工具类:

```

/**
 * 一个对于所有 Enum 类通用的方法,因为他们不能有另一个基类
 * @param <T> Enum type
 * @param c enum type. All enums must be all caps.
 * @param string case insensitive
 * @return corresponding enum, or null
 */
public static <T extends Enum<T>> T getEnumFromString(Class<T> c, String string) {
    if( c != null && string != null ) {
        try {
            return Enum.valueOf(c, string.trim().toUpperCase());
        } catch (IllegalArgumentException ex) {
        }
    }
    return null;
}

```

```
}
```

之后,在我的 enum 类中通常如此使用来减少打字:

```
public static MyEnum fromString(String name) {  
    return getEnumFromString(MyEnum.class, name);  
}
```

如果的 enums 不是全部大写,只需要修改 `Enum.valueOf` 这一行。很遗憾,我不能使用 `T.class` 传给 `Enum.valueOf`,因为 `T` 会被擦出。

*评论区对于答主的异常处理一片指责 - 译者注*

## A4

如果你不想编写自己的工具类,可以使用 Google 的 guava 库:

```
Enums.getIfPresent(Blah.class, "A")
```

它让你检查是否 `Blah` 中存在 `A` 并且不抛出异常

完整方法签名 `Optional<T> getIfPresent(Class<T> enumClass, String value)`, `Optional` 对象可以优雅的解决 `null` 值问题 - 译者注

---

其他的答案都大同小异,感兴趣的可以看原帖 [stackoverflow](#) 链接

接 <http://stackoverflow.com/questions/604424/lookup-enum-by-string-value> 译者:[MagicWolf](#)

## [32. 该什么时候使用 ThreadLocal 变量，它是如何工作的](#)

---

该什么时候使用 **ThreadLocal** 变量，它是如何工作的？



## 回答 1

一种可能的（也是常见的）使用情形是你不想通过同步方式（synchronized）访问非线程安全的对象（说的就是 SimpleDateFormat）,而是想给每个线程一个对象实例的时候。 例如

```
public class Foo
{
    // SimpleDateFormat is not thread-safe, so give one to each thread
    private static final ThreadLocal<SimpleDateFormat> formatter = new
ThreadLocal<SimpleDateFormat>(){
        @Override
        protected SimpleDateFormat initialValue()
        {
            return new SimpleDateFormat("yyyyMMdd HHmm");
        }
    };

    public String formatIt(Date date)
    {
        return formatter.get().format(date);
    }
}
```

## 回答 2

因为 ThreadLocal 是一个既定线程内部的数据引用，你可能在使用线程池的应用服务器上因此引起类加载时候的内存泄漏。你需要使用 remove()方法很小心地清理 ThreadLocal 中 get()或者 set()的变量。 如果程序执行完毕没有清理的话，它持有的任何对类的引用将作为部署的 Web 应用程序的一部分仍保持在永久堆，永远无法得到回收。重新部署/取消部署也无法清理对应用程序类的引用，因为线程不是被你的应用程序所拥有的。 每次成功部署都会创建一个永远不会被垃圾回收类的实例。

最后将会遇到内存不足的异常-`java.lang.OutOfMemoryError`:

PermGen space -XX:MaxPermSize, 在 google 了很多答案之后你可能只是增加了-XX:MaxPermSize, 而不是修复这个 bug。 倘若你的确遇到这种问题, 可以通过 [Eclipse's Memory Analyzer](#) 或根据 [Frank Kieviet's guide](#) 和 [followup](#) 来判断哪些线程和类保留了那些引用。

更新: 又发现了 [Alex Vasseur's blog entry](#), 它帮助我查清楚了一些 ThreadLocal 的问题。

stackoverflow 链接: <http://stackoverflow.com/questions/817856/when-and-how-should-i-use-a-threadlocal-variable>

## 33. servlets 的运行原理

---

### **How do servlets work? Instantiation, shared variables and multithreading**

问题:

假设, 我有一个 web 服务器可以支持无数的 servlets, 对于通过这些 servlets 的信息, 我正在获取这些 servlets 的上下文环境, 并设置 session 变量。 现在, 如果有两个或者更多的 user 用户发送请求到这个服务器, session 变量会发生什么变化? session 对于所有的 user 是公共的还是不同的 user 拥有不同的 session。 如果用户彼此之间的 session 是不同的, 那么服务器怎么区分辨别不同的用户呢? 另外一些相似的问题, 如果有 N 个用户访问一个具体的

servlets, 那么这个 servlets 是只在第一个用户第一次访问的时候实例化, 还是为每一个用户各自实例化呢?

**答案:**

## **ServletContext**

当 servletcontainer (像 tomcat) 启动的时候, 它会部署和加载所有的 webapplications, 当一个 webapplication 加载完成后, servletcontainer 就会创建一个 ServletContext, 并且保存在服务器的内存中。这个 webapp 的 web.xml 会被解析, web.xml 中的每个<servlet>, <filter> and <listener>或者通过注解 @WebServlet, @WebFilter and @WebListener, 都会被创建一次并且也保存在服务器的内存中。对于所有 filter, init()方法会被直接触发, 当 servletcontainer 关闭的时候, 它会 unload 所有的 webapplications,触发所有实例化的 servlets 和 filters 的 destroy()方法,最后, servletcontext 和所有的 servlets, filter 和 listener 实例都会被销毁。

## **HttpServletRequest and HttpServletResponse**

servletcontainer 是附属与 webserver 的, 而这个 webserver 会持续监听一个目标端口的 HTTP request 请求, 这个端口在开发中经常会被设置成 8080, 而在生产环境会被设置成 80。当一个客户端 (比如用户的浏览器) 发送一个 HTTP request, servletcontainer 就会创建新的 HttpServletRequest 对象和 HttpServletResponse 对象。。。。

在有 filter 的情况下, doFilter()方法会被触发。当代码调用

chain.doFilter(request, response)时候, 请求会经过下一个过滤器 filter, 如果

没有了过滤器，会到达 servlet。在 servlets 的情况下，`service()`触发，然后根据 `request.getMethod()`确定执行 `doGet()`还是 `doPost()`，如果当前 servlet 找不到请求的方法，返回 405error。

`request` 对象提供了 HTTP 请求所有的信息，比如 request headers 和 request body，`response` 对象提供了控制和发送 HTTP 响应的的能力，并且以你想要的方式，比如设置 headers 和 body。当 HTTP 响应结束，请求和响应对象会被销毁（实际上，大多数 container 将会清洗到这些对象的状态然后回收这些事例以重新利用）

## httpSession

当客户端第一次访问 webapp 或者通过 `request.getSession()`方法第一次获取 `httpSession`，`servletcontainer` 将会创建一个新的 `HttpSession` 对象，产生一个长的唯一的 ID 标记 `session`（可以通过 `session.getId()`），并且将这个 `session` 存储在 server 内存中。`servletcontainer` 同时会在 HTTP response 的 Header 中设置 `Set-Cookie`cookie 值，其中 cookie name 为 `JSESSIONID`，cookie value 为唯一的长 ID 值。

在接下来的连续请求中，客户端浏览器都要 cookie 通过 header 带回，然后 `servletcontainer` 会根据 cookie 中的 `JSESSIONID` 值，获得 server 内存中的对应的 `httpSession`。

只要没超过`<session-timeout>`设定的值，`httpSession` 对象会一直存在，`<session-timeout>`大小可以在 `web.xml` 中设定，默认是 30 分钟。所以如果连续 30 分钟之内客户端不再访问 webapp，`servletcontainer` 就会销毁对应的

session。接下来的 request 请求即使 cookies 依旧存在，但是却不再有对应的 session 了。servletcontainer 会创建新的 session。

另外一方面，session cookie 在浏览器端有默认的生命时长，就是只要浏览器一直在运行，所以当浏览器关闭，浏览器端的 cookie 会被销毁。

## 最后

- 只要 webapp 存在，ServletContext 一定会存在。并且 ServletContext 是被所有 session 和 request 共享的。
- 只要客户端用同一个浏览器和 webapp 交互并且该 session 没有在服务端超时，HttpSession 就会一直存在。并且在同一个会话中所有请求都是共享的。
- 只有当完整的 response 响应到达，HttpServletRequest 和 HttpServletResponse 才不再存活，并且不被共享。
- 只要 webapp 存在，servlet、filter 和 listener 就会存在。他们被所有请求和会话共享。
- 只要问题中的对象存在，任何设置在 ServletContext, HttpServletRequest 和 HttpSession 中的属性就会存在。

## 线程安全

就是说，你主要关注的是线程安全性。你应该了解到，servlets 和 filter 是被所有请求共享的。这正是 Java 的美妙之处，它的多线程和不同的线程可以利用同样的实例 instance，否则对于每一个 request 请求都要重复创建和调用 init()和 destroy()开销太大。

但是你也应该注意到，你不应该把任何请求或会话作用域的数据作为一个 `servlet` 或过滤器的实例变量。这样会被其他会话的请求共享，并且那是线程不安全的！下面的例子阐明的这点：

```
public class ExampleServlet extends HttpServlet {

    private Object thisIsNOTThreadSafe;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        Object thisIsThreadSafe;

        thisIsNOTThreadSafe = request.getParameter("foo"); // BAD!! Shared
among all requests!
        thisIsThreadSafe = request.getParameter("foo"); // OK, this is
thread safe.
    }
}
```

stackoverflow 链接：

<http://stackoverflow.com/questions/3106452/how-do-servlets-work-instantiation-shared-variables-and-multithreading>

## 34. 如何计算 MD5 值

---

### 如何计算 MD5 值

#### 问题

Java 中有没有方法可以计算一个 String 的 MD5 值？

#### 回答

你可以用 `MessageDigest` 的 MD5 实例来计算 String 的 MD5 值。

使用 `MessageDigest` 和 `String` 时，一定要显式声明你的数据编码类型。如果你使用无参的 `String.getBytes()`，它会以当前平台的默认编码来转换数据。不同平台的默认编码可能是不同的，这可能会导致你的数据不一致。

```
import java.security.*;

..

byte[] bytesOfMessage = yourString.getBytes("UTF-8");
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] thedigest = md.digest(bytesOfMessage);
```

如果你的要计算的数据量很大，你可以循环使用 `.update(byte[])` 方法来加载数据。加载完毕后用 `.digest()` 方法来得到计算出的 MD5 值。

stackoverflow 链接 <http://stackoverflow.com/questions/415953/how-can-i-generate-an-md5-hash>

## 35. Java 中软引用和弱引用的区别

### Java 中软引用和弱引用的区别

#### 问题

题目就是问题

#### 解答

#### 回答 1

从 Ethan Nicholas 的《Understanding Weak References》中

弱引用： 放置一个弱引用的作用，不是强有力强制一个对象保存在内存中。弱引用允许利用垃圾收集者的能力去决定可达性，所以你不需要做，你只需要创建一个软引用：

```
WeakReference weakWidget = new WeakReference(widget);
```

然后在代码别的地方你可以使用 `weakWidget.get()` 来获取真实的 `Widget` 对象，当然弱引用足以强大能抵制垃圾收集器，所以你也也许发现（如果没有强引用指向 `widget`）`weakWidget.get()` 突然开始返回 `null`

## 软引用

软引用就像弱引用一样，除了它不会着急将引用的对象扔出去。只有弱可达性的对象（这样的对象最强的引用只能是弱引用）将在下一次垃圾收集处理中被抛弃，但是软可达性的对象通常可以坚持一会。

软引用不要求与弱引用有什么不同，但是实际中，只要内存足够，软可达的对象通常会维持下去。对于缓存来说，这是个不错的基础，就像以上图像缓存描述，虽然可以让垃圾收集者担心对象是如何可达（一个强可达性的对象从不会从缓存中移除）和她们需要消耗多少内存

而且 Peter Kessler 备注到

Sun JRE 对待软引用和弱引用是不同的。如果内存是够用的。我们应坚持用软引用引用对象。一个细节是：对于客户端和服务端，JRE 的政策是不同的：客户端，JRE 试图保持通过清除软引用而不是扩大堆内存来使改变小点，而服务端，JRE 通过扩大堆内存让性能更好。没有一种通用的方法。

## 回答 2

弱引用对象很快被收集。如果 GC 发现一个对象是弱引用（只能通过弱引用可达），它会立刻清除弱引用对象。同样的，对于在程序保持关联信息的对象保持一个引用是不错的，像关于类的缓存存储的反射信息或一个对象的包装器等。没有意义地跟随相连对象的任何事物都会被清除掉。当弱引用清除掉时，它会进入到引用队列中，同时丢弃关联的对象。你保持关于对象额外的信息，但是一旦对象引用不要了，信息也就不需要了。总之，在某些情境下，你可以创建 `WeakReference` 的子类，保持在 `WeakReference` 的子类中对象的额外信息。`WeakReference` 的其他典型应用是与 `Map` 连接，以保持规范化的例子。



在另一方面，软引用有利于外部缓存，再创造资源，因为 GC 会延迟清理他们。它能保证所有软引用会在内存溢出之前被清除，所以它们不会造成内存溢出。

典型的使用例子是保持从一个文件内容解析形式。在你载入文件，解析和与解析过代表的根对象保持一个软引用的地方扩展系统。在你下次需要文件时，你试图通过软引用恢复。如果可以恢复，你会在其他地方载入、解析你分享的文件，如果同时 GC 清理掉，你也可以重新载入。这样的话，你利用空内存可以做到性能最优化，但是不要内存溢出。光保持一个软引用不会造成溢出。如果在另一方面你误用软引用，且弱引用被使用了（也就是说，你保持与较强引用的对象相连的信息，然后当引用对象被清除，你也丢弃信息），你可能会内存溢出，因为在进入引用队列时，也许碰巧没有及时丢弃相连的对象。

所以，使用软引用还是弱引用是取决于用法的。如果你的信息构造起来较为复杂，但是尽管如此仍想从别的数据再构造信息，使用软引用。如果你对一些数据的规范化实例保持引用，或者你想对一个“不拥有的”对象保持引用（就是防止被垃圾回收），这样就使用弱引用。

原文：

<http://stackoverflow.com/questions/299659/what-is-the-difference-between-a-soft-reference-and-a-weak-reference-in-java>

## **36. JSF, Servlet 和 JSP (三种技术)有什么区别**

---

### **JSF, Servlet 和 JSP (三种技术)有什么区别？**

#### **问题**

JSP 和 Servlet 有什么关系？JSP 是某种 Servlet 吗？JSP 和 JSF 又有什么关系？JSF 是某种基于 JSP 的，预构建好的 UI 吗，像 ASP.NET-MVC 那样？

#### **回答 1**

## **JSP(Java Server Pages)**

JSP 是一种运行在服务器上的 Java 视图技术，它允许你写入模版化的文本(例如客户端代码 HTML, CSS, JavaScript 等)。JSP 支持标签库(taglibs)，标签库由 Java 代码实现，让你可以动态地控制页面输出。JSTL 便是一种比较有名的标签库。JSP 同样支持表达式语言(expression language)，表达式语言可以用来访问后台数据(页面上可用的属性，request/session 对象等等)，通常与标签库结合使用。

当一个 JSP 第一次被访问或者 webapp 启动时，servlet 容器会将 JSP 编译成一个继承了 HttpServlet 的类，然后在整个 webapp 生命周期内使用被编译后的类。可以在 servlet 容器的 work 目录下找到 JSP 对应的源代码。例如 Tomcat 的 CATALINA.BASE/work 目录。当收到一个 JSP 请求时，servlet 容器会执行编译 JSP 生成的类，并将该类的输出(通常是 HTML/CSS/JS)发送到客户端，客户端(WEB 浏览器)会展示从服务端收到的内容。

## **Servlet**

Servlet 是一种针对服务器端的 API，它用来响应客户端请求，并生成响应。比较有名的例子是 HttpServlet，它提供了响应 HTTP 请求(例如 GET POST)的方法。你可以从 web.xml 配置 HttpServlet 来监听某种 HTTP URL pattern 的请求，或者使用较新的 Java EE 6 @WebServlet 注解。

当 Servlet 第一次被请求，或者 webapp 启动时，servlet 容器会创建该 Servlet 的实例，并在整个 webapp 的生命周期维持该实例在内存中。同一个

实例会被复用，来响应匹配到 URL pattern 的请求。可以通过 `HttpServletRequest` 访问请求里的数据，通过 `HttpServletResponse` 控制响应。上边两个对象会是 `HttpServlet` 的重载方法 `doGet()`和 `doPost()` 的参数。

## JSF (JavaServer Faces)

JSF 是一个基于组件的 MVC 框架，建立在 Servlet API 基础上，JSF 通过标签库提供组件，标签库又可以用于 JSP 或者其它 Java 视图技术例如 Facelets。Facelets 更适合 JSF。即它提供了很厉害的模版功能例如组合组件，而 JSP 基本上只提供了 `<jsp:include>` 来支持模版，所以 当你想用一個组件替换一组重复出现的组件时，你不得不使用原生的 Java 代码来创建自定义组件(这在 JSF 里并不那么清晰明了，而且带来很多冗余工作)。为了推进 Facelets，自从 JSF 2.0 之后，JSP 这种视图技术已经被废弃了。作为一种 MVC(Model-View-Controller)框架，JSF 提供了唯一的 `FacesServlet` 请求/响应控制器。它负责所有的 HTTP 请求/响应工作，例如 收集/校验/转换用户输入，将输入设置到 model 对象里，调用处理逻辑并输出响应。这样你基本上 只有一个 JSP 或者 Facelets(XHTML) 页面用作视图，再加一个 `Javabeen` 类当作 模型。JSF 组件用来将模型和视图绑定起来(类似 ASP.NET web control 做的)，然后 `FacesServlet` 使用 JSF 组件树来完成整个工作。

## 其它答案选编

参考以下链接

<http://www.oracle.com/technetwork/java/faq-137059.html>

<https://jcp.org/en/introduction/faq>

JSP 是一种特殊的 Servlet。

JSF 是一个可以配合 JSP 使用的标签集。

**stackoverflow** 原文链接:

<http://stackoverflow.com/questions/2095397/what-is-the-difference-between-jsf-servlet-and-jsp>

## 37. Java 内部类和嵌套静态类

---

### Java 内部类和嵌套静态类

#### 问题

Java 当中的内部类和静态嵌套类有什么主要区别? 在这两者中有什么设计或者实现么?

#### 回答

嵌套类分为两类: 静态和非静态. 用 `static` 装饰的嵌套类叫做静态类, 非静态的嵌套类叫做内部类.

静态嵌套类使用外围类名来访问:

```
OuterClass.StaticNestedClass
```

例如, 实例化一个静态嵌套类的对象就要使用这种语法:

```
OuterClass.StaticNestedClass nestedObject = new
OuterClass.StaticNestedClass();
```

内部类对象的存在需要依靠一个外部类的对象. 看看下面的类:

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }
}
```

内部类对象只有当外部类对象存在时才有效, 并且可以直接访问他的包裹对象 (外部类对象)的方法以及成员.

因此, 要实例化一个内部类对象, 必须先实例化外部类对象. 然后用这种语法来创建内部类对象:

```
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

参考: [Java Tutorial - Nested Classes](#)

提醒一下, 还有一种不用外部类对象来创建内部类对象的方法: [inner class](#)

[without an enclosing](#)

```
class A {
    int t() { return 1; }
    static A a = new A() { int t() { return 2; } };
}
```

在这里, `new A() { ... }` 是一个定义在静态上下文的内部类对象, 并没有一个外围对象.

stackoverflow 链接: [Java inner class and static nested class](#)

## **38. @Component, @Repository, @Service 的区别**

---

## @Component, @Repository, @Service 的区别

### 问题

在 spring 集成的框架中，注解在类上的@Component, @Repository, @Service 等注解能否被互换？或者说这些注解有什么区别？

### 回答 1

引用 spring 的官方文档中的一段描述：

在 Spring2.0 之前的版本中，@Repository 注解可以标记在任何的类上，用来表明该类是用来执行与数据库相关的操作（即 dao 对象），并支持自动处理数据库操作产生的异常

在 Spring2.5 版本中，引入了更多的 Spring 类注解：

@Component, @Service, @Controller。Component 是一个通用的 Spring 容器管理的单

例 bean 组件。而@Repository, @Service, @Controller 就是针对不同的使用场景所采取的特定功能化的注解组件。

因此，当你的一类被@Component 所注解，那么就意味着同样可以用

@Repository, @Service, @Controller 来替代它，同时这些注解会具备有更多的功能，而且功能各异。

最后，如果你不知道要在项目的业务层采用@Service 还是@Component 注解。那么，@Service 是一个更好的选择。

就如上文所说的，@Repository 早已被支持了在你的持久层作为一个标记可以去自动处理数据库操作产生的异常（译者注：因为原生的 java 操作数据库所产生的异常只定义了几种，但是产生数据库异常的原因却有很多种，这样对于数据库操作的报错排查造成了一定的影响；而 Spring 拓展了原生的持久层异常，针对不同的产生原因有了更多的异常进行描述。所以，在注解了@Repository 的类上如果数据库操作中抛出了异常，就能对其进行处理，转而抛出的是翻译后的 spring 专属数据库异常，方便我们对异常进行排查处理）。

注解	含义
@Component	最普通的组件，可以被注入到 spring 容器中
@Repository	作用于持久层
@Service	作用于业务逻辑层
@Controller	作用于表现层（spring-mvc 的注解）

## 回答 2

这几个注解几乎可以说是一样的：因为被这些注解修饰的类就会被 Spring 扫描到并注入到 Spring 的 bean 容器中。

这里，有两个注解是不能被其他注解所互换的：

- @Controller 注解的 bean 会被 spring-mvc 框架所使用。
- @Repository 会被作为持久层操作（数据库）的 bean 来使用

如果想使用自定义的组件注解，那么只要在你定义的新注解中加上@Component即可：

```
@Component
@Scope("prototype")
public @interface ScheduleJob {...}
```

这样，所有被@ScheduleJob 注解的类就都可以注入到 spring 容器来进行管理。

我们所需要的，就是写一些新的代码来处理这个自定义注解（译者注：可以用反射的方法），进而执行我们想要执行的工作。

## 回答 3

@Component 就是跟<bean>一样，可以托管到 Spring 容器进行管理。

@Service, @Controller , @Repository = {@Component + 一些特定的功能}。

这个就意味着这些注解在部分功能上是一样的。

当然，下面三个注解被用于为我们的应用进行分层：

- @Controller 注解类进行前端请求的处理，转发，重定向。包括调用 Service 层的方法
- @Service 注解类处理业务逻辑
- @Repository 注解类作为 DAO 对象（数据访问对象，Data Access Objects），这些类可以直接对数据库进行操作

有这些分层操作的话，代码之间就实现了松耦合，代码之间的调用也清晰明朗，便于项目的管理；假想一下，如果只用@Controller 注解，那么所有的请求转发，业务处理，数据库操作代码都糅合在一个地方，那这样的代码该有多难拓展和维护。

## 总结

- @Component, @Service, @Controller, @Repository 是 spring 注解，注解后可以被 spring 框架所扫描并注入到 spring 容器来进行管理
- @Component 是通用注解，其他三个注解是这个注解的拓展，并且具有了特定的功能
- @Repository 注解在持久层中，具有将数据库操作抛出的原生异常翻译转化为 spring 的持久层异常的功能。
- @Controller 层是 spring-mvc 的注解，具有将请求进行转发，重定向的功能。
- @Service 层是业务逻辑层注解，这个注解只是标注该类处于业务逻辑层。
- 用这些注解对应用进行分层之后，就能将请求处理，业务逻辑处理，数据库操作处理分离出来，为代码解耦，也方便了以后项目的维护和开发。

**Stackoverflow 链接：**



<http://stackoverflow.com/questions/6827752/whats-the-difference-between-component-repository-service-annotations-in>

拓展

1. [Spring 注解@Component、@Repository、@Service、@Controller 区别](#)
2. [Spring 注解@Autowired、@Resource 区别](#)

## 39. 如何创建泛型 java 数组

---

### 如何创建泛型 java 数组

问题

数组是不能通过泛型创建的，因为我们不能创建不可具体化的类型的数组。如下面的代码：

```
public class GenSet<E> {
    private E a[];

    public GenSet() {
        a = new E[INITIAL_ARRAY_LENGTH]; //编译期就会报错：不能创建泛型数组
    }
}
```

采纳答案

- 检查：强类型。GenSet 明确知道数组中包含的类型是什么（例如通过构造器传入 Class<E>，当方法中传入类型不是 E 将抛出异常）

```
public class GenSet<E> {

    private E[] a;

    public GenSet(Class<E> c, int s) {
        // 使用原生的反射方法，在运行时知道其数组对象类型
        @SuppressWarnings("unchecked")
```

```

        final E[] a = (E[]) Array.newInstance(c, s);
        this.a = a;
    }

    E get(int i) {
        return a[i];
    }

    //...如果传入参数不为 E 类型，那么强制添加进数组将会抛出异常
    void add(E e) {...}
}

```

- 未检查：弱类型。数组内对象不会有任何类型检查，而是作为 `Object` 类型传入。

在这种情况下，你可以采取如下写法：

```

public class GenSet<E> {

    private Object[] a;

    public GenSet(int s) {
        a = new Object[s];
    }

    E get(int i) {
        @SuppressWarnings("unchecked")
        final E e = (E) a[i];
        return e;
    }
}

```

上述代码在编译期能够通过，但因为泛型擦除的缘故，在程序执行过程中，数组的类型有且仅有 `Object` 类型存在，这个时候如果我们强制转化为 `E` 类型的话，在运行时会有 `ClassCastException` 抛出。所以，要确定好泛型的上界，将上边的代码重写一下：

```

public class GenSet<E extends Foo> { // E has an upper bound of Foo

    private Foo[] a; // E 泛型在运行期会被擦除为 Foo 类型，所以这里使用 Foo[]

    public GenSet(int s) {
        a = new Foo[s];
    }

    //...
}

```

```
}
```

**StackOverflow** 地址:

<http://stackoverflow.com/questions/529085/how-to-create-a-generic-array-in-java>

## 编程技巧

### 1. 去掉烦人的“!=null”(判空语句)

---

#### 去掉烦人的“!=null”(判空语句)

##### 问题

为了避免空指针调用，我们经常会看到这样的语句

```
if (someobject != null) {  
    someobject.doCalc();  
}
```

最终，项目中会存在大量判空代码，多么丑陋繁冗！如何避免这种情况？我们是否滥用了判空呢？

##### 回答

这是初、中级程序员经常会遇到的问题。他们总喜欢在方法中返回 `null`，因此，在调用这些方法时，也不得不判空。另外，也许受此习惯影响，他们总潜意识地认为，所有的返回都是不可信任的，为了保护自己程序，就加了大量的判空。

吐槽完毕，回到这个题目本身，进行判空前，请区分以下两种情况：

1. `null` 是一个有效有意义的返回值(Where `null` is a valid response in terms of the contract; and)

## 2. null 是无效有误的(Where it isn't a valid response.)

你可能还不明白这两句话的意思，不急，继续往下看，接下来将详细讨论这两种情况

### 先说第 2 种情况

null 就是一个不合理的参数，就应该明确地中断程序，往外抛错误。这种情况常见于 api 方法。例如你开发了一个接口，id 是一个必选的参数，如果调用方没传这个参数给你，当然不行。你要感知到这个情况，告诉调用方“嘿，哥们，你传个 null 给我做甚”。

相对于判空语句，更好的检查方式有两个

1. assert 语句，你可以把错误原因放到 assert 的参数中，这样不仅能保护你的程序不往下走，而且还能把错误原因返回给调用方，岂不是一举两得。（原文介绍了 assert 的使用，这里省略）
2. 也可以直接抛出空指针异常。上面说了，此时 null 是个不合理的参数，有问题就是有问题，就应该大大方方往外抛。

### 第 1 种情况会更复杂一些。

这种情况下，null 是个“看上去”合理的值，例如，我查询数据库，某个查询条件下，就是没有对应值，此时 null 算是表达了“空”的概念。

这里给一些实践建议：

- 假如方法的返回类型是 collections，当返回结果是空时，你可以返回一个空的 collections (empty list),而不要返回 null.这样调用侧就能大胆地处理这个返回，例如调用侧拿到返回后，可以直接 print list.size(), 又无

需担心空指针问题。（什么？想调用这个方法时，不记得之前实现该方法有没按照这个原则？所以说，代码习惯很重要！如果你养成习惯，都是这样写代码（返回空 collections 而不返回 null），你调用自己写的方法时，就能大胆地忽略判空）

- 返回类型不是 collections，又怎么办呢？那就返回一个空对象（而非 null 对象），下面举个“栗子”，假设有如下代码

```
public interface Action {  
    void doSomething();  
  
public interface Parser {  
    Action findAction(String userInput);  
}
```

其中，Parser 有一个接口 findAction，这个接口会依据用户的输入，找到并执行对应的动作。假如用户输入不对，可能就找不到对应的动作（Action），因此 findAction 就会返回 null，接下来 action 调用 doSomething 方法时,就会出现空指针。 解决这个问题的一种方式，就是使用 Null Object pattern（空对象模式）

我们来改造一下

类定义如下，这样定义 findAction 方法后，确保无论用户输入什么，都不会返回 null 对象：

```
public class MyParser implements Parser {  
    private static Action DO_NOTHING = new Action() {  
        public void doSomething() { /* do nothing */ }  
    };  
  
    public Action findAction(String userInput) {  
        // ...  
        if ( /* we can't find any actions */ ) {  
            return DO_NOTHING;  
        }  
    }  
}
```

对比下面两份调用实例

## 1. 冗余: 每获取一个对象, 就判一次空

```
Parser parser = ParserFactory.getParser();
if (parser == null) {
    // now what?
    // this would be an example of where null isn't (or shouldn't be) a valid
    response
}
Action action = parser.findAction(someInput);
if (action == null) {
    // do nothing}
else {
    action.doSomething();}
```

## 2. 精简

```
ParserFactory.getParser().findAction(someInput).doSomething();
```

因为无论什么情况, 都不会返回空对象, 因此通过 `findAction` 拿到 `action` 后, 可以放心地调用 `action` 的方法。

### 其他回答精选:

- 如果要用 `equal` 方法, 请用 `object<不可能为空>.equal(object<可能为空>))` 例如: 使用 `"bar".equals(foo)` 而不是 `foo.equals("bar")`
- Java8 或者 guava lib 中, 提供了 `Optional` 类, 这是一个元素容器, 通过它来封装对象, 可以减少判空。不过代码量还是不少。不爽。
- 如果你想返回 `null`, 请停下来想一想, 这个地方是否更应该抛出一个异常

stackoverflow 链接: <http://stackoverflow.com/questions/271526/avoiding-null-statements-in-java?page=2&tab=votes#tab-top>

## 2. 获取完整的堆栈信息

---

## 获取完整的堆栈信息

### 问题

捕获了异常后，如何获取完整的堆栈轨迹（stack trace）

### 回答

```
String fullStackTrace =  
org.apache.commons.lang.exception.ExceptionUtils.getFullStackTrace(e)  
Thread.currentThread().getStackTrace();
```

stackoverflow 原址: <http://stackoverflow.com/questions/1069066/how-can-i-get-the-current-stack-trace>

## 3. 如何用一行代码初始化一个 ArrayList

### 如何用一行代码初始化一个 ArrayList

#### 问题

为了测试，我需要临时快速创建一个 list。一开始我这样做：

```
ArrayList<String> places = new ArrayList<String>();  
places.add("Buenos Aires");  
places.add("Córdoba");  
places.add("La Plata");
```

之后我重构了下

```
ArrayList<String> places = new ArrayList<String>(  
    Arrays.asList("Buenos Aires", "Córdoba", "La Plata"));
```

是否有更加简便的方法呢？

#### 回答

## 常见方式

实际上，也许“最好”的方式，就是你写的这个方式，因为它不用再创建新的

List:

```
ArrayList<String> list = new ArrayList<String>();  
list.add("A");  
list.add("B");  
list.add("C");
```

只是这个方式看上去要多写些代码，让人郁闷

## 匿名内部类

当然，还有其他方式，例如,写一个匿名内部类，然后在其中做初始化（也被称为 brace initialization）：

```
ArrayList<String> list = new ArrayList<String>() {{  
    add("A");  
    add("B");  
    add("C");  
}};
```

但是，我不喜欢这个方式。只是为了做个初始化，却要在 `ArrayList` 的同一行后面加这么一坨代码。

## Arrays.asList

```
List<String> places = Arrays.asList("Buenos Aires", "Córdoba", "La Plata");
```

## Collections.singletonList

```
List<String> places = Collections.singletonList("Buenos Aires");
```

注意：后面的这两种方式，得到的是一个定长的 List(如果 add 操作会抛异常)。如果你需要一个不定长的 List,可以这样做：

```
ArrayList<String> places = new ArrayList<>(Arrays.asList("Buenos Aires",  
"Córdoba", "La Plata"));
```



stackoverflow 链

接: <http://stackoverflow.com/questions/1005073/initialization-of-an-arraylist-in-one-line>

## 4. 初始化静态 map

---

### 初始化静态 map

#### 问题

怎么在 Java 中初始化一个静态的 map

我想到的两种方法如下, 大家是否有更好的建议呢?

方法一: static 初始化器

方法二: 实例初始化 (匿名子类)

下面是描述上面两种方法的例子

```
import java.util.HashMap;
import java.util.Map;
public class Test{
    private static final Map<Integer, String> myMap = new
HashMap<Integer, String>();
    static {
        myMap.put(1, "one");
        myMap.put(2, "two");
    }

    private static final Map<Integer, String> myMap2 = new
HashMap<Integer, String>(){
        {
            put(1, "one");
            put(2, "two");
        }
    };
};
```

```
}
```

## 答案

### 答案 1

匿名子类初始化器是 java 的语法糖，我搞不明白为什么要用匿名子类来初始化，而且，如果类是 final 的话，它将不起作用

我使用 static 初始化器来创建一个固定长度的静态 map

```
public class Test{
    private static final Map<Integer, String> myMap;
    static{
        Map<Integer, String> aMap = ...;
        aMap.put(1,"one");
        aMap.put(2,"two");
        myMap = Collections.unmodifiableMap(aMap);
    }
}
```

### 答案 2

我喜欢用 Guava（是 Collection 框架的增强）的方法初始化一个静态的，不可改变的 map

```
static final Map<Integer, String> myMap = ImmutableMap.of(
    1, "one",
    2, "two"
)
```

· 当 map 的 entry 个数超过 5 个时，你就不能使用 ImmutableMap.of。可以试试

```
ImmutableMap.builder()
static final Map<Integer, String> myMap = ImmutableMap.<Integer,
String>builder()
{
    .put(1, "one")
```

```
.put(2, "two")

.put(15, "fifteen")
.build();
}
```

原文链接

<http://stackoverflow.com/questions/507602/how-can-i-initialize-a-static-map>

## 5. 给 3 个布尔变量，当其中有 2 个或者 2 个以上为 true 才返回 true

---

给 3 个布尔变量，当其中有 2 个或者 2 个以上为 **true** 才返回 **true**

问题

给 3 个 boolean 变量，a,b,c，当其中有 2 个或 2 个以上为 true 时才返回 true？

- 最笨的方法：

```
boolean atLeastTwo(boolean a, boolean b, boolean c)
{
    if ((a && b) || (b && c) || (a && c))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

- 优雅解法 1

```
return a ? (b || c) : (b && c);
```

- 优雅解法 2

```
return (a==b) ? a : c;
```

- 优雅解法 3

```
return a ^ b ? c : a
```

- 优雅解法 4

```
return a ? (b || c) : (b && c);
```

stackoverflow 链接: <http://stackoverflow.com/questions/3076078/check-if-at-least-two-out-of-three-booleans-are-true>

## 6. 输出 Java 数组最简单的方式

### 输出 Java 数组最简单的方式

#### 问题

因为 Java 数组中没有 toString() 方法, 所以我如果直接调用数组 toString() 方法的话, 只会得到它的内存地址。像这样, 显得并不人性化:

```
int[] intArray = new int[] {1, 2, 3, 4, 5};  
System.out.println(intArray);    // 有时会输出 '[I@3343c8b3'
```

所以输出一个数组最简单的方法是什么? 我想要的效果是

// 数字数组:

```
int[] intArray = new int[] {1, 2, 3, 4, 5};  
//输出: [1, 2, 3, 4, 5]
```

// 对象数组:

```
String[] strArray = new String[] {"John", "Mary", "Bob"};  
//输出: [John, Mary, Bob]
```

## 回答

在 Java 5+ 以上中使用 `Arrays.toString(arr)` 或 `Arrays.deepToString(arr)` 来打印（输出）数组。

不要忘了引入 `import java.util.Arrays;`

```
package packageName;
import java.util.Arrays;
int[] intArray = new int[] {1, 2, 3, 4, 5};
System.out.println(Arrays.toString(intArray));
//输出: [1, 2, 3, 4, 5]

String[] strArray = new String[] {"John", "Mary", "Bob"};
System.out.println(Arrays.deepToString(strArray));
*//输出: [John, Mary, Bob]
```

`Arrays.deepToString` 与 `Arrays.toString` 不同之处在于, `Arrays.deepToString` 更适合打印多维数组

比如:

```
String[][] b = new String[3][4];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            b[i][j] = "A" + j;
        }
    }
System.out.println(Arrays.toString(b));
//输出[[Ljava.lang.String;@55e6cb2a,
[Ljava.lang.String;@23245e75, [Ljava.lang.String;@28b56559]
System.out.println(Arrays.deepToString(b));
//输出[[A0, A1, A2, A3], [A0, A1, A2, A3], [A0, A1, A2,
A3]]
```

stackoverflow 链接: <http://stackoverflow.com/questions/409784/whats-the-simplest-way-to-print-a-java-array>

## 7. 为什么以下用随机生成的文字会得出 “hello world”?

为什么以下用随机生成的文字会得出 “hello world”?

### 问题

为什么以下用随机生成的文字会得出"hello world". 有人能解释一下吗?

```
System.out.println(randomString(-229985452) + " " + randomString(-147909649));

public static String randomString(int i)
{
    Random ran = new Random(i);
    StringBuilder sb = new StringBuilder();
    while (true)
    {
        int k = ran.nextInt(27);
        if (k == 0)
            break;

        sb.append((char)('`' + k));
    }
    return sb.toString();
}
```

### 回答 1 (最佳)

在 JAVA 里面，随机类的实现不是真正的随机,是伪随机. 就是说如果随机类的种子是一样的话，他们会生成同一组的数字。

比如说这个问题:

```
new Random(-229985452).nextInt(27)
```

首 6 个生成的数字一定是:

```
8
5
12
12
15
0
```

而 `new Random(-147909649).nextInt(27)` 首 6 个生成的数字一定是:

```
23
15
18
12
4
0
```

而把每一个数目字加 ` (which is 96), 就会得到了相应的英文字母:

```
8 + 96 = 104 --> h
5 + 96 = 101 --> e
12 + 96 = 108 --> l
12 + 96 = 108 --> l
15 + 96 = 111 --> o

23 + 96 = 119 --> w
15 + 96 = 111 --> o
18 + 96 = 114 --> r
12 + 96 = 108 --> l
4 + 96 = 100 --> d
```

stackoverflow 链接: <http://stackoverflow.com/questions/15182496/why-does-this-code-using-random-strings-print-hello-world>

## 8. 什么在 java 中存放密码更倾向于 `char[]`而不是 `String`

---

为什么在 **java** 中存放密码更倾向于 **`char[]`**而不是 **`String`**

问题

在 Swing 中，password 字段有一个 getPassword()方法（返回 char[]），而不是通常的 getText()方法(返回 String 字符串)。同样的，我看到一个建议说不要使用字符串处理密码。为什么在涉及 passwords 时，都说字符串会对安全构成威胁？感觉使用 char[]不是那么的方便。

## 回答

String 是不可变的。虽然 String 加载密码之后可以把这个变量扔掉，但是字符串并不会马上被 GC 回收，一旦进程在 GC 执行到这个字符串之前被 dump，dump 出的转储中就会含有这个明文的字符串。那如果我去“修改”这个字符串，比如把它赋一个新值，那么是不是就没有这个问题了？答案是否定的，因为 String 本身是不可修改的，任何基于 String 的修改函数都是返回一个新的字符串，原有的还会在内存里。

然而对于数组，你可以在抛弃它之前直接修改掉它里面的内容或者置为乱码，密码就不会存在了。但是如果你什么也不做直接交给 gc 的话，也会存在上面一样的问题。

所以，这是一个安全性的问题--但是，即使使用 char[]也仅仅是降低了攻击者攻击的机会，而且仅仅对这种特定的攻击有效。

**stackoverflow 链接：** <http://stackoverflow.com/questions/8881291/why-is-char-preferred-over-string-for-passwords-in-java>

知乎上也有相关讨论： <https://www.zhihu.com/question/36734157>

## 9. 如何避免在 JSP 文件中使用 Java 代码

---

如何避免在 JSP 文件中使用 Java 代码



## 问题

如何避免在 JSP 文件中使用 Java 代码？

我对 Java EE 不是很熟悉，我知道类似如下的三行代码

```
<%= x+1 %>  
<%= request.getParameter("name") %>  
<%! counter++; %>
```

这三行代码是学校教的老式代码。在 JSP 2，存在一些方法可以避免在 JSP 文件中使用 Java 代码。有人可以告诉我在 JSP 2 中如何避免使用 Java 代码吗，这些方法该如何使用？

## 回答

在大约十年前，taglibs（比如 JSTL）和 EL（EL 表达式，`${}`）诞生的时候，在 JSP 中使用 scriptlets（类似`<% %>`）这种做法，就确实已经是不被鼓励使用的做法了。

scriptlets 主要的缺点有：

1. **重用性**：你不可以重用 scriptlets
2. **可替换性**：你不可以让 scriptlets 抽象化
3. **面向对象能力**：你不可以使用继承或组合
4. **调试性**：如果 scriptlets 中途抛出了异常，你只能获得一个空白页
5. **可测试性**：scriptlets 不能进行单元测试
6. **可维护性**：（这句有些词语不确定）需要更多的时间去维护混合的/杂乱的/冲突的 代码逻辑

Oracle 自己也在 [JSP coding conventions](#) 一文中推荐在功能可以被标签库所替代的时候避免使用 `scriptlets` 语法。以下引用它提出的几个观点：

在 JSP 1.2 规范中，强烈推荐使用 JSTL 来减少 JSP `scriptlets` 语法的使用。一个使用 JSTL 的页面，总得来说会更加地容易阅读和维护。

...

在任何可能的地方，当标签库能够提供相同的功能时，尽量避免使用 JSP `scriptlets` 语法。这会让页面更加容易阅读和维护，帮助将 业务逻辑 从 表现层逻辑 中分离，也会让页面往更符合 JSP 2.0 风格的方向发展（JSP 2.0 规范中，支持但是极大弱化了 JSP `scriptlets` 语法）

...

本着适应 模型-显示层-控制器（MVC） 设计模式中关于减少业务逻辑层与显示层之间的耦合的精神，**JSP `scriptlets` 语法不应该被用来编写业务逻辑**。相应的，JSP `scriptlets` 语法在传送一些服务端返回的处理客户端请求的数据（也称为 value objects）的时候会被使用。尽管如此，使用一个 `controller servlet` 来处理或者用自定义标签来处理会更好。

---

如何替换 **`scriptlets`** 语句，取决于代码/逻辑的目的。更常见的是，被替换的语句会被放在另外的一些更值得放的 **Java** 类里（这里翻译得不一定清楚）

- 如果你想在每个请求、每个页面请求都运行**相同的 Java** 代码，，比如说检查一个用户是否在登录状态，就要实现一个 过滤器，在 `doFilter()` 方法中编写正确的代码，例如

```

public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws ServletException, IOException {
    if (((HttpServletRequest) request).getSession().getAttribute("user") ==
    null) {
        ((HttpServletResponse) response).sendRedirect("login"); // Not
        logged in, redirect to login page.
    } else {
        chain.doFilter(request, response); // Logged in, just continue
        request.
    }
}

```

当你在<url-pattern>中做好恰当的地址映射，覆盖所有应该被覆盖的 JSP 文件，也就不需要再 JSP 文件中添加这些相同的 Java 代码

---

- 如果你想执行一些 Java 代码来**预处理**一个请求，例如，预加载某些从数据库加载的数据来显示在一些表格里，可能还会有一些查询参数，那么，实现一个 Servlet，在 doGet()方法里编写正确的代码，例如

```

protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
    try {
        List<Product> products = productService.list(); // Obtain all
        products.
        request.setAttribute("products", products); // Store products in
        request scope.
        request.getRequestDispatcher("/WEB-INF/products.jsp").forward(request, response); // Forward to JSP page to
        display them in a HTML table.
    } catch (SQLException e) {
        throw new ServletException("Retrieving products failed!", e);
    }
}

```

这个方法能够更方便地处理异常。这样会在渲染、展示 JSP 页面时访问数据库。在数据库抛出异常的时候，你可以根据情况返回不同的响应或页面。在上面的例子，出错时默认会展示 500 页面，你也可以改变 web.xml 的<error-page>来自定义异常处理错误页。

- 
- 如果你想执行一些 Java 代码来\*\*后置处理（postprocess）\*\*一个请求，例如处理表单提交，那么，实现一个 Servlet，在 doPost()里写上正确的代码：

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    User user = userService.find(username, password);

    if (user != null) {
        request.getSession().setAttribute("user", user); // Login user.
        response.sendRedirect("home"); // Redirect to home page.
    } else {
        request.setAttribute("message", "Unknown username/password. Please retry."); // Store error message in request scope.
        request.getRequestDispatcher("/WEB-INF/login.jsp").forward(request, response); // Forward to JSP page to redisplay login form with error.
    }
}
```

这个处理不同目标结果页的方法会比原来更加简单： 可以显示一个带有表单验证错误提示的表单（在这个特别的例子中，你可以用 EL 表达式`${message}`来显示错误提示），或者仅仅跳转到成功的页面

- 
- 如果你想执行一些 Java 代码来控制执行计划（control the execution plan） 和/或 request 和 response 的跳转目标，用 [MVC 模式](#)实现一个 Servlet，例如：

```
protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        Action action = ActionFactory.getAction(request);
        String view = action.execute(request, response);
    }
}
```

```

        if (view.equals(request.getPathInfo().substring(1)) {
            request.getRequestDispatcher("/WEB-INF/" + view +
".jsp").forward(request, response);
        } else {
            response.sendRedirect(view);
        }
    } catch (Exception e) {
        throw new ServletException("Executing action failed.", e);
    }
}

```

或者使用一些 MVC 框架例如 [JSF](#), [Spring MVC](#), [Wicket](#) 这样你就不用自定义 servlet，只要写一些页面和 javabeen class 就可以了。

---

- 如果你想执行一些 Java 代码来控制 **JSP 页面的数据渲染流程（control the flow inside a JSP page）**，那么你需要使用一些（已经存在的）流程控制标签库，比如 [JSTL core](#)，例如，在一个表格显示 `List<Product>`

```

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
...
<table>
    <c:forEach items="${products}" var="product">
        <tr>
            <td>${product.name}</td>
            <td>${product.description}</td>
            <td>${product.price}</td>
        </tr>
    </c:forEach>
</table>

```

这些 XML 风格的标签可以很好地适应 HTML 代码，代码变得更好阅读（也因此更好地维护），相比于杂乱无章的 scriptlets 的分支大括号（Where the heck does this closing brace belong to?（到底这个结束大括号是属于哪个代码段的？））。一个简单的设置可以配置你的 Web 程序让在使用 scriptlets 的时候自动抛出异常

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>true</scripting-invalid>
  </jsp-property-group>
</jsp-config>
```

在 JSP 的继承者 [Facelets](#) 里（Java EE 提供的 MVC 框架 [JSF](#)），已经不可能使用 scriptlets 语法了。这是一个让你强制使用“正确的方法”的方法

- 
- 如果你想执行一些 Java 代码来在 JSP 中 **访问和显示** 一些“后端”数据，你需要使用 EL（表达式），`${}`，例如，显示已经提交的数值：

```
<input type="text" name="foo" value="${param.foo}" />
```

`${param.foo}` 会显示 `request.getParameter("foo")` 这句话的输出结果。

- 
- 如果你想在 JSP 直接执行一些工具类 Java 代码（典型的，一些 public static 方法），你需要定义它，并使用 EL 表达式函数。这是 JSTL 里的标准函数标签库，但是你也可以[轻松地创建自己需要的功能](#)，下面是一个使用有用的 `fn:escapeXml` 来避免 XSS 攻击的例子。

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
...
<input type="text" name="foo" value="${fn:escapeXml(param.foo)}" />
```

注意，XSS 并不是 Java/JSP/JSTL/EL/任何技术相关的东西，这个问题是任何 Web 应用程序都需要关心的问题，scriptlets 并没有为这个问题提供良好的解决方案，至少没有标准的 Java API 的解决方案。JSP 的继承者 Facelets 内含了 HTML 转义功能，所以在 Facelets 里你不用担心 XSS 攻击的问题。

See Also:

- [JSP, Servlet, JSF 的不同点在哪里?](#)
- [Servlet, ServletContext, HttpSession 和 HttpServletRequest/Response 是如何工作的?](#)
- [JSP, Servlet and JDBC 的基本 MVC 例子](#)
- [Java Web 应用程序中的设计模式](#)
- [JSP/Servlet 中的隐藏功能](#)

stackoverflow 原址: <http://stackoverflow.com/questions/3177733/how-to-avoid-java-code-in-jsp-files>

## **10. Java 源码里的设计模式**

---

### **Java 源码里的设计模式**

从 [维基百科](#) 中,可以让你对大部分设计模式有一个概览,而且它也指出了哪些设计模式是 GoF 中规范.下面列出可以从 JavaSE 和 JavaEE API 中找到的设计模式:

#### **创建型模式**

#### **抽象工厂**

- [javax.xml.parsers.DocumentBuilderFactory#newInstance\(\)](#)
- [javax.xml.transform.TransformerFactory#newInstance\(\)](#)
- [javax.xml.xpath.XPathFactory#newInstance\(\)](#)

## 建造者模式

- [java.lang.StringBuilder#append\(\)](#)(非同步)
- [java.lang.StringBuffer#append\(\)](#)(同步)
- [java.nio.ByteBuffer#put\(\)](#)(类似的还有, [CharBuffer](#), [ShortBuffer](#), [IntBuffer](#), [LongBuffer](#), [FloatBuffer](#) 和 [DoubleBuffer](#))
- [javax.swing.GroupLayout.Group#addComponent\(\)](#)

## 工厂模式

- [java.util.Calendar#getInstance\(\)](#)
- [java.util.ResourceBundle#getBundle\(\)](#)
- [java.text.NumberFormat#getInstance\(\)](#)
- [java.nio.charset.Charset#forName\(\)](#)
- [java.net.URLStreamHandlerFactory#createURLStreamHandler\(String\)](#)

## 原型模式

- [java.lang.Object#clone\(\)](#)(类需要实现 [java.lang.Cloneable](#) 接口)

## 单例模式

- [java.lang.Runtime#getRuntime\(\)](#)
- [java.awt.Desktop#getDesktop\(\)](#)



- [java.lang.System#getSecurityManager\(\)](#)

## 结构型模式

## 适配器模式

- [java.util.Arrays#asList\(\)](#)
- [java.io.InputStreamReader\(InputStream\)](#) (返回 Reader)
- [java.io.OutputStreamWriter\(OutputStream\)](#)(返回 Writer)
- [javax.xml.bind.annotation.adapters.XmlAdapter#marshal\(\)](#) 和 [#unmarshal\(\)](#)

## 桥模式

暂时没有发现

## 合成模式

- [java.awt.Container#add\(Component\)](#)(Swing 中几乎所有类都使用)
- [javax.faces.component.UIComponent#getChildren\(\)](#)(JSF UI 中几乎所有类都使用)

## 装饰模式

- [java.io.InputStream,OutputStream,Reader](#) 和 [Writer](#) 的所有资料都有一个使用 `InputStream,OutputStream,Reader,Writer` 的构造器

- [java.util.Collections](#) 中的 [checkedXXX\(\)](#), [synchronizedXXX\(\)](#) 和 [unmodifiableXXX\(\)](#) 方法
- [javax.servlet.http.HttpServletRequestWrapper](#) 和 [HttpServletResponseWrapper](#)

## 门面模式

[javax.faces.context.FacesContext](#), 其内部使用

了 [Lifecycle](#), [ViewHandler](#), [NavigationHandler](#) 等接口或抽象类，没有这一个门面类，终端就需要考虑如何去使用接口或抽象类（实际上不需要，因为门面类通过反射完成了）[javax.faces.context.ExternalContext](#), 其内部使用

了 [ServletContext](#), [HttpSession](#), [HttpServletRequest](#), [HttpServletResponse](#) 等

## 享元模式

- [java.lang.Integer#valueOf\(int\)](#)，类似得还有 [Boolean](#), [Byte](#), [Character](#), [Short](#) 和 [Long](#)

## 代理模式

- [java.lang.reflect.Proxy](#)
- [java.rmi.\\*](#)(所有 api)

## 表现型模式

---

## 责任链模式

- [java.util.logging.Logger#log\(\)](#)
- [javax.servlet.Filter#doFilter\(\)](#)

## 命令模式

- 所有 [java.lang.Runnable](#) 的实现
- 所有 [javax.swing.Action](#) 的实现

## 解释器模式

- [java.util.Pattern](#)
- [java.text.Normalizer](#)
- 所有 [java.text.Format](#) 的子类
- 所有 [javax.el.ELResolver](#) 的子类

## 迭代模式

- 所有 [java.util.Iterator](#) 的实现(因此也包含了所有 [java.util.Scanner](#) 的子类)
- 所有 [java.util.Enumeration](#) 的实现

## 中介模式

- [java.util.Timer](#) 中的所有 [scheduleXXX\(\)](#) 方法)
- [java.util.concurrent.Executor#execute\(\)](#)

- [java.util.concurrent.ExecutorService](#) 中的 [invokeXXX\(\)](#) 和 [submit\(\)](#) 方法
- [java.util.concurrent.ScheduledExecutorService](#) 中的所有 [scheduleXXX\(\)](#) 方法
- [java.lang.reflect.Method#invoke\(\)](#)

## 备忘录模式

[java.util.Date](#)([setXXX](#) 方法更新的就是其内部的 [Date](#) 的值) [java.io.Serializable](#) 的所有实现 [javax.faces.component.StateHolder](#) 的所有实现

## 观察者模式（订阅模式）

[java.util.Observer](#)/[java.util.Observable](#)(实际应用中，很少会用到) [java.util.EventListener](#) 的所有实现(几乎包含了所有 [Swing](#) 中使用到的类) [javax.servlet.http.HttpSessionBindingListener](#) [javax.servlet.http.HttpSessionAttributeListener](#) [javax.faces.event.PhaseListener](#)

## 状态模式

[javax.faces.lifecycle.Lifecycle#execute\(\)](#)(由 [FacesServlet](#) 控制,行为是依赖于当前 JSF 生命周期阶段(状态))

## 策略模式

[java.util.Comparator#compare\(\)](#), 在 [Collections#sort\(\)](#) 中会使用到. [javax.servlet.http.HttpServlet.service\(\)](#) 和 所有 [doXXX\(\)](#) 方法都以

HttpServletRequest 和 HttpServletResponse 作为参数，所有方法的实现都需要显式处理这两个参数(而不是持有这个变量。) [javax.servlet.Filter#doFilter\(\)](#)

## 模板模式

[java.io.InputStream](#), [java.io.OutputStream](#), [java.io.Reader](#) 和 [java.io.Writer](#) 的所有非抽象方法。[java.util.AbstractList](#), [java.util.AbstractSet](#) 和 [java.util.AbstractMap](#) 的所有非抽象方法。

[javax.servlet.http.HttpServlet](#) 中 [doXXX\(\)](#) 方法,这些方法默认返回 405

"Method Not Allowed" ，你可以自由地选择覆盖实现其中的一个或多个。

## 访问者模式

[javax.lang.model.element.AnnotationValue](#) 和 [AnnotationValueVisitor](#) [javax.lang.model.element.Element](#) 和 [ElementVisitor](#) [javax.lang.model.type.TypeMirror](#) 和 [TypeVisitor](#) [java.nio.file.FileVisitor](#) 和 [SimpleFileVisitor](#)

附录拓展：

- [设计模式-百度百科](#)
- stackoverflow 原

址: <http://stackoverflow.com/questions/1673841/examples-of-gof-design-patterns-in-javas-core-libraries>

## 11. 如何产生一个随机的字母数字串作为 session 的唯一标识符

如何产生一个随机的字母数字串作为 **session** 的唯一标识符?

如果允许产生的随机字符串是可猜测的(随机字符串比较都短,或者使用有缺陷的随机数生成器),进而导致攻击者可能会劫持到会话的,可以使用一个相对简单随机数生成代码,如下所示:

```
public class RandomString {

    private static final char[] symbols;

    static {
        StringBuilder tmp = new StringBuilder();
        for (char ch = '0'; ch <= '9'; ++ch)
            tmp.append(ch);
        for (char ch = 'a'; ch <= 'z'; ++ch)
            tmp.append(ch);
        symbols = tmp.toString().toCharArray();
    }

    private final Random random = new Random();

    private final char[] buf;

    public RandomString(int length) {
        if (length < 1)
            throw new IllegalArgumentException("length < 1: " + length);
        buf = new char[length];
    }

    public String nextString() {
        for (int idx = 0; idx < buf.length; ++idx)
            buf[idx] = symbols[random.nextInt(symbols.length)];
        return new String(buf);
    }
}
```

为了安全,可以考虑使用下面这段简洁且安全的代码,不过用其作为 session 的标识符,倒显得有点大材小用了（比较耗时）:

```
import java.security.SecureRandom;

public final class SessionIdentifierGenerator {
    private SecureRandom random = new SecureRandom();

    public String nextSessionId() {
        return new BigInteger(130, random).toString(32);
    }
}
```

其工作原理就是，使用一个 130 位的安全的随机数生成器生成一个随机数，接着转化为 32 进制。我们知道，128 位安全随机数的生成已经是足够安全的，不过以 32 进制编码的每一个数字可编码 5 位，所以需要取大于 128 且是 5 的倍数，所以就选择了 130 位。相对于 随机 UUID 来说(在标准输出中，每个字符使用 3.4 bit，共 122 bit)，每个字符使用 5 个随机的 bit 来编码的方式，显得更为简洁和高效。

译者注：上面两段代码，生成 26 位随机字符串，第一段代码每次耗时不到 1ms，第二段耗时约 100ms。也就是说第一段代码更快，但第二段代码更安全，但更耗时。

stackoverflow 原链接: <http://stackoverflow.com/questions/41107/how-to-generate-a-random-alpha-numeric-string>

## 12. 如何创建单例

---

如何创建单例 ？

问题

Java 创建单例有哪些方式？

## 解答

实现单例，从加载方式来看，有两种：

- 预加载
- 懒加载

先看一下实现单例最简单的方式(预加载)：

```
public class Foo {  
  
    private static final Foo INSTANCE = new Foo();  
  
    private Foo() {  
        if (INSTANCE != null) {  
            throw new IllegalStateException("Already instantiated");  
        }  
    }  
  
    public static Foo getInstance() {  
        return INSTANCE;  
    }  
}
```

再来看一下懒加载的方式：

```
class Foo {  
  
    private static Foo INSTANCE = null;  
  
    private Foo() {  
        if (INSTANCE != null) {  
            throw new IllegalStateException("Already instantiated");  
        }  
    }  
  
    public static Foo getInstance() {  
        if (INSTANCE == null) {  
            INSTANCE = new Foo();  
        }  
    }  
}
```



```

    }
    return INSTANCE;
}
}

```

以上方式在单线程的情况可以很好的满足需要，换言之，若是在多线程，还需要作一定的改进,如下所示:

```

class Foo {
    // 请注意 volatile 关键字的使用
    private static volatile Foo INSTANCE = null;

    private Foo() {
        if (INSTANCE != null) {
            throw new IllegalStateException("Already instantiated");
        }
    }

    public static Foo getInstance() {
        if (INSTANCE == null) { // Check 1
            synchronized (Foo.class) {
                if (INSTANCE == null) { // Check 2
                    INSTANCE = new Foo();
                }
            }
        }
        return INSTANCE;
    }
}

```

上述代码运用了 [Double-Checked Locking idiom](#)。

解决了多线程环境下的单例，可以进一步思考如何实现可序列化的单例？反序列化可以不通过构造函数直接生成一个对象，所以反序列化时，我们需要保证其不再创建新的对象。

```

class Foo implements Serializable {

    private static final long serialVersionUID = 1L;

    private static volatile Foo INSTANCE = null;

    private Foo() {
        if (INSTANCE != null) {

```

```

        throw new IllegalStateException("Already instantiated");
    }
}

public static Foo getInstance() {
    if (INSTANCE == null) { // Check 1
        synchronized (Foo.class) {
            if (INSTANCE == null) { // Check 2
                INSTANCE = new Foo();
            }
        }
    }
    return INSTANCE;
}

@SuppressWarnings("unused")
private Foo readResolve() {
    return INSTANCE;
}
}

```

`readResolve` 方法可以保证，即使程序在上一次运行时序列化过此单例，也只会返回全局唯一的单例。对于 Java 对象序列化机制，可参考[附录拓展](#)。

java 创建单例的方法基本实现了，不过我们还可以作进一步的改进 —— 代码重构:

```

public final class Foo implements Serializable {

    private static final long serialVersionUID = 1L;

    // 使用内部静态 class 实现懒加载
    private static class FooLoader {
        // 保证在多线程环境下无差错运行
        private static final Foo INSTANCE = new Foo();
    }

    private Foo() {
        if (INSTANCE != null) {
            throw new IllegalStateException("Already instantiated");
        }
    }
}

```

```

    public static Foo getInstance() {
        return FooLoader.INSTANCE;
    }

    @SuppressWarnings("unused")
    private Foo readResolve() {
        return FooLoader.INSTANCE;
    }
}

```

好了，现在已经很完美实现了单例的创建，是不是很高兴。单例实现的基本原理，我们已经基本清楚里，最后提供一种更加简洁方法,如下:

```

public enum Foo {
    INSTANCE;
}

```

08 年 google 开发者年会中，Joshua Bloch 在 [高效 Java 话题中](#) 解释了这种方法，视频请戳 [这里](#). 在他[演讲的 ppt](#) 30-32 页提到:

实现单例正确的方式如下:

```

...
public enum Elvis {
    INSTANCE;
    private final String[] favoriteSongs =
        { "Hound Dog", "Heartbreak Hotel" };
    public void printFavorites() {
        System.out.println(Arrays.toString(favoriteSongs));
    }
}
...

```

在 [高效 Java 线上部分](#) 有说到:

上述实现单例的方式，其实等同于，将 `INSTANCE` 设置为 `public static final` 的方式，不同之处在于，使用枚举的方式显得更为简洁，且默认提供了序列化机制，也保证了多线程访问的安全。虽然这种单例的实现方式还未被广泛使用，可实现单例的最好方式就是使用一个单元素的枚举。

为什么可以这么简洁？因为 Java 中每一个枚举类型都默认继承了

`java.lang.Enum`，而 `Enum` 实现了 `Serializable` 接口，所以枚举类型对象都是默认可以被序列化的。通过反编译，也可以知道枚举常量本质上就是一个

```
public static final xxx
```

对于枚举的进一步理解，请参考[附录拓展](#)。

附录拓展: [深入理解 Java 对象序列

化](<http://developer.51cto.com/art/201202/317181.htm>) [对象的序列化和反序

列化](<http://www.blogjava.net/lingy/archive/2008/10/10/233630.html>) [通过反

编译字节码来理解 Java 枚举]([http://unmi.cc/understand-java-enum-with-](http://unmi.cc/understand-java-enum-with-bytecode/)

bytecode/)

stackoverflow 原址: [http://stackoverflow.com/questions/70689/what-is-an-](http://stackoverflow.com/questions/70689/what-is-an-efficient-way-to-implement-a-singleton-pattern-in-java)

[efficient-way-to-implement-a-singleton-pattern-in-java](http://stackoverflow.com/questions/70689/what-is-an-efficient-way-to-implement-a-singleton-pattern-in-java)

## **13. 实现 Runnable 接口 VS. 继承 Thread 类**

### **实现 Runnable 接口 VS. 继承 Thread 类 ?**

在 Java 中，并发执行任务一般有两种方式： （1）实现 Runnable 接口 （2）

继承 Thread 类

一般而言，推荐使用方式（1），主要是由于大多数情况下，人们并不会特别去

关注线程的行为，也不会去改写 Thread 已有的行为或方法，仅仅是期望执行任务而已。因此，使用接口的方式能避免引入一些并不需要的东西，同时也不会影响继承其他类，并使程序更加灵活。

### **额外的 tips:**

（1）Runnable 与 Thread 不是对等的概念 在 Thinking in Java 中，作者吐槽过 Runnable 的命名，称其叫做 Task 更为合理。在 Java 中，Runnable 只是一段

用于描述任务的代码段而已，是静态的概念，需要通过线程来执行。而 Thread 更像是一个活体，自身就具有很多行为，能够用来执行任务。

(2) 仅仅当你确实想要重写 (override) 一些已有行为时，才使用继承，否则请使用接口。

(3) 在 Java 5 之前，创建了 Thread 却没调用其 start() 方法，可能导致内存泄露。

stackoverflow 链

接: <http://stackoverflow.com/questions/541487/implements-runnable-vs-extends-thread>

## 14. 我应该用哪一个@NotNull 注解

### 我应该用哪一个@NotNull 注解?

我希望能通过注解的方式，尽量避免程序中出现空指针问题，同时既能保障代码的可读性，又能和 IDE 的代码检查，静态代码扫描工具结合起来。相关的注解，我看到有好多种@NotNull/@NonNull/@Nonnull，而他们彼此间又有冲突，不能共用，下面是我找到的一些注解，哪个是最好的选择呢？

#### 1.javax.validation.constraints.NotNull

运行时进验证，不静态分析

#### 2.edu.umd.cs.findbugs.annotations.NonNull

用于 finbugs 和 Sonar 静态分析

#### 3.javax.annotation.Nonnull

只适用 FindBugs, JSR-305 不适用

#### 4.org.jetbrains.annotations.NotNull

适用于 IntelliJ IDEA 静态分析

## 5.lombok.NonNull

适用 Lombok 项目中代码生成器。不是一个标准的占位符注解。

## 6.android.support.annotation.NonNull

适用于 Android 项目的标记注解,位于 support-annotations 包中

## 回答

---

我推荐用 javax 命名空间下的注解(虽然我喜欢 Lombok 和 IntelliJ 做的事情), 使用其他命名空间的注解, 等于你还需要引入其他依赖。

我用 javax.validation.constraints.NotNull, 因为它已经在 Java EE 6 中定义

javax.annotation.NotNull 可能直到 java 8 都不存在(正如 Stephen 指出)。其他的都不是标准的注解。

如果注解是可扩展的, 那将是一件美好的事情.你可以自己写一个 non-null 注解, 然后继承上面说的这些注解。如果标准的注解不支持某个特性, 你就可以在自己定义的注解里面扩展。

stackoverflow 链接: <http://stackoverflow.com/questions/4963300/which-notnull-java-annotation-should-i-use>

## 15. 怎样将堆栈追踪信息转换为字符串

---

### 怎样将堆栈追踪信息转换为字符串

#### 问题

将 Throwable.printStackTrace() 的结果转换为一个字符串来描述堆栈信息的最简单的方法是什么

## 最佳答案

可以用下面的方法将异常堆栈信息转换为字符串类型。该类在 Apache

commons-lang-2.2.jar 中可以找到：

`org.apache.commons.lang.exception.ExceptionUtils.getStackTrace(Throwable)`

## 答案二

用 `Throwable.printStackTrace(PrintWriter pw)` 可以输出堆栈信息：

```
StringWriter sw = new StringWriter();
PrintWriter pw = new PrintWriter(sw);
t.printStackTrace(pw);
sw.toString(); // stack trace as a string
```

## 答案三

```
StringWriter sw = new StringWriter();
e.printStackTrace(new PrintWriter(sw));
String exceptionAsString = sw.toString();
```

## 答案四

```
public String stackTraceToString(Throwable e) {
    StringBuilder sb = new StringBuilder();
    for (StackTraceElement element : e.getStackTrace()) {
        sb.append(element.toString());
        sb.append("\n");
    }
    return sb.toString();
}
```

stackoverflow 链接：<http://stackoverflow.com/questions/1149703/how-can-i-convert-a-stack-trace-to-a-string>

## 16. 如何处理 `java.lang.OutOfMemoryError` PermGen space error

---

### 如何处理 `java.lang.OutOfMemoryError PermGen space error`

#### 问题

最近，我在运行我的 web 应用时得到：`java.lang.OutOfMemoryError: PermGen space`。我的应用是一个典型的 Hibernate/JPA + IceFaces/JSF 的应用。运行于 Tomcat6.0 和 jdk1.6。我发布了多次以后，产生了这个错误。

是什么原因造成的，我如何避免？我怎样修复？

#### 回答

解决方案是当 Tomcat 启动时，在 JVM 的命令中添加参数

```
-XX:+CMSClassUnloadingEnabled -XX:+CMSPermGenSweepingEnabled
```

你也可以停止 tomcat 的服务，直接进入 Tomcat/bin 目录，运行 tomcat6w.exe。

在 Java 的标签下，加好上面的参数。单击"OK"，重新启动 Tomcat 的服务。

如果系统返回错误，提示指定的服务不存在，你可以运行：

```
tomcat6w //ES//servicename
```

servicename 的名字你可以在 services.msc 中查看。

stackoverflow 链接：<http://stackoverflow.com/questions/88235/dealing-with-java-lang-outofmemoryerror-permgen-space-error>



## 17. 如何在整数左填充 0

---

### 如何在整数左填充 0

#### 问题

如何在整数左填充 0 举例 1 = "0001"

#### 答案一，String.format

```
String.format("%05d", yournumber);
```

用 0 填充，总长度为

5 <https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html>

#### 答案二，ApacheCommonsLanguage

如果需要在 Java 1.5 前使用，可以利用 Apache Commons Language 方法

```
org.apache.commons.lang.StringUtils.leftPad(String str, int size, '0')
```

#### 答案三，DecimalFormat

```
import java.text.DecimalFormat;
class TestingAndQualityAssuranceDepartment
{
    public static void main(String [] args)
    {
        int x=1;
        DecimalFormat df = new DecimalFormat("00");
        System.out.println(df.format(x));
    }
}
```

#### 答案四，自己实现

如果效率很重要的话，相比于 `String.format` 函数的可以自己实现

```
/**
 * @param in The integer value
 * @param fill The number of digits to fill
 * @return The given value left padded with the given number of digits
 */
public static String lPadZero(int in, int fill){

    boolean negative = false;
    int value, len = 0;

    if(in >= 0){
        value = in;
    } else {
        negative = true;
        value = - in;
        in = - in;
        len ++;
    }

    if(value == 0){
        len = 1;
    } else{
        for(; value != 0; len ++){
            value /= 10;
        }
    }

    StringBuilder sb = new StringBuilder();

    if(negative){
        sb.append('-');
    }

    for(int i = fill; i > len; i--){
        sb.append('0');
    }

    sb.append(in);

    return sb.toString();
}
```

## 效率对比

```
public static void main(String[] args) {
    Random rdm;
    long start;

    // Using own function
    rdm = new Random(0);
    start = System.nanoTime();

    for(int i = 10000000; i != 0; i--){
        lPadZero(rdm.nextInt(20000) - 10000, 4);
    }
    System.out.println("Own function: " + ((System.nanoTime() - start) /
1000000) + "ms");

    // Using String.format
    rdm = new Random(0);
    start = System.nanoTime();

    for(int i = 10000000; i != 0; i--){
        String.format("%04d", rdm.nextInt(20000) - 10000);
    }
    System.out.println("String.format: " + ((System.nanoTime() - start) /
1000000) + "ms");
}
```

结果 自己的实现: 1697ms String.format: 38134ms

## 答案, Google Guava

Maven:

```
<dependency>
    <artifactId>guava</artifactId>
    <groupId>com.google.guava</groupId>
    <version>14.0.1</version>
</dependency>
```

样例:

```
Strings.padStart("7", 3, '0') returns "007"
Strings.padStart("2020", 3, '0') returns "2020"
```

注意： Guava 是非常有用的库，它提供了很多有用的功能，包括了 Collections, Caches, Functional idioms, Concurrency, Strings, Primitives, Ranges, IO, Hashing, EventBus 等

stackoverflow 原址： <http://stackoverflow.com/questions/473282/how-can-i-pad-an-integers-with-zeros-on-the-left>

## 18. 在调用 instanceof 前需要进行 null 检查吗

---

在调用 instanceof 前需要进行 null 检查吗

问题：

null instanceof SomeClass 会返回 null 还是抛出 NullPointerException 异常

答案一

在调用 instanceof 前不要进行 null 检查 null instanceof SomeClass 会返回 null 在 Java Language Specification

中 <http://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html#jls-15.20.2>

在运行时，如果该 instanceof 运算符的关系表达式（RelationExpression）不为 null，且这个引用可以被成功转型（ §15.16），不抛出 ClassCastException，则结果为 true；否则结果为 false。

答案二

```
public class IsInstanceOfTest {  
  
    public static void main(final String[] args) {
```

```

String s;

s = "";

System.out.println((s instanceof String));
System.out.println(String.class.isInstance(s));

s = null;

System.out.println((s instanceof String));
System.out.println(String.class.isInstance(s));
}
}

```

打印出

```

true
true
false
false

```

原文链接

<http://stackoverflow.com/questions/2950319/is-null-check-needed-before-calling-instanceof>

## 19. 如何从文件里读取字符串

如何从文件里读取字符串

从文件里读取所有文本：

代码：

```

static String readFile(String path, Charset encoding)
    throws IOException
{
    byte[] encoded = Files.readAllBytes(Paths.get(path));

```

```
return new String(encoded, encoding);  
}
```

## 一行一行读入文本：

Java 7 提供了一个方便的方法可以直接将文件中的文本一行一行读入，存放在一个 List 容器里。

```
List<String> lines = Files.readAllLines(Paths.get(path), encoding);
```

## 内存使用率

第一个方法，一次读取所有文本的方法，占用内存较多，因为它一次性保留了文件的所有原始信息，包括换行符之类的“无用”字符。

第二个方法，按行读入，比起一次性全部读入，要消耗更少的内存。因为它每次只将一行的文件信息放在缓存中。然而，如果文本文件很大，这种方法依然会占用很多内存。

如果你的程序需要处理很大的文本文件，在设计的时候就要考虑，分配一块固定的缓存，每次从流中读入文件的一部分放入缓存，处理，然后清空缓存，把下一部分读入缓存，直到处理完所有的数据。

这里的“很大”是相对于计算机性能的。一般来说，几十个 G 的文件应当算是大文件。

## 字符编码

还有一件事需要注意，就是字符编码。不同的平台有自己的默认编码，所以有时候你的程序需要指定编码，来保持平台无关/跨平台。

StandardCharsets 类定义了常用的编码类型，你可以用如下方法调用：

```
String content = readFile("test.txt", StandardCharsets.UTF_8);
```

可以通过 Charset 类来获得平台默认的字符编码。

```
String content = readFile("test.txt", Charset.defaultCharset());
```

注： 这个答案与之前 Java6 版本时的答案完全不同。Java 7 新增的工具类极大的优化了字符处理，文件读取等功能。Java 6 常用的内存映射方法已不适合在 Java 7 以后的版本使用。

## 原文链接

<http://stackoverflow.com/questions/326390/how-do-i-create-a-java-string-from-the-contents-of-a-file>

## 20. 遍历集合时移除元素，怎样避免

### ConcurrentModificationException 异常抛出

---

#### 遍历集合时移除元素，怎样避免

#### ConcurrentModificationException 异常抛出

问题：

在遍历集合的过程中，不会总出现 ConcurrentModificationException 异常的抛出，但是在下面的代码块中：

```
public static void main(String[] args) {
    Collection<Integer> l = new ArrayList<Integer>();

    for (int i=0; i < 10; ++i) {
        l.add(new Integer(4));
        l.add(new Integer(5));
        l.add(new Integer(6));
    }

    //遍历的过程中移除部分集合元素
    for (Integer i : l) {
        if (i.intValue() == 5) {
            l.remove(i);
        }
    }
}
```

```
System.out.println(1);  
}
```

运行之后，结果显而易见，总是会抛出异常：

```
Exception in thread "main" java.util.ConcurrentModificationException
```

所以，遍历集合时移除元素，怎样避免 `ConcurrentModificationException` 异常的产生？有什么好的解决办法？

回答：

`Iterator.remove()`是线程安全的，所以你的代码可以这样写：

```
List<String> list = new ArrayList<>();  
  
for (Iterator<String> iterator = list.iterator(); iterator.hasNext();) {  
    String string = iterator.next();  
    if (string.isEmpty()) {  
  
        // 从迭代器中移除集合元素，集合中相应的集合元素也会安全地被移除  
        // 在这里，如果继续调用的是 list.remove(string)，那么仍会抛出异常  
        iterator.remove();  
    }  
}
```

在遍历集合时修改集合的结构或内容的情况下，`Iterator.remove()`是唯一线程安全的方法。

问题原因：

fail-fast, 快速失败机制，是 java 集合类的一种错误检查机制。当有多个线程同时对集合进行遍历以及内容或者结构的修改时，就有可能产生 fail-fast 机制。这意味着，当它们发现容器在迭代的过程中被修改时，就会抛出一个 `ConcurrentModificationException` 异常。

迭代器的快速失败行为无法得到保证，它不能保证一定会出现该错误，但是快速失败操作会尽最大努力抛出 `ConcurrentModificationException` 异常，这个异常仅用于检测 bug。这种迭代器并不是完备的处理机制，而只是作为并发问题的一个预警指示器。

拓展阅读：



[fail-fast 机制的原理解析](#)

**StackOverflow** 地址:

<http://stackoverflow.com/questions/223918/iterating-through-a-collection-avoiding-concurrentmodificationexception-when-re>

## **21. 如何让 IntelliJ 编辑器永久性显示代码行数**

---

如何让 **IntelliJ** 编辑器永久性显示代码行数

问题

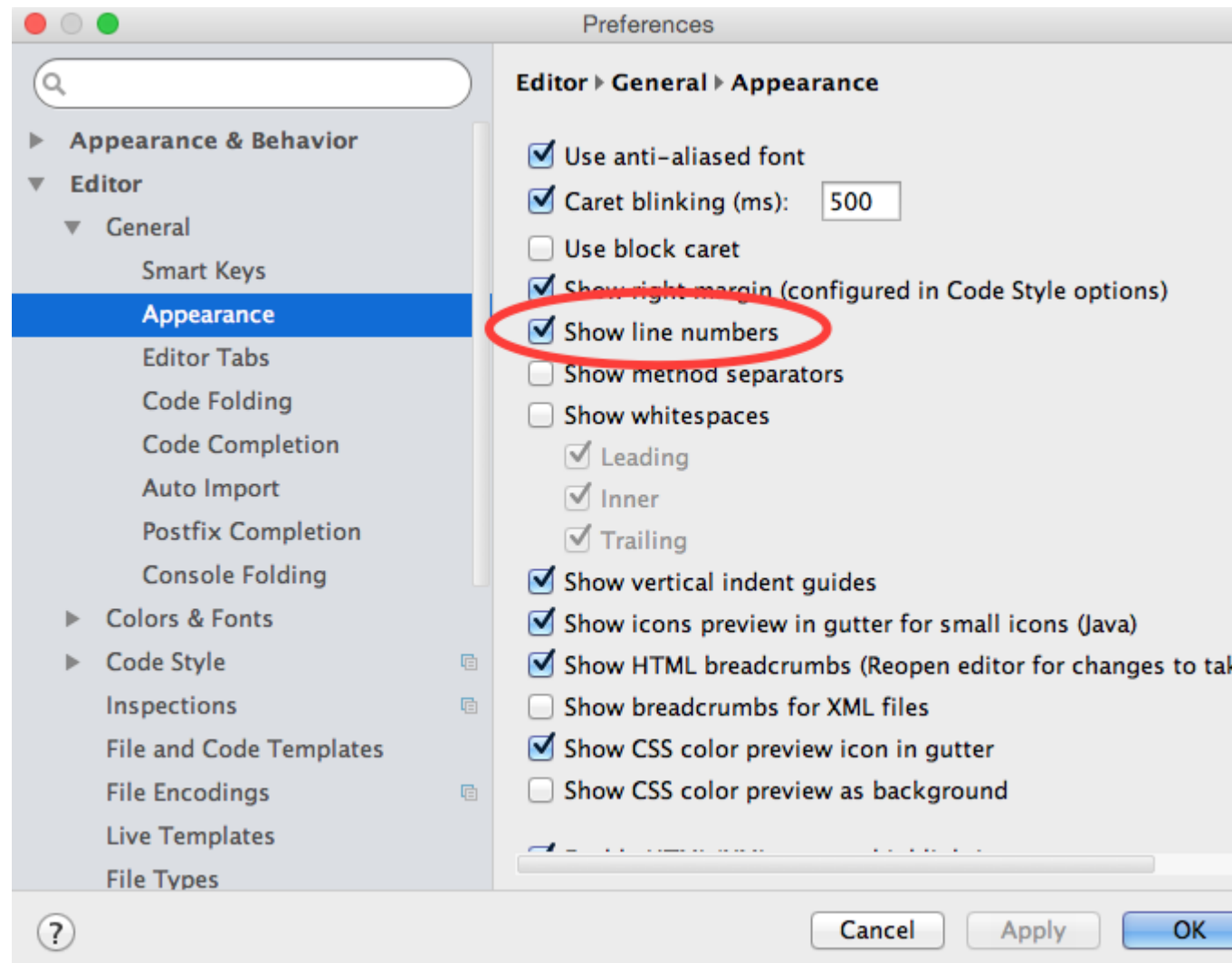
如何让 IntelliJ 编辑器永久性显示代码行数

回答

**IntelliJ 14.0** 之后的版本

打开软件的菜单 **File->Settings->Editor->General->Appearance**，在右侧的配置 **Show line numbers** 打

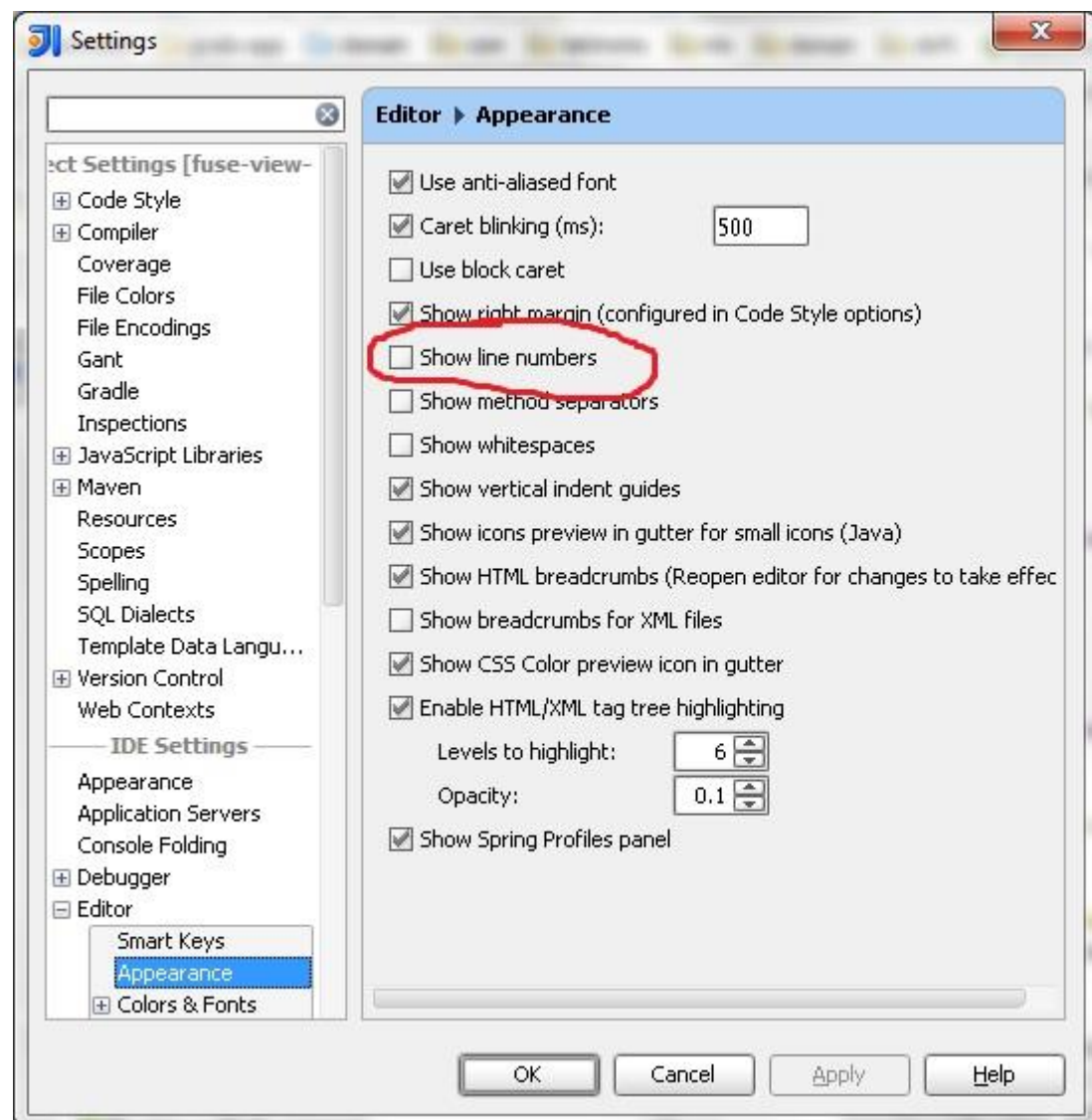
勾：



### IntelliJ 8.1.2 - 13.X 的版本

打开软件的菜单 File->Settings->Editor->Appearance，在右侧的配置 Show line numbers 打

勾：



拓展

[IntelliJ IDEA 使用教程](#)

**StackOverflow** 地址

<http://stackoverflow.com/questions/13751/how-can-i-permanently-have-line-numbers-in-intellij>

## 22. 如何使用 maven 把项目及其依赖打包为可运行 jar 包

### 如何使用 maven 把项目及其依赖打包为可运行 jar 包

#### 问题

我想把 java 项目打包为可运行的分布式 jar 包。我该怎样做，才能把项目中 maven 所依赖的 jar 包导入到我的项目 jar 包中？

#### 回答

在 pom.xml 文件中，加入如下的插件：

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <!-- 这里是你的项目 main 函数所在的类的全限定名 -->
            <mainClass>fully.qualified.MainClass</mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </plugin>
  </plugins>
</build>
```

之后，运行 maven 命令：

mvn clean compile assembly:single

clean,compile,assembly:single 任务将会依次被执行；compile 任务必须写在 assembly:single 之前，否则打包后的 jar 包内将不会有你的编译代码。

（译注：执行完后，会在你的 maven 项目的 target 目录下，生成想要的 jar 包，而不再需要使用 mvn package 命令进行打包）

通常情况下，上述 maven 命令执行后会自动绑定到项目的构建阶段，从而保证了以后在执行 mvn install 命令时的 jar 包也会被构建。（译注：下面是实际上完整的默认的 pom.xml 配置，只不过<executions>可以被省略，若省略则按照下述默认的配置执行）

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
    <archive>
      <manifest>
        <mainClass>fully.qualified.MainClass</mainClass>
      </manifest>
    </archive>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>
  <executions>
    <execution>
      <id>make-assembly</id> <!-- 用于 maven 继承项目的聚合 -->
      <phase>package</phase> <!-- 绑定到 package 阶段 -->
      <goals>
        <goal>single</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

## 拓展

怎样去运行打包后的可运行 jar 包？

- 对上述配置中已经指定了 main 函数所在类的 jar 包，打开命令行窗口，输入命令：

```
java -jar jar 包的路径/jar 包的名字.jar
```

例如：

```
java -jar D:\my_java_project\maven_test.jar
```

- 若在 pom.xml 并没有指定 main 方法所在类，那么该 jar 的运行应采取如下命令：

```
java -cp jar 包的路径/jar 包的名字.jar main 方法所在类的全限定名
```

例如:

```
java -cp D:\my_java_project\maven_test.jar com.my.path.MainClass
```

**StackOverflow** 地址

<http://stackoverflow.com/questions/574594/how-can-i-create-an-executable-jar-with-dependencies-using-maven>

## 网络

### 1. 如何使用 `java.net.URLConnection` 接收及发送 HTTP 请求

#### 如何使用 `java.net.URLConnection` 接收及发送 HTTP 请求

首先声明，下面的代码，都是基本的例子。更严谨的话，还应加入处理各种异常的代码(如 `IOExceptions`、`NullPointerException`、

`ArrayIndexOutOfBoundsException`)

#### 准备

首先，需要设置请求的 URL 以及 charset(编码)；额外的参数，则取决于各自 url 的要求。

```
String url = "http://example.com";  
String charset = "UTF-8";
```

```
String param1 = "value1";
String param2 = "value2";
// ...
String query = String.format("param1=%s&param2=%s",
    URLEncoder.encode(param1, charset),
    URLEncoder.encode(param2, charset));
```

url 中附带的请求参数，必须是 name=value 这样的格式，每个参数间用&连接。一般来说，你还得用 [URLEncoder#encode\(\)](#)对参数做[编码](#)

上面例子还用到了 String#format()。字符拼接方式，看个人喜好，我更喜欢用这个方式。

## 发送一个 [HTTP GET](#) 请求（可选：带上参数）

这依然是个繁琐的事情。默认的方式如下：

```
URLConnection connection = new URL(url + "?" + query).openConnection();
connection.setRequestProperty("Accept-Charset", charset);
InputStream response = connection.getInputStream();
```

url 和参数之间，要用？号连接。请求头（header）中的 [Accept-Charset](#)，用于告诉服务器，你所发送参数的编码。如果你不发送任何参数，也可以不管 Accept-Charset。另外如果你无需设置 header，也可以用

[URL#openStream\(\)](#) 而非 openConnection。不管那种方式，假设服务器端是 [HttpServlet](#)，那么你的 get 请求将会触发它的 doGet()方法，它能通过 [HttpServletRequest#getParameter\(\)](#)获取你传递的参数。

## 发送一个 [HTTP POST](#) 请求，并带上参数

设置 [URLConnection#setDoOutput\(\)](#)，等于隐式地将请求方法设为 POST。标准的 HTTP POST 表单，其 Content-Type 为 application/x-www-form-urlencoded，请求的内容放到 body 中。也就是如下代码：

```
URLConnection connection = new URL(url).openConnection();
connection.setDoOutput(true); // Triggers POST.
connection.setRequestProperty("Accept-Charset", charset);
connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded; charset=" + charset);

try (OutputStream output = connection.getOutputStream()) {
    output.write(query.getBytes(charset));
}

InputStream response = connection.getInputStream();
```

提醒：

当你要提交一个 HTML 表单时，务必要把,

type="submit">这类元素的值，也以 name=value 的形式提交，因为，服务端通常也需要这个信息，已确认哪一个按钮触发了这个提交动作。

也可以使用 [URLConnection](#) 来代替 [URLConnection](#)，然后调用

[URLConnection#setRequestMethod\(\)](#)来将请求设为 POST 类型。

```
URLConnection httpConnection = (URLConnection) new
URL(url).openConnection();
httpConnection.setRequestMethod("POST");
```

同样的，如果服务端是 [HttpServlet](#),将会触发它的 [doPost\(\)](#)方法,可以通过

[HttpServletRequest#getParameter\(\)](#)获取 post 参数

## 触发 HTTP 请求的发送

你可以显式地通过 [URLConnection#connect\(\)](#)来发送请求，但是，当你调用获取响应信息的方法时，一样将自动发送请求。例如当你使用

[URLConnection#getInputStream\(\)](#)时，就会自动触发请求，因此，connect()方法

往往都是多余的。上面我的例子，也都是直接调用 [getInputStream\(\)](#)方法。

获取 HTTP 响应信息



1. HTTP 响应码： 首先默认你使用了 [HttpURLConnection](#)

```
int status = httpConnection.getResponseCode();
```

2. HTTP 响应头 (headers)

```
for (Entry<String, List<String>> header :  
connection.getHeaderFields().entrySet()) {  
    System.out.println(header.getKey() + "=" + header.getValue());  
}
```

3. HTTP 响应编码： 当 Content-Type 中包含 charset 参数时，说明响应内容是基于 charset 参数指定的编码。因此，解码响应信息时，也要按照这个编码格式来。

```
String contentType = connection.getHeaderField("Content-Type");  
String charset = null;  
  
for (String param : contentType.replace(" ", "").split(";")) {  
    if (param.startsWith("charset=")) {  
        charset = param.split("=", 2)[1];  
        break;  
    }  
}  
  
if (charset != null) {  
    try (BufferedReader reader = new BufferedReader(new  
InputStreamReader(response, charset))) {  
        for (String line; (line = reader.readLine()) != null;) {  
            // ... System.out.println(line) ?  
        }  
    }  
}  
else {  
    // It's likely binary content, use InputStream/OutputStream.  
}
```

## session 的维护

服务端 session，通常是基于 cookie 实现的。你可以通过 [CookieHandlerAPI](#) 来管理 cookie。在发送 HTTP 请求前，初始化一个 [CookieManager](#)，然后设置参数为 [CookiePolicy.ACCEPT\\_ALL](#)。

```
// First set the default cookie manager.
CookieHandler.setDefault(new CookieManager(null, CookiePolicy.ACCEPT_ALL));
// All the following subsequent URLConnections will use the same cookie
manager.
URLConnection connection = new URL(url).openConnection();
// ...
connection = new URL(url).openConnection();
// ...
connection = new URL(url).openConnection();
// ...
```

请注意，这个方式并非适用于所有场景。如果使用这个方式失败了，你可以尝试自己设置 cookie：你需要从响应头中拿到 Set-Cookie 参数，然后再把 cookie 设置到接下来的其他请求中。

```
// Gather all cookies on the first request.
URLConnection connection = new URL(url).openConnection();
List<String> cookies = connection.getHeaderFields().get("Set-Cookie");
// ...

// Then use the same cookies on all subsequent requests.
connection = new URL(url).openConnection();
for (String cookie : cookies) {
    connection.addRequestProperty("Cookie", cookie.split(";", 2)[0]);
}
// ...
```

上面的 `split(";", 2)[0]` 作用是去掉一些跟服务端无关的 cookie 信息（例如 expires, path 等）。也可用 `cookie.substring(0, cookie.indexOf(';'))` 达到同样的目的

## 流的处理

不管你是否通过 `connection.setRequestProperty("Content-Length", contentLength)` 方法，为 `content` 设置了定长，[URLConnection](#) 在发送请求前，默认都会缓存整个请求的 `body`。如果发送一个比较大的 `post` 请求（例如上传文件），有可能会发生 `OutOfMemoryException`。为了避免这个问题，可以设置 [URLConnection#setFixedLengthStreamingMode\(\)](#)

```
httpConnection.setFixedLengthStreamingMode(contentLength);
```

但如果 `content` 长度是未知的，则可以用

[URLConnection#setChunkedStreamingMode\(\)](#)。这样，`header` 中

`Transfer-Encoding` 会变成 `chunked`，你的请求将会分块发送，例如下面的例子，请求的 `body`，将会按 1KB 一块，分块发送

```
httpConnection.setChunkedStreamingMode(1024);
```

## User-Agent

有时候，你发送的请求，可能只有在浏览器下才能正常返回，而其他方式却不。这可能跟请求头中的 `User-Agent` 有关。通过 `URLConnection` 发送的请求，默认会带上的 `User-Agent` 信息是 `Java/1.6.0_19`，也就是 `java+jre` 的版本。你可以重写这个信息：

```
connection.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401"); // Do as if you're using Firefox 3.6.3.
```

这里有一份更全的浏览器 [User-Agent 清单](#)

## 错误处理

如果 HTTP 的响应码是 4xx(客户端异常) 或者 5xx(服务端异常)，你可以通过 `URLConnection#getErrorStream()` 获取信息，服务端可能会将一些有用的错误信息放到这里面。

```
InputStream error = ((URLConnection) connection).getErrorStream();
```

## 上传文件

一般来说，你需要将 post 的内容设为 [multipart/form-data](#)(相关的 RFC 文档: [RFC2388](#))

```
String param = "value";
File textFile = new File("/path/to/file.txt");
File binaryFile = new File("/path/to/file.bin");
String boundary = Long.toHexString(System.currentTimeMillis()); // Just
generate some unique random value.
String CRLF = "\r\n"; // Line separator required by multipart/form-data.
URLConnection connection = new URL(url).openConnection();
connection.setDoOutput(true);
connection.setRequestProperty("Content-Type", "multipart/form-data;
boundary=" + boundary);

try (
    OutputStream output = connection.getOutputStream();
    PrintWriter writer = new PrintWriter(new OutputStreamWriter(output,
charset), true);
) {
    // Send normal param.
    writer.append("--" + boundary).append(CRLF);
    writer.append("Content-Disposition: form-data;
name=\"param\"").append(CRLF);
    writer.append("Content-Type: text/plain; charset=" +
charset).append(CRLF);
    writer.append(CRLF).append(param).append(CRLF).flush();

    // Send text file.
    writer.append("--" + boundary).append(CRLF);
    writer.append("Content-Disposition: form-data; name=\"textFile\";
filename=\"" + textFile.getName() + "\"").append(CRLF);
```

```

        writer.append("Content-Type: text/plain; charset=" +
charset).append(CRLF); // Text file itself must be saved in this charset!
        writer.append(CRLF).flush();
        Files.copy(textFile.toPath(), output);
        output.flush(); // Important before continuing with writer!
        writer.append(CRLF).flush(); // CRLF is important! It indicates end of
boundary.

        // Send binary file.
        writer.append("--" + boundary).append(CRLF);
        writer.append("Content-Disposition: form-data; name=\"binaryFile\";
filename=\"" + binaryFile.getName() + "\"").append(CRLF);
        writer.append("Content-Type: " +
URLConnection.guessContentTypeFromName(binaryFile.getName()).append(CRLF);
        writer.append("Content-Transfer-Encoding: binary").append(CRLF);
        writer.append(CRLF).flush();
        Files.copy(binaryFile.toPath(), output);
        output.flush(); // Important before continuing with writer!
        writer.append(CRLF).flush(); // CRLF is important! It indicates end of
boundary.

        // End of multipart/form-data.
        writer.append("--" + boundary + "--").append(CRLF).flush();
    }

```

假设服务端还是一个 [HttpServlet](#), 它的 doPost() 方法将会处理这个请求, 服务端通过 [HttpServletRequest#getPart\(\)](#) 获取你发送的内容（注意了, 不是 getParameter()）。getPart() 是个比较新的方法, 是在 Servlet 3.0 后才引入的。如果你是 Servlet 3.0 之前的版本, 则可以选用 [Apache Commons FileUpload](<http://commons.apache.org/fileupload>) 来解析 multipart/form-data 的请求。可以参考这里的[例子](<http://stackoverflow.com/questions/2422468/upload-big-file-to-servlet/2424824#2424824>)

## 最后的话

上面啰嗦了很多，Apache 提供了工具包，帮助我们更方便地完成这些事情 [Apache HttpComponents HttpClient](#):

- [HttpClient Tutorial](#)
- [HttpClient Examples](#)

google 也有类似的[工具包](#)

解析、提取 HTML 内容 如果你想解析提取 html 的内容，你可以用 [Jsoup](#) 等解析器

- [一些比较有名的 HTML 解析器的优缺点](#)
- [用 java 如何扫描和解析网页](#)

stackoverflow 原址: <http://stackoverflow.com/questions/2793150/using-java-net-urlconnection-to-fire-and-handle-http-requests>

## 性能

### 1. LinkedList、ArrayList 各自的使用场景，如何确认应该用哪一个呢？

---

**LinkedList、ArrayList** 各自的使用场景，如何确认应该用哪一个呢？

一言以蔽之，在大部分情况下，使用 ArrayList 会好一些。

耗时上各有优缺点。**ArrayList** 稍有优势

List 只是一个接口，而 LinkedList、ArrayList 是 List 的不同实现。LinkedList 的模型是双向链表，而 ArrayList 则是动态数组

首先对比下常用操作的算法复杂度

### LinkedList

- `get(int index)` :  $O(n)$
- `add(E element)` :  $O(1)$
- `add(int index, E element)` :  $O(n)$
- `remove(int index)` :  $O(n)$
- `Iterator.remove()` :  $O(1)$  <--- LinkedList 的主要优点
- `ListIterator.add(E element)` is  $O(1)$  <--- LinkedList 的主要优点

### ArrayList

- `get(int index)` :  $O(1)$  <--- ArrayList 的主要优点
- `add(E element)` : 基本是  $O(1)$ ，因为动态扩容的关系，最差时是  $O(n)$
- `add(int index, E element)` : 基本是  $O(n - index)$ ，因为动态扩容的关系，最差时是  $O(n)$
- `remove(int index)` :  $O(n - index)$  (例如，移除最后一个元素，是  $O(1)$ )
- `Iterator.remove()` :  $O(n - index)$
- `ListIterator.add(E element)` :  $O(n - index)$

- **LinkedList**，因为本质是个链表，所以通过 **Iterator** 来插入和移除操作的耗时，都是个恒量，但如果要获取某个位置的元素，则要做指针遍历。因此，**get** 操作的耗时会跟 **List** 长度有关

对于 **ArrayList** 来说，得益于快速随机访问的特性，获取任意位置元素的耗时，是常量的。但是，如果是 **add** 或者 **remove** 操作，要分两种情况，如果是在尾部做 **add**，也就是执行 **add** 方法（没有 **index** 参数），此时不需要移动其他元素，耗时是  $O(1)$ ，但如果不是在尾部做 **add**，也就是执行 **add(int index, E element)**，这时候在插入新元素的同时，也要移动该位置后面的所有元素，以为新元素腾出位置，此时耗时是  $O(n - \text{index})$ 。另外，当 **List** 长度超过初始化容量时，会自动生成一个新的 **array**（长度是之前的 1.5 倍），此时会将旧的 **array** 移动到新的 **array** 上，这种情况下的耗时是  $O(n)$ 。

总之，**get** 操作，**ArrayList** 快一些。而 **add** 操作，两者差不多。（除非是你希望在 **List** 中间插入节点，且维护了一个 **Iterator** 指向指定位置，这时候 **LinkedList** 能快一些，但是，我们更多时候是直接在尾部插入节点，这种特例的情况并不多）

## 空间占用上，**ArrayList** 完胜

看下两者的内存占用图

这三个图，横轴是 **list** 长度，纵轴是内存占用值。两条蓝线是 **LinkedList**，两条红线是 **ArrayList**

可以看到，**LinkedList** 的空间占用，要远超 **ArrayList**。**LinkedList** 的线更陡，随着 **List** 长度的扩大，所占用的空间要比同长度的 **ArrayList** 大得多。注：从



mid JDK6 之后，默认启用了 CompressedOops ，因此 64 位及 32 位下的结果没有差异，LinkedList x64 和 LinkedList x32 的线是一样的。

stackoverflow 原址: <http://stackoverflow.com/questions/322715/when-to-use-linkedlist-over-arraylist>

## 2. StringBuilder 和 StringBuffer 有哪些区别呢

---

### StringBuilder 和 StringBuffer 有哪些区别呢

最主要的区别，**StringBuffer** 的实现用了 **synchronized**（锁），而 **StringBuilder** 没有。

因此，StringBuilder 会比 StringBuffer 快。

如果你

- 非常非常追求性能（其实这两个都不慢，比直接操作 String，要快非常多了）
- 不需要考虑线程安全问题,
- JRE 是 1.5+

可以用 StringBuilder,反之，请用 StringBuffer。

性能测试例子:

如下这个例子，使用 StringBuffer，耗时 2241ms,而 StringBuilder 是 753ms

```
public class Main {  
    public static void main(String[] args) {
```

```

int N = 77777777;
long t;

{
    StringBuffer sb = new StringBuffer();
    t = System.currentTimeMillis();
    for (int i = N; i --> 0 ; ) {
        sb.append("");
    }
    System.out.println(System.currentTimeMillis() - t);
}

{
    StringBuilder sb = new StringBuilder();
    t = System.currentTimeMillis();
    for (int i = N; i --> 0 ; ) {
        sb.append("");
    }
    System.out.println(System.currentTimeMillis() - t);
}
}

```

stackoverflow 讨论原

址 <http://stackoverflow.com/questions/355089/stringbuilder-and-stringbuffer>

### 3. 为什么处理排序的数组要比非排序的快

## 为什么处理排序的数组要比非排序的快

### 问题

以下是\*\*c++\*\*的一段非常神奇的代码。由于一些奇怪原因，对数据排序后奇迹般的让这段代码快了近 6 倍！！

```

#include <algorithm>
#include <ctime>
#include <iostream>

```

```

int main()
{
    // Generate data
    const unsigned arraySize = 32768;
    int data[arraySize];

    for (unsigned c = 0; c < arraySize; ++c)
        data[c] = std::rand() % 256;

    // !!! With this, the next loop runs faster
    std::sort(data, data + arraySize);

    // Test
    clock_t start = clock();
    long long sum = 0;

    for (unsigned i = 0; i < 100000; ++i)
    {
        // Primary loop
        for (unsigned c = 0; c < arraySize; ++c)
        {
            if (data[c] >= 128)
                sum += data[c];
        }
    }

    double elapsedTime = static_cast<double>(clock() - start) /
CLOCKS_PER_SEC;

    std::cout << elapsedTime << std::endl;
    std::cout << "sum = " << sum << std::endl;
}

```

- 没有 `std::sort(data, data + arraySize);`,这段代码运行了 11.54 秒.
- 有这个排序的代码, 则运行了 1.93 秒. 我原以为这也许只是语言或者编译器的不一样的问题, 所以我又用 Java 试了一下。

以下是 Java 代码段

```
import java.util.Arrays;
```

```

import java.util.Random;

public class Main
{
    public static void main(String[] args)
    {
        // Generate data
        int arraySize = 32768;
        int data[] = new int[arraySize];

        Random rnd = new Random(0);
        for (int c = 0; c < arraySize; ++c)
            data[c] = rnd.nextInt() % 256;

        // !!! With this, the next loop runs faster
        Arrays.sort(data);

        // Test
        long start = System.nanoTime();
        long sum = 0;

        for (int i = 0; i < 100000; ++i)
        {
            // Primary loop
            for (int c = 0; c < arraySize; ++c)
            {
                if (data[c] >= 128)
                    sum += data[c];
            }
        }

        System.out.println((System.nanoTime() - start) / 1000000000.0);
        System.out.println("sum = " + sum);
    }
}

```

结果相似，没有很大的差别。

---

我首先得想法是排序把数据放到了 cache 中，但是我下一个想法是我之前的想法是多么傻啊，因为这个数组刚刚被构造。

- 到底这是为什么呢？

- 为什么排序的数组会快于没有排序的数组？
- 这段代码是为了求一些无关联的数据的和，排不排序应该没有关系啊。

## 回答

什么是分支预测？

看看这个铁路岔

口



Image by Mecanismo, via Wikimedia Commons. Used under the CC-BY-SA 3.0 license.

为了理解这个问题，想象一下，如果我们回到 19 世纪.

你是在分岔口的操作员。当你听到列车来了，你没办法知道这两条路哪一条是正确的。然后呢，你让列车停下来，问列车员哪条路是对的，然后你才转换铁路方向。

火车很重有很大的惯性。所以他们得花费很长的时间开车和减速。

是不是有个更好的办法呢？你猜测哪个是火车正确的行驶方向

- 如果你猜对了，火车继续前行
- 如果你猜错了，火车得停下来，返回去，然后你再换条路。

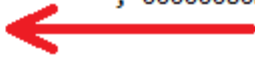
如果你每次都猜对了，那么火车永远不会停下来。如果你猜错太多次，那么火车会花费很多时间来停车，返回，然后再启动

---

考虑一个 **if** 条件语句：在处理器层面上，这是一个分支指令：

```
if (data[c] >= 128)
    sum += data[c];
```

```
2876      movsxd  rdx, DWORD PTR [rcx]
2877      cmp     edx, 128                ; 00000080H
2878      jl      SHORT $LN3@main
2879      add     rbx, rdx
2880  $LN3@main:
```



当处理器看到这个分支时，没办法知道哪个将是下一条指令。该怎么办呢？貌似只能暂停执行，直到前面的指令完成，然后再继续执行正确的下一条指令？现代处理器很复杂，因此它需要很长的时间"热身"、"冷却"

是不是有个更好的办法呢？你猜测下一个指令在哪！

- 如果你猜对了，你继续执行。
- 如果你猜错了，你需要 flush the pipeline，返回到那个出错的分支，然后你才能继续。

如果你每次都猜对了，那么你永远不会停 如果你猜错了太多次，你就要花很多时间来滚回，重启。

---

这就是分支预测。我承认这不是一个好的类比，因为火车可以用旗帜来作为方向的标识。但是在电脑中，处理器不能知道哪一个分支将走到最后。

所以怎样能很好的预测，尽可能地使火车必须返回的次数变小？你看看火车之前的选择过程，如果这个火车往左的概率是 99%。那么你猜左，反之亦然。如果每 3 次会有 1 次走这条路，那么你也按这个三分之一的规律进行。

**换句话说，你试着定下一个模式，然后按照这个模式去执行。**这就差不多是分支预测是怎么工作的。

大多数的应用都有很好的分支预测。所以现代的分支预测器通常能实现大于 90% 的命中率。但是当面对没有模式识别、无法预测的分支，那分支预测基本就没用了。

如果你想知道更多:[Branch predictor" article on Wikipedia.](#)

---

有了前面的说明，问题的来源就是这个 **if** 条件判断语句

```
if (data[c] >= 128)
    sum += data[c];
```

注意到数据是分布在 0 到 255 之间的。当数据排好序后，基本上前一半大的数据不会进入这个条件语句，而后一半的数据，会进入该条件语句。

连续的进入同一个执行分支很多次，这对分支预测是非常友好的。可以更准确地预测，从而带来更高的执行效率。

**快速理解一下**

```
T = branch taken
N = branch not taken

data[] = 0, 1, 2, 3, 4, ... 126, 127, 128, 129, 130, ... 250, 251, 252, ...
branch = N N N N N ... N N T T T ... T T T ...

        = NNNNNNNNNNNN ... NNNNNNTTTTTTTTTT ... TTTTTTTTTT (easy to predict)
```



但是当数据是完全随机的，分支预测就没什么用了。因为他无法预测随机的数据。因此就会有大概 50%的概率预测出错。

```
data[] = 226, 185, 125, 158, 198, 144, 217, 79, 202, 118, 14, 150, 177,
182, 133, ...
branch =  T,  T,  N,  T,  T,  T,  T,  N,  T,  N,  N,  T,  T,  T,
N  ...

          = TTNTTTTNTNNTTTN ...  (completely random - hard to predict)
```

---

我们能做些什么呢

如果编译器无法优化带条件的分支，如果你愿意牺牲代码的可读性换来更好的性能的话，你可以用下面的一些技巧。

把

```
if (data[c] >= 128)
    sum += data[c];
```

替换成

```
int t = (data[c] - 128) >> 31;
sum += ~t & data[c];
```

这消灭了分支，把它替换成按位操作。

（说明：这个技巧不是非常严格的等同于原来的 if 条件语句。但是在 data[] 当前这些值下是 OK 的）

使用的设备参数是：**Core i7 920 @ 3.5 GHz** C++ - Visual Studio 2010 - x64

Release

```
// Branch - Random
seconds = 11.777

// Branch - Sorted
seconds = 2.352

// Branchless - Random
seconds = 2.564
```



```
// Branchless - Sorted  
seconds = 2.587
```

Java - Netbeans 7.1.1 JDK 7 - x64

```
// Branch - Random  
seconds = 10.93293813  
  
// Branch - Sorted  
seconds = 5.643797077  
  
// Branchless - Random  
seconds = 3.113581453  
  
// Branchless - Sorted  
seconds = 3.186068823
```

结论：

- 用了分支(if)：没有排序和排序的数据，效率有很大的区别
- 用了上面提到的按位操作替换：排序与否，效率没有很大的区别
- 在使用 C++ 的情况下，按位操作还是要比排好序的分支操作要慢。

一般的建议是尽量避免在关键循环上出现对数据很依赖的分支。（就像这个例子）

---

更新：

- GCC 4.6.1 用了 -O3 or -ftree-vectorize，在 64 位机器上，数据有没有排序，都是一样快。 ... ..等各种例子

说明了现代编译器越发成熟强大，可以在这方面充分优化代码的执行效率

## 相关内容

CPU 的流水线指令执行

想象现在有一堆指令等待 CPU 去执行，那么 CPU 是如何执行的呢？具体的细节可以找一本计算机组成原理来看。CPU 执行一堆指令时，并不是单纯地一条一条取出来执行，而是按照一种流水线的方式，在 CPU 真正指令前，这条指令就像工厂里流水线生产的产品一样，已经被经过一些处理。简单来说，一条指令可能经过过程：取指(Fetch)、解码(Decode)、执行(Execute)、放回(Write-back)。

假设现在有指令序列 ABCDEFG。当 CPU 正在执行(execute)指令 A 时，CPU 的其他处理单元（CPU 是由若干部件构成的）其实已经预先处理到了指令 A 后面的指令，例如 B 可能已经被解码，C 已经被取指。这就是流水线执行，这可以保证 CPU 高效地执行指令。

### 分支预测

如上所说，CPU 在执行一堆顺序执行的指令时，因为对于执行指令的部件来说，其基本不需要等待，因为诸如取指、解码这些过程早就被做了。但是，当 CPU 面临非顺序执行的指令序列时，例如之前提到的跳转指令，情况会怎样呢？

取指、解码这些 CPU 单元并不知道程序流程会跳转，只有当 CPU 执行到跳转指令本身时，才知道该不该跳转。所以，取指解码这些单元就会继续取跳转指令之后的指令。当 CPU 执行到跳转指令时，如果真的发生了跳转，那么之前的预处理（取指、解码）就白做了。这个时候，CPU 得从跳转目标处临时取指、解码，然后才开始执行，这意味着：CPU 停了若干个时钟周期！

这其实是个问题，如果 CPU 的设计放任这个问题，那么其速度就很难提升起来。为此，人们发明了一种技术，称为 branch prediction，也就是分支预测。

分支预测的作用，就是预测某个跳转指令是否会跳转。而 CPU 就根据自己的预

测到目标地址取指令。这样，即可从一定程度提高运行速度。当然，分支预测在实现上有很多方法。

**stackoverflow 链接:**

这个问题的所有回答中，最高的回答，获取了上万个 vote，还有很多个回答，非常疯狂，大家觉得不过瘾可以移步到这里查看

<http://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-an-unsorted-array>

## 4. 如何使用 Java 创建一个内存泄漏的程序

---

如何使用 **Java** 创建一个内存泄漏的程序

问题:

我在一个面试的过程中被问到如何使用 **Java** 创建一个内存泄漏的程序。毫无疑问地说，我当时哑口无言，根本不知道如何开始编码。

解答

在 **Java** 下有一个很好的方法来创建内存泄漏程序--通过使得对象不可访问但任然存储在内存中。

1. 应用程序创建一个长期运行的线程 **A** 或者 使用一个线程池来加快泄漏的速度。
2. 线程 **A** 使用 **ClassLoader**（用户可以自定义）加载一个类 **B**。
3. 在类 **B** 申请一块很大的连续内存（例如：`new byte[1000000]`）， 并使用一个静态成员变量保存该空间的一个强引用，之后在一个 **ThreadLocal**

对象中存储类 B 对象的引用。虽然泄漏这个类的一个实例就足够了，但是也可以通过申请多个实例的方法来加快内存泄漏的速度。

4. 线程 A 清理所有指向自定义类或者通过 ClassLoader 加载的引用。

5. 重复上述步骤

上述方式可以达到内存泄漏的目的，因为 ThreadLocal 存储了一个指向类 B 对象的引用，而该对象又保存了一个指向其类的引用，这个类又保存了一个指向其 ClassLoader 的引用，而 ClassLoader 又保存了一个通过它加载的所有类的引用。这种方法在许多的 JVM 的实现中表现更糟糕，因为 Classes 和 ClassLoader 被直接存储在老年代（permgen）并且永远都不会被 GC 处理。

下方为个人理解\*\*\*\*\*

通过一个简单的图来描述上述关系：

ThreadLocal.obj ---> B.obj ---> B.class <--> ClassLoader.obj

注：上图的\*.obj 表示\*类的一个实例对象，B.class 表示类 B 的 Class 对象

上方为个人理解\*\*\*\*\*

这个模式的一个变形：如果频繁的重新部署那些可能使用 ThreadLocals 的应用，应用容器（例如 Tomcat）就会像筛子一样泄漏内存。因为应用容器使用上述所说的线程，每次重新部署应用时，应用容器都会使用一个新的 ClassLoader。

具体代码可以参考：<https://gist.github.com/dpryden/b2bb29ee2d146901b4ae>

参考：<http://frankkieviet.blogspot.com/2006/10/classloader-leaks-dreaded-permgen-space.html>

stackoverflow 原址: <http://stackoverflow.com/questions/6470651/creating-a-memory-leak-with-java>

## 5. 为什么打印“B”会明显的比打印“#”慢

---

为什么打印“B”会明显的比打印“#”慢

### 问题

我生成了两个 1000x1000 的矩阵:

第一个矩阵: 0 和#。

第二个矩阵: 0 和 B。

使用如下的代码, 生成第一个矩阵需要 8.52 秒:

```
Random r = new Random();
for (int i = 0; i < 1000; i++) {
    for (int j = 0; j < 1000; j++) {
        if(r.nextInt(4) == 0) {
            System.out.print("0");
        } else {
            System.out.print("#");
        }
    }
}

System.out.println("");
}
```

而使用这段代码, 生成第二个矩阵花费了 259.152 秒:

```
Random r = new Random();
for (int i = 0; i < 1000; i++) {
    for (int j = 0; j < 1000; j++) {
        if(r.nextInt(4) == 0) {
            System.out.print("0");
        } else {
            System.out.print("B"); //only line changed
        }
    }
}
```

```
}  
  
    System.out.println("");  
}
```

如此大的运行时间差异的背后究竟是什么原因呢？

---

正如评论中所建议的，只打印 `System.out.print("#");` 用时 7.8871 秒，而 `System.out.print("B");` 则给出 `still printing...`。

另外有人指出这段代码对他们来说是正常的，我使用了 [Ideone.com](http://Ideone.com)，这两段代码的执行速度是相同的。

测试条件：

- 我在 Netbeans 7.2 中运行测试，由控制台显示输出
- 我使用了 `System.nanoTime()` 来计算时间

## 解答一

*纯粹的推测*是因为你使用的终端尝试使用[单词换行](#)而不是字符换行，并且它认为 `B` 是一个单词而 `#` 却不是。所以当它到达行尾并且寻找一个换行的地方的时候，如果是 `#` 就可以马上换行；而如果是 `B`，它则需要花更长时间搜索，因为可能会有更多的内容才能换行（在某些终端会非常费时，比如说它会先输出退格再输出空格去覆盖被换行的那部分字符）。但这都只是纯粹的推测。

## 解答二

我用 Eclipse 和 Netbeans 8.0.2 做了测试，他们的 Java 版本都是 1.8；我用了 `System.nanoTime()` 来计时。

## Eclipse:

我得到了用时相同的结果 - 大约 **1.564** 秒。

## Netbeans:

- 使用"#": **1.536 秒**
- 使用"B": **44.164 秒**

所以看起来像是 Netbeans 输出到控制台的性能问题。

在做了更多研究以后我发现问题所在是 Netbeans [换行](#) 的最大缓存（这并不限于 System.out.println 命令），参见以下代码：

```
for (int i = 0; i < 1000; i++) {  
    long t1 = System.nanoTime();  
    System.out.print("BBB.....BBB"); \\<-contain 1000 "B"  
    long t2 = System.nanoTime();  
    System.out.println(t2-t1);  
    System.out.println("");  
}
```

每一个循环所花费的时间都不到 1 毫秒，除了 **每第五个循环**会花掉大约 225 毫秒。像这样（单位是毫秒）：

```
BBB...31744  
BBB...31744  
BBB...31744  
BBB...31744  
BBB...226365807  
BBB...31744  
BBB...31744  
BBB...31744  
BBB...31744  
BBB...226365807  
.  
.  
.
```

以此类推。

## 总结：

1. 使用 Eclipse 打印"B"完全没有问题
2. Netbeans 有换行的问题但是可以被解决（因为在 Eclipse 并没有这个问题）（而不用在 B 后面添加空格（"B "））。

stackoverflow 原址: <http://stackoverflow.com/questions/21947452/why-is-printing-b-dramatically-slower-than-printing>

## 测试

### 1. 如何测试 **private** 方法, 变量或者内部类

#### 如何测试 **private** 方法, 变量或者内部类

当你需要测试一个遗留的应用程序, 且不能更改方法的可见性时, 那么, 测试私有方法/属性的最好方式就是使用[反射](#)。

实际测试时, 可以通过一些反射辅助类, 设置和获取私有(静态)的变量、调用私有(静态)方法。遵循下面的窍门, 你可以很好地处理私有方法和变量的测试。

```
Method method = targetClass.getDeclaredMethod(methodName, argClasses);
method.setAccessible(true);
return method.invoke(targetObject, argObjects);
```

私有变量:

```
Field field = targetClass.getDeclaredField(fieldName);
field.setAccessible(true);
field.set(object, value);
```

note:



1. `targetClass.getDeclaredMethod(methodName, argClasses)`这个方法能让你获取到私有方法。`getDeclaredField` 让你获取到私有变量
2. 在对私有变量（方法）进行处理前，需要先 `setAccessible(true)`

stackoverflow 原址: <http://stackoverflow.com/questions/34571/how-to-test-a-class-that-has-private-methods-fields-or-inner-classes>

## 2. JUnit4 如何断言确定异常的抛出

### JUnit4 如何断言确定异常的抛出

#### 问题

在 JUnit4 单元测试中，我要怎样做才能测试出有特定的异常抛出？我能想到的就只有下面的方法：

```
@Test
public void testFooThrowsIndexOutOfBoundsException() {
    boolean thrown = false;

    try {
        foo.doStuff();
    } catch (IndexOutOfBoundsException e) {
        thrown = true;
    }

    assertTrue(thrown);
}
```

#### 回答 1

在 JUnit4 后支持下面的写法：

```
@Test(expected=IndexOutOfBoundsException.class)
public void testIndexOutOfBoundsException() {
```

```
ArrayList emptyList = new ArrayList();
Object o = emptyList.get(0);
}
```

（译者：在@Test注解内提供了 expected 属性，你可以用它来指定一个 Throwable 类型，如果方法调用中抛出了这个异常，那么这条测试用例就相当于通过了）

## 回答 2

如果你使用的是 JUnit4.7，你可以使用如下的期望异常规则来验证异常信息：

```
public class FooTest {
    @Rule
    public final ExpectedException exception = ExpectedException.none();

    @Test
    public void doStuffThrowsIndexOutOfBoundsException() {
        Foo foo = new Foo();

        exception.expect(IndexOutOfBoundsException.class);
        foo.doStuff();
    }
}
```

这种方式比@Test(expected=IndexOutOfBoundsException.class)要更好，如果是在调用 foo.doStuff()方法之前就已经抛出异常的话，测试结果就不是我们想要的了。（译者：同时，ExpectedException 还能够验证异常信息，如 exception.expectMessage("there is an exception!");

## 拓展阅读

1. [JUnit: 使用 ExpectedException 进行异常测试](#)
2. [JUnit4 用法详解](#)

**StackOverflow** 地址：

<http://stackoverflow.com/questions/156503/how-do-you-assert-that-a-certain-exception-is-thrown-in-junit-4-tests>

# Android

## 1. 在 Android 里面下载文件，并在 ProgressDialog 显示进度

---

### 在 Android 里面下载文件，并在 ProgressDialog 显示进度

#### 问题

尝试写一个可以进行“应用更新”的 APP。为了达到这个效果，我写了一个可以下载文件并且在一个 ProgressDialog 里面显示进度的简单方法。我知道如何使用 ProgressDialog，但是我不太确定怎么显示当前进度和下载文件。

#### 回答

有很多方式去下载文件。我给出一些最常用的方法；由你来选择选择哪一个最适合你的应用。

#### 1. 使用 AsyncTask，并且在一个 dialog 里面显示进度

这种方法允许你执行一些后台任务，并且同时更新 UI（在这里，我们是更新进度条 progress bar）。

首先是实例代码

```
// 定义一个 dialog 为 Activity 的成员变量
ProgressDialog mProgressDialog;

// 在 onCreate() 方法里面初始化
mProgressDialog = new ProgressDialog(YourActivity.this);
mProgressDialog.setMessage("A message");
mProgressDialog.setIndeterminate(true);
mProgressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
```

```

mProgressDialog.setCancelable(true);

// 执行下载器
final DownloadTask downloadTask = new DownloadTask(YourActivity.this);
downloadTask.execute("你要下载文件的 Url");

mProgressDialog.setOnCancelListener(new DialogInterface.OnCancelListener()
{
    @Override
    public void onCancel(DialogInterface dialog) {
        downloadTask.cancel(true);
    }
});

```

AsyncTask 看起来像这样

```

// 一般我们把 AsyncTask 的子类定义在 Activity 的内部
// 通过这种方式，我们就可以轻松地在这里更改 UI 线程
private class DownloadTask extends AsyncTask<String, Integer, String> {

    private Context context;
    private PowerManager.WakeLock mWakeLock;

    public DownloadTask(Context context) {
        this.context = context;
    }

    @Override
    protected String doInBackground(String... sUrl) {
        InputStream input = null;
        OutputStream output = null;
        HttpURLConnection connection = null;
        try {
            URL url = new URL(sUrl[0]);
            connection = (HttpURLConnection) url.openConnection();
            connection.connect();

            // 避免因为接收到非 HTTP 200 OK 状态，而导致只或者错误代码，而不是要下
            // 载的文件
            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                return "Server returned HTTP " +
connection.getResponseCode()
                + " " + connection.getResponseMessage();
            }

            // 这对显示下载百分比有帮助

```

```

// 当服务器没有返回文件的大小时，数字可能为-1
int fileLength = connection.getContentLength();

// 下载文件
input = connection.getInputStream();
output = new FileOutputStream("/sdcard/file_name.extension");

byte data[] = new byte[4096];
long total = 0;
int count;
while ((count = input.read(data)) != -1) {
    // 允许用返回键取消下载
    if (isCancelled()) {
        input.close();
        return null;
    }
    total += count;
    // 更新下载进度
    if (fileLength > 0) // 只有当 fileLength>0 的时候才会调用
        publishProgress((int) (total * 100 / fileLength));
    output.write(data, 0, count);
}
} catch (Exception e) {
    return e.toString();
} finally {
    try {
        if (output != null)
            output.close();
        if (input != null)
            input.close();
    } catch (IOException ignored) {
    }

    if (connection != null)
        connection.disconnect();
}
return null;
}

```

上面的 `doInBackground` 方法总是在后台线程中运行。你不能在这里做任何 UI 线程相关的任务。另一方面，`onProgressUpdate` 和 `onPostExecute` 是在 UI 线程里面运行的，所以你可以在这里更改进度条。

`@Override`

```

protected void onPreExecute() {
    super.onPreExecute();
    // 取得 CPU 锁，避免因为用户在下载过程中按了电源键而导致的失效
    PowerManager pm = (PowerManager)
context.getSystemService(Context.POWER_SERVICE);
    mWakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
        getClass().getName());
    mWakeLock.acquire();
    mProgressDialog.show();
}

@Override
protected void onProgressUpdate(Integer... progress) {
    super.onProgressUpdate(progress);
    // 如果到了这里，文件长度是确定的，设置 indeterminate 为 false
    mProgressDialog.setIndeterminate(false);
    mProgressDialog.setMax(100);
    mProgressDialog.setProgress(progress[0]);
}

@Override
protected void onPostExecute(String result) {
    mWakeLock.release();
    mProgressDialog.dismiss();
    if (result != null)
        Toast.makeText(context, "Download error: "+result,
Toast.LENGTH_LONG).show();
    else
        Toast.makeText(context, "File downloaded",
Toast.LENGTH_SHORT).show();
}

```

为了可正常运行，你还要取得 WAKE\_LOCK 权限

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

## 2. 从服务器上下载文件

这里有个最大的问题：*我怎么从 service 来更新我的 activity?*。在下一个例子当中我们会使用两个你可能不熟悉的类：ResultReceiver 和 IntentService。

ResultReceiver 是一个可以允许我们用 Service 来更新线程的类；IntentService 是一个可以生成用来处理后台任务的线程的 Service 子类（你需要知道，

Service 实际上是和你的应用运行在同一个线程的；当你继承了 Service 之后，你必须手动生成一个新的线程来处理费时操作）。  
一个提供下载功能的 Service 看起来像这样：

```
public class DownloadService extends IntentService {
    public static final int UPDATE_PROGRESS = 8344;
    public DownloadService() {
        super("DownloadService");
    }
    @Override
    protected void onHandleIntent(Intent intent) {
        String urlToDownload = intent.getStringExtra("url");
        ResultReceiver receiver = (ResultReceiver)
intent.getParcelableExtra("receiver");
        try {
            URL url = new URL(urlToDownload);
            URLConnection connection = url.openConnection();
            connection.connect();
            // 这对你在进度条上面显示百分比很有用
            int fileLength = connection.getContentLength();

            // download the file
            InputStream input = new
BufferedInputStream(connection.getInputStream());
            OutputStream output = new
FileOutputStream("/sdcard/BarcodeScanner-debug.apk");

            byte data[] = new byte[1024];
            long total = 0;
            int count;
            while ((count = input.read(data)) != -1) {
                total += count;
                // 更新进度条....
                Bundle resultData = new Bundle();
                resultData.putInt("progress" ,(int) (total * 100 /
fileLength));
                receiver.send(UPDATE_PROGRESS, resultData);
                output.write(data, 0, count);
            }

            output.flush();
            output.close();
            input.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

    }

    Bundle resultData = new Bundle();
    resultData.putInt("progress", 100);
    receiver.send(UPDATE_PROGRESS, resultData);
}
}

```

把这个 Service 添加到清单文件中:

```
<service android:name=".DownloadService"/>
```

activity 里面的代码:

```
// 像第一个例子里面一样初始化 ProgressBar
```

```
// 在这里启动下载
```

```

mProgressDialog.show();
Intent intent = new Intent(this, DownloadService.class);
intent.putExtra("url", "url of the file to download");
intent.putExtra("receiver", new DownloadReceiver(new Handler()));
startService(intent);

```

然后像这样来使用 ResultReceiver:

```

private class DownloadReceiver extends ResultReceiver{
    public DownloadReceiver(Handler handler) {
        super(handler);
    }

    @Override
    protected void onReceiveResult(int resultCode, Bundle resultData) {
        super.onReceiveResult(resultCode, resultData);
        if (resultCode == DownloadService.UPDATE_PROGRESS) {
            int progress = resultData.getInt("progress");
            mProgressDialog.setProgress(progress);
            if (progress == 100) {
                mProgressDialog.dismiss();
            }
        }
    }
}

```

## 2.1 使用 Groundy 库



[Groundy](#) 是一个可以帮助你后台服务中运行代码片段的库，它是基于 `ResultReceiver` 这一概念。但是这个库现在已经被标记为过时了（deprecated）。下面是完整代码的样子。

你要展示 dialog 的 Activity:

```
public class MainActivity extends Activity {

    private ProgressDialog mProgressDialog;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        findViewById(R.id.btn_download).setOnClickListener(new
View.OnClickListener() {
            public void onClick(View view) {
                String url = ((EditText)
findViewById(R.id.edit_url)).getText().toString().trim();
                Bundle extras = new Bundler().add(DownloadTask.PARAM_URL,
url).build();
                Groundy.create(DownloadExample.this, DownloadTask.class)
                    .receiver(mReceiver)
                    .params(extras)
                    .queue();

                mProgressDialog = new ProgressDialog(MainActivity.this);
mProgressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
                mProgressDialog.setCancelable(false);
                mProgressDialog.show();
            }
        });
    }

    private ResultReceiver mReceiver = new ResultReceiver(new Handler()) {
        @Override
        protected void onReceiveResult(int resultCode, Bundle resultData) {
            super.onReceiveResult(resultCode, resultData);
            switch (resultCode) {
                case Groundy.STATUS_PROGRESS:
```

```

mProgressDialog.setProgress(resultData.getInt(Groundy.KEY_PROGRESS));
        break;
        case Groundy.STATUS_FINISHED:
            Toast.makeText(DownloadExample.this,
R.string.file_downloaded, Toast.LENGTH_LONG);
            mProgressDialog.dismiss();
            break;
        case Groundy.STATUS_ERROR:
            Toast.makeText(DownloadExample.this,
resultData.getString(Groundy.KEY_ERROR), Toast.LENGTH_LONG).show();
            mProgressDialog.dismiss();
            break;
    }
}
};
}

```

**Groundy** 使用一个 GroundyTask 的实现类来下载文件和显示进度:

```

public class DownloadTask extends GroundyTask {
    public static final String PARAM_URL = "com.groundy.sample.param.url";

    @Override
    protected boolean doInBackground() {
        try {
            String url = getParameters().getString(PARAM_URL);
            File dest = new File(getContext().getFilesDir(), new
File(url).getName());
            DownloadUtils.downloadFile(getContext(), url, dest,
DownloadUtils.getDownloadListenerForTask(this));
            return true;
        } catch (Exception pokemon) {
            return false;
        }
    }
}

```

添加这行代码到清单文件中:

```
<service android:name="com.codeslap.groundy.GroundyService"/>
```

这实在是太简单了！只需要从 [Github](#) 上下载最新的 jar 文件就可以开始了。但是要记住，Groundy 的主要用途是在后台服务中调用外部的 REST API，然后更

简单地在 UI 上更新结果。如果你要在你的应用里面做类似的事情，这个库将非常有帮助。

## 2.2 使用 [ion](#)

### 3. 使用 **DownloadManager** 类（只适用于 **GingerBread** 及其以上的系统）

这个方法很酷炫，你不需要担心手动下载文件、处理线程和流之类等乱七八糟的东西。GingerBread 带来一项新功能：DownloadManager。DownloadManager 允许你轻松地下载文件和把复杂计算的任务委托给系统。  
首先，我们来看一下工具方法：

```
/**
 * @param 使用 context 来检查设备信息和 DownloadManager 的信息
 * @return 如果 downloadmanager 可用则返回 true
 */
public static boolean isDownloadManagerAvailable(Context context) {

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
        return true;
    }
    return false;
}
```

方法的名字就已经告诉了我们一切，只有当你确保可以使用 DownloadManager 的时候，你才可以做下面的事情：

```
String url = "url you want to download";
DownloadManager.Request request = new
DownloadManager.Request(Uri.parse(url));
request.setDescription("Some description");
request.setTitle("Some title");
// in order for this if to run, you must use the android 3.2 to compile
your app
// 为了保证这个 if 语句会运行，你必须使用 android 3.2 来编译（译者注：应该是大于
android 3.2 的版本）
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
    request.allowScanningByMediaScanner();

    request.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED);
}
request.setDestinationInExternalPublicDir(Environment.DIRECTORY_DOWNLOADS,
"name-of-the-file.ext");
```

```
// 获得下载服务和队列文件
DownloadManager manager = (DownloadManager)
getSystemService(Context.DOWNLOAD_SERVICE);
manager.enqueue(request);
```

## 最后的一些思考

第一个和第二个方法只是冰山一角。如果你想你的应用更加健壮，你得留意许多事情。这里是一些建议：

- 你必须检查用户是否有 Internet 连接。
- 确保你有正确的权限（Internet 和 WRITE\_EXTERNAL\_STORAGE），如果要检查网络可用性，你还需要 ACCESS\_NETWORK\_STATE 权限。
- 确保你要保存下载文件的目录存在，并且有相应的写入权限。
- 如果下载的文件太大，你可能需要实现一种方法来确保上次的请求失败后，可以接着重来。
- 如果有暂停或者取消下载的选项，用户会很感激你的！

除非你想对下载过程有绝对的控制权，否则我强烈推荐你使用 DownloadManager。因为他已经处理好了上面的大部分建议。

stackoverflow 链接：<http://stackoverflow.com/questions/3028306/download-a-file-with-android-and-showing-the-progress-in-a-progressdialog>

## 2. 如何获取 Android 设备唯一 ID

---

### 如何获取 Android 设备唯一 ID？

#### 问题

每一个 android 设备都有唯一 ID 吗？如果有？怎么用 java 最简单取得呢？

#### 回答 1（最佳）

如何取得 android 唯一码？

好处:

- 1.不需要特定权限.
- 2.在 99.5% Android 装置（包括 root 过的）上，即 API => 9，保证唯一性.
- 3.重装 app 之后仍能取得相同唯一值.

伪代码:

```
if API => 9/10: (99.5% of devices)

return unique ID containing serial id (rooted devices may be different)

else

return unique ID of build information (may overlap data - API < 9)
```

代码:

```
/**
 * Return pseudo unique ID
 * @return ID
 */public static String getUniquePsuedoID() {
    // If all else fails, if the user does have lower than API 9 (lower
    // than Gingerbread), has reset their device or 'Secure.ANDROID_ID'
    // returns 'null', then simply the ID returned will be solely based
    // off their Android device information. This is where the collisions
    // can happen.
    // Thanks http://www.pocketmagic.net/?p=1662!
    // Try not to use DISPLAY, HOST or ID - these items could change.
    // If there are collisions, there will be overlapping data
    String m_szDevIDShort = "35" + (Build.BOARD.length() % 10) +
    (Build.BRAND.length() % 10) + (Build.CPU_ABI.length() % 10) +
    (Build.DEVICE.length() % 10) + (Build.MANUFACTURER.length() % 10) +
    (Build.MODEL.length() % 10) + (Build.PRODUCT.length() % 10);

    // Thanks to @Roman SL!
    // http://stackoverflow.com/a/4789483/950427
    // Only devices with API >= 9 have android.os.Build.SERIAL
    // http://developer.android.com/reference/android/os/Build.html#SERIAL
```

```

    // If a user upgrades software or roots their device, there will be a
    duplicate entry
    String serial = null;
    try {
        serial =
android.os.Build.class.getField("SERIAL").get(null).toString();

        // Go ahead and return the serial for api => 9
        return new UUID(m_szDevIDShort.hashCode(),
serial.hashCode()).toString();
    } catch (Exception exception) {
        // String needs to be initialized
        serial = "serial"; // some value
    }

    // Thanks @Joe!
    // http://stackoverflow.com/a/2853253/950427
    // Finally, combine the values we have found by using the UUID class to
    create a unique identifier
    return new UUID(m_szDevIDShort.hashCode(),
serial.hashCode()).toString();}

```

## 回答 2

好处:

- 1.不需要特定权限.
- 2.在 100% Android 装置（包括 root 过的）上，保证唯一性.

坏处

- 1.重装 app 之后不能取得相同唯一值.

```

private static String uniqueID = null;
private static final String PREF_UNIQUE_ID = "PREF_UNIQUE_ID";

public synchronized static String id(Context context) {
    if (uniqueID == null) {
        SharedPreferences sharedPrefs = context.getSharedPreferences(
            PREF_UNIQUE_ID, Context.MODE_PRIVATE);
        uniqueID = sharedPrefs.getString(PREF_UNIQUE_ID, null);
    }
}

```

```

        if (uniqueID == null) {
            uniqueID = UUID.randomUUID().toString();
            Editor editor = sharedPrefs.edit();
            editor.putString(PREF_UNIQUE_ID, uniqueID);
            editor.commit();
        }
    }
    return uniqueID;
}

```

### 回答 3（需要有电话卡）

好处： 1.重装 app 之后仍能取得相同唯一值.

代码：

```

    final TelephonyManager tm = (TelephonyManager)
getBaseContext().getSystemService(Context.TELEPHONY_SERVICE);
    final String tmDevice, tmSerial, androidId;
    tmDevice = "" + tm.getDeviceId();
    tmSerial = "" + tm.getSimSerialNumber();
    androidId = "" +
android.provider.Settings.Secure.getString(getContentResolver(),
android.provider.Settings.Secure.ANDROID_ID);
    UUID deviceUuid = new UUID(androidId.hashCode(),
((long)tmDevice.hashCode() << 32) | tmSerial.hashCode());
    String deviceId = deviceUuid.toString();

```

谨记：要取得以下权限

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

stackoverflow 链接：<http://stackoverflow.com/questions/2785485/is-there-a-unique-android-device-id>

## 3. 安装 Android SDK 的时候找不到 JDK

### 安装 Android SDK 的时候找不到 JDK

问题

我在我的 win7 64 位的系统上安装 Android SDK 时，jdk-6u23-windows-x64.exe 已经安装上了，但是 Android SDK 的安装程序却因为找不到已安装的 JDK 无法继续下去。这个问题出现过吗？有没有办法解决呢？



## 回答 1:

当你看到这个提示（找不到jdk）的时候按 Back（返回），然后再点 Next(下一步)。这个时候，它将会去寻找 JDK

## 回答 2:

实际安装:

- 系统: windows 8.1
- JDK 文件: jdk-8u11-windows-x64.exe
- ADT 文件: installer\_r23.0.2-windows.exe



安装 64 位 JDK，然后尝试第一个回答中的 back-next 的方法。然后尝试设置 JAVA\_HOME 根据错误信息的提示，但是，仍旧对我没有用处，然后，尝试如下解决办法：

按照它说的做，设置 JAVA\_HOME 在你的系统环境变量中，这个路径要使用正斜杠(/)而非反斜杠()

**注意：** 当我把 JAVA\_HOME 设置为 C:\Program Files\Java\jdk1.6.0\_31 的时候还是不行，但是当我设置成 C:/Program Files/Java/jdk1.6.0\_31 的时候就 ok 了。快把我逼疯了。

如果还不行，就把 %JAVA\_HOME%加在环境变量 Path 的头部。

下面是我的环境变量的配置： - JAVA\_HOME=C:/Program  
Files/Java/jdk1.8.0\_11 - JRE\_HOME=C:/Program Files/Java/jre8 -  
Path=%JAVA\_HOME%;C:...

stackoverflow 链接： <http://stackoverflow.com/questions/4382178/android-sdk-installation-doesnt-find-jdk>