数组扩容

arrays.copyof(arr, len)

数组合并

System.arraycopy(newArr, oldStart, oldArr, oldLen, 扩容的数量)

# [Java transient关键字使用小结及一些需要注意的细节](https://blog.51cto.com/wujuxiang/430211)

1、transient关键字只能修饰变量，而不能修饰方法和类。注意，本地变量是不能被transient关键字修饰的。

2、被transient关键字修饰的变量不再能被序列化，一个静态变量不管是否被transient修饰，均不能被序列化。

3、一旦变量被transient修饰，变量将不再是对象持久化的一部分，该变量内容在序列化后无法获得访问。也可以认为在将持久化的对象反序列化后，被transient修饰的变量将按照普通类成员变量一样被初始化。

序列化保存的是对象，但静态变量是属于类的。

**boolean 和 Boolean**

**原始类型和包装类，一个可以为null一个不可。主要区别是从列表或者键值取值时。因为从hashMap可以存放null**

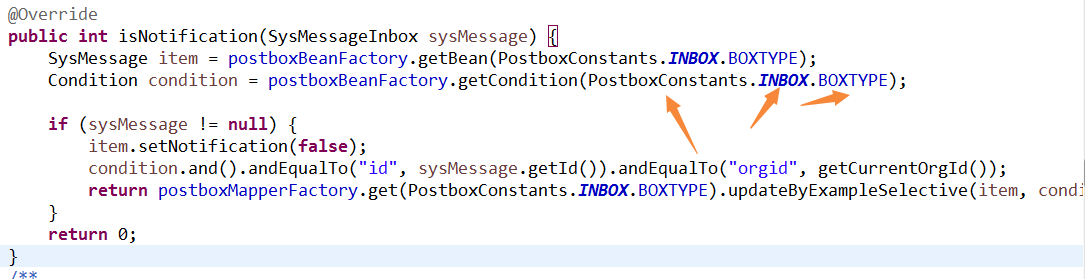
**博客参考:**

[**https://blog.csdn.net/kewen1989/article/details/48679985**](https://blog.csdn.net/kewen1989/article/details/48679985)

枚举的简单使用

声明一堆（可看做初始化的对象），接着可直接调用其属性，为了更加符合面向对象特性，建议把属性进行封装，暴露get方法进行调用，**下面例子没有进行封装，不推荐使用**。





PostboxConstants.***INBOX***.BOXTYPE这里获取到了INBOX这个字符串

# [Java基础——枚举详解](https://www.cnblogs.com/kailejun/p/6624471.html)

# [Java枚举7常见种用法](https://jingyan.baidu.com/article/e4d08ffdcf664d0fd2f60dbf.html)

[那么如何在java中获取Request body中的内容呢](https://blog.csdn.net/pengjunlee/article/details/79416687)





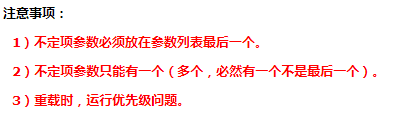
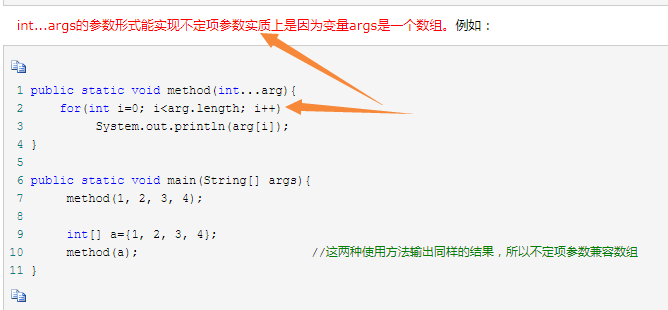
获取项目根路径

类类型.getResource("/").getFile().toString()

实际操作的文件时class文件，java是解释型语言



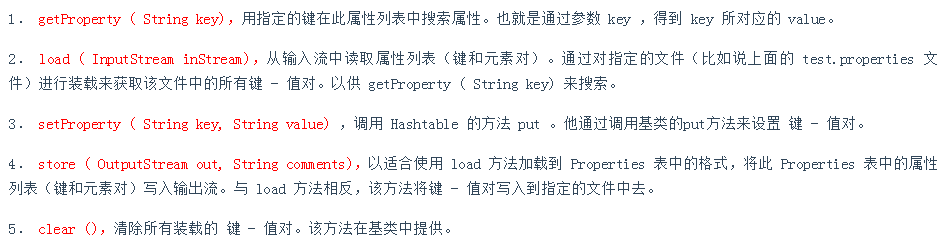
它的本质，如何获取它



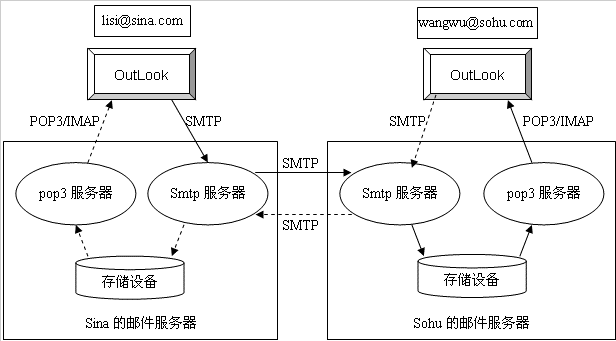
先匹配确定参数的方法，再匹配不定项参数的方法

<https://www.cnblogs.com/xy-hong/p/7192796.html>

[Properties读取配置文件](https://www.cnblogs.com/bakari/p/3562244.html)



JavaMail简述

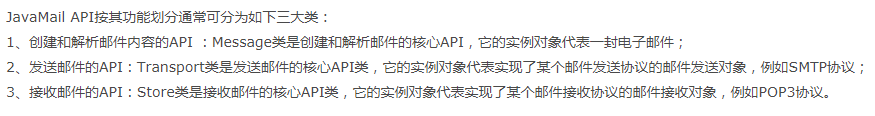


lisi@sina.com与sina.com的SMTP服务器建立连接，通过lisi的account、password。Lisi向sina.com的SMTP服务器发送邮件，SMTP服务器解析目标地址，属于自己的范围则存储起来，不属于则通过DNS查询目标地址，这里既是sohu.com。查询后得到相应的服务器信息，然后通过SMTP协议和对方进行连接并把邮件发送给对方。

邮件操作也是基于Session进行。通过Session获得Store(服务器信息)，通过Store获得服务器的“文件夹”，从“文件夹”中获取邮件(载体Message)。

JavaMail API是Sun公司为方便Java开发人员在应用程序中实线邮件发送和接收功能而通过的一套标准开发包，它支持一些常用的邮件协议，如SMTP、POP3、IMAP和MIME等。要用这套API，当然需要先下载相关jar包。

下载地址：<https://java.net/projects/javamail/pages/Home>。

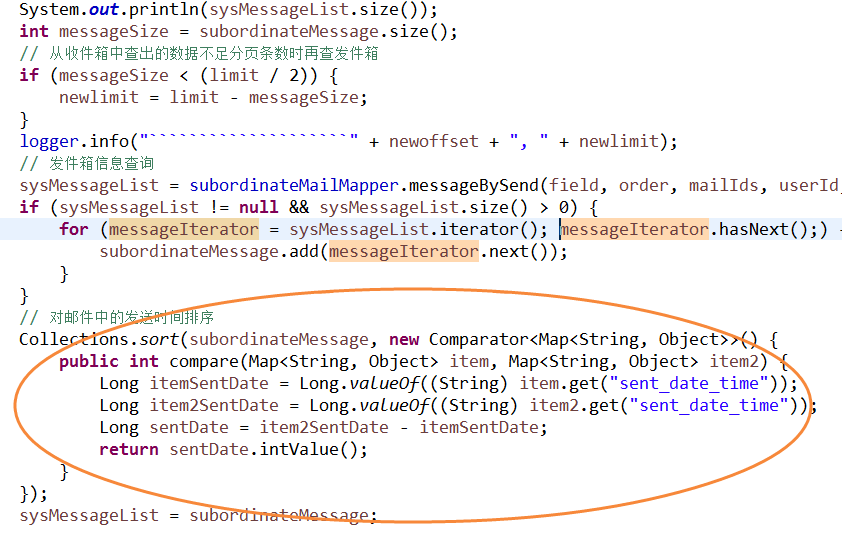


<https://www.cnblogs.com/huangminwen/p/6069072.html>

[Long转int](https://www.cnblogs.com/EasonJim/p/7722860.html)

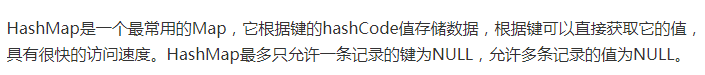
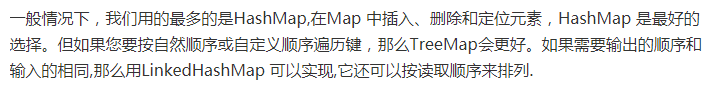


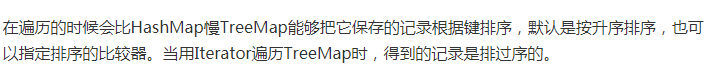
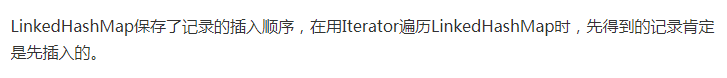
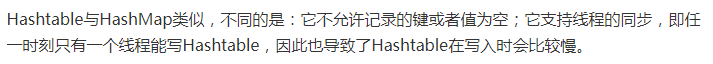
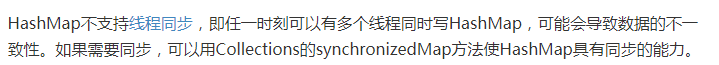
[对List Map根据Map某个key-value进行排序](https://www.cnblogs.com/wangqiang4518/p/7144421.html)



[SimpleDateFormat(SSS(大写)，毫秒)](https://blog.csdn.net/qq_36761831/article/details/79691119)

[**HashMap、LinkedHashMap、TreeMap**](https://zhidao.baidu.com/question/553310922585280092.html)



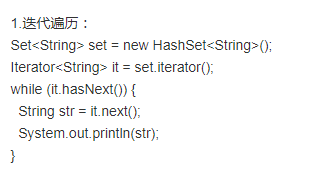


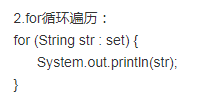
for循环前加"out:"

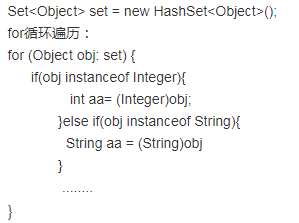


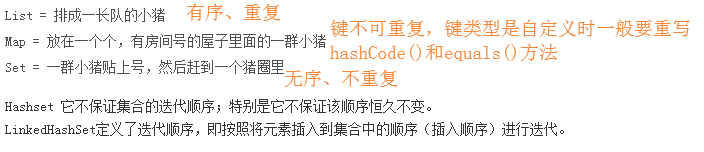
<https://zhidao.baidu.com/question/215134231.html>

Set遍历方式









<https://blog.csdn.net/u011301203/article/details/42422713>

List、Set、Map遍历

<https://www.cnblogs.com/quyixuanblog/p/5216881.html>



正则表达式，匹配空格、数字

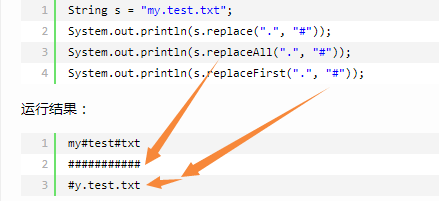




<https://blog.csdn.net/y394996630/article/details/46841807>

# [java字符串的替换replace、replaceAll、replaceFirst的区别详解](https://www.cnblogs.com/jun9207/p/5191857.html)

String的replaceAll跟replaceFirst用到了正则表达式！



<https://www.cnblogs.com/jun9207/p/5191857.html>

为什么使用Log而不使用System.out

Linux环境下中不能出来控制台，只能输出到文件中

System.out日志打印不可控制、打印时间无法确定、不能添加过滤器、日志没有级别区分。在LogCat中可以根据需要添加一个自定义的过滤器，在成百上千条日志中轻松找到我们要想要的信息。

Log.d（Tag，msg）；

第一个参数是tag，一般传入当前的类名就好，主要用于对打印信息的过滤。第二个参数是msg，即想要打印的具体内容。

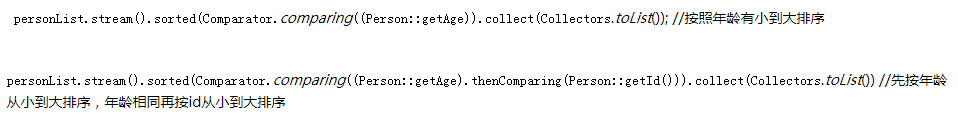
<https://blog.csdn.net/Sky_xiaye/article/details/51224738?utm_source=blogxgwz8>

[Java集合中removeIf的使用](https://blog.csdn.net/qq_33829547/article/details/80277956)

在JDK1.8中，Collection以及其子类新加入了removeIf方法，作用是**按照一定规则过滤集合中的元素**



sorted 排序值comparing 和thenComparing

排序

学习链接：

<http://www.importnew.com/15259.html>

<https://blog.csdn.net/u011381576/article/details/79422498>

<https://my.oschina.net/xinxingegeya/blog/2046405>

src = source      源  
dst = destination 目的

File file = FiltUtil.copyFile(src, dst);

**JAVA中json格式判断空值**

**json串**

**{“error\_code”: \*\*, “error\_msg”: “SUCCESS”, “result”: null}**

// 返回的值

System.***out***.println(jsonObj.get("result"));

// 返回的类

System.***out***.println(jsonObj.get("result").getClass());

// 判断空值

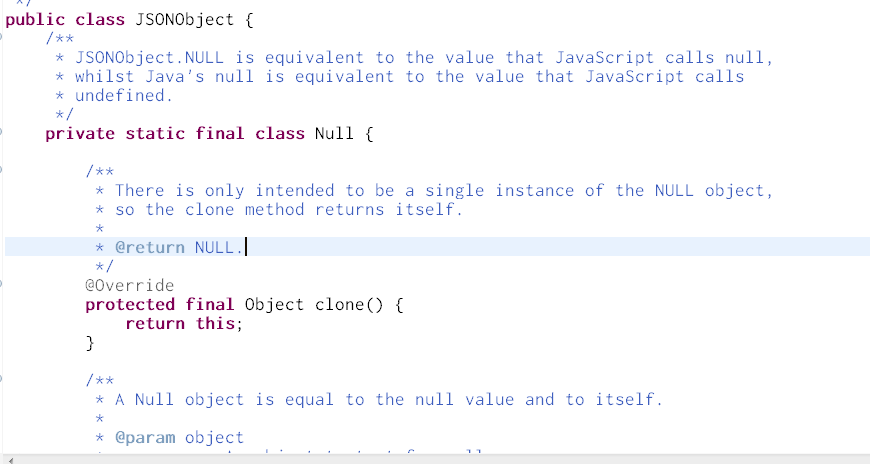
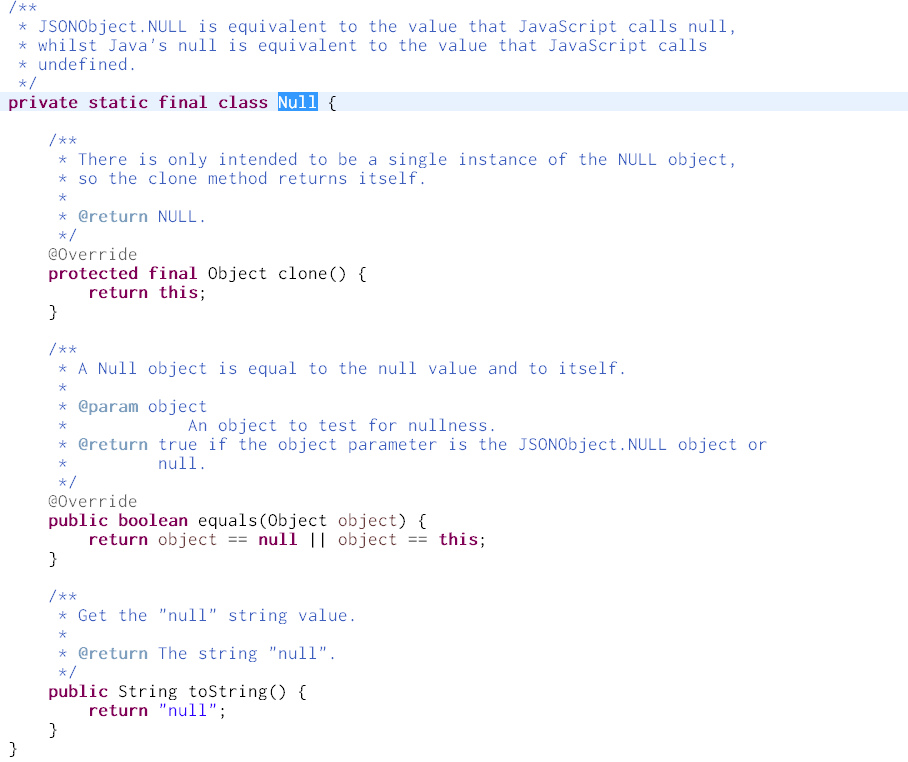
System.***out***.println(JSONObject.***NULL***.equals(sonObj.get("result")));

// 直接取类名，完整类名换成->getName() System.***out***.println(jsonObj.get("result").getClass().getSimpleName());

System.***out***.println(jsonObj.get("result").getClass().getSimpleName().equals("Null"));

这里的JSON包是





总而言之，JSON对象返回的null值是JSONObject类的一个内部类的类型，所以此null非彼null

判断空值总结：

1、用引用包自带的判断null值方法来判断，不同包判断方式存在差异；

2、先取返回值为null的键，分析返回的类；

创建Java对象的四种方式

1、new关键字

高效、可以通过任何构造器来进行

2、反射 — newInstance()

只能通过**无参构造器**来进行，所以该类必要拥有一个无参构造器。相对new关键字比较低效，结果返回Object类型，需要进行强制类型转换。

3、反序列化的方式

通过磁盘文件或者网络传输的方式进行反序列化。

ObjectOutputStream.readObject();

4、clone()方式

要使用它的类必须实现Cloneable接口，且要重写clone()方法

Java序列化与反序列化

序列化：

**static** **void** objetSerializable() {

System.***out***.println("@@ 对象序列化-->开始");

String objectFilePath = "C:/Users/Administrator/Desktop/temporarya.txt";

TemporaryA temporaryA = **new** TemporaryA("admin", 22);

FileOutputStream outputFile = **null**;

ObjectOutputStream output = **null**;

**try** {

outputFile = **new** FileOutputStream(**new** File(objectFilePath));

output = **new** ObjectOutputStream(outputFile);

output.writeObject(temporaryA);

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if** (outputFile != **null**) {

outputFile.close();

}

**if** (output != **null**) {

output.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

System.***out***.println("@@ 对象序列化-->结束");

}

反序列化：

**static** **void** objetUnSerializable() {

System.***out***.println("@@ 对象反序列化-->开始");

String objectFilePath = "C:/Users/Administrator/Desktop/temporarya.txt";

FileInputStream inputFile = **null**;

ObjectInputStream input = **null**;

**try** {

inputFile = **new** FileInputStream(**new** File(objectFilePath));

input = **new** ObjectInputStream(inputFile);

Object obj = input.readObject();

TemporaryA temporaryA = (TemporaryA) obj;

System.***out***.println(temporaryA.getAge() + ", " + temporaryA.getUserName());

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

} **finally** {

**try** {

**if** (inputFile != **null**) {

inputFile.close();

}

**if** (input != **null**) {

input.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

System.***out***.println("@@ 对象反序列化-->结束");

}

[Java大神告诉你，有哪些创建对象的方法？](https://baijiahao.baidu.com/s?id=1618808708841190285&wfr=spider&for=pc)

Java存储区详解

寄存器、栈、堆、静态存储区、常量存储区、非RAM存储区。

1. **寄存器(register)**。寄存器与其他的存储区不同,它位于CPU中,所以它是存取速度最快的存储区.但是由于CPU中寄存器的数量是非常有限的,所以寄存器的使用是由CPU按需进行分配.编程的时候我们没有必要去对它进行直接控制.并且在程序中也没办法找到到底什么时候使用了寄存器的依据。

2. **栈(stack)**。它位于随机存取存储器(RAM)中,并通过由CPU直接支持的栈指针(stack pointer)来进行操作.栈指针通过向下移动来分配新的存储空间,并通过向上移动来释放存储空间.这是仅次于寄存器的另一种高速有效地分配存储空间的 方式.在创建程序期间,Java编译器必须知道存储在栈上的所有数据的准确大小和它们的生命周期.这样它才能够产生代码来让栈指针上下地移动.但这种约束 限制了代码的灵活性,所以当某些Java存储区在栈上的时候——特别是,对象引用(object reference)――而Java对象本身并不被存储在栈上。

3. **堆(heap)**。这是存储Java所有对象的内存池(同样在RAM区域中).堆的优点是,它不像栈一样,编译器并不需 要知道它应该从堆中分配多大的存储空间,或者存储空间应该在堆上存在多久.因此,在堆上分配多大的存储空间具有很大的灵活性.无论你何时需要创建对象,你 只需要简单的在代码中使用关键字new来创建它就行了.在这段代码被执行时候就在堆(heap)上面分配相应的存储空间.当然,为这样的灵活性也需要付出 一定的代价:在堆上分配存储空间比在栈上分配存储空间要更加耗时(如果你能够在Java中像在C++一样,在stack上创建对象的话)。

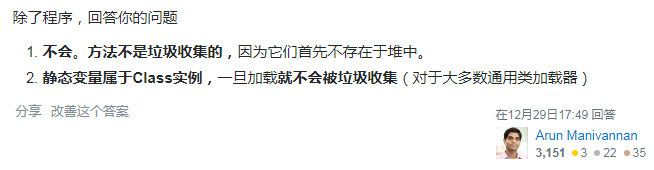
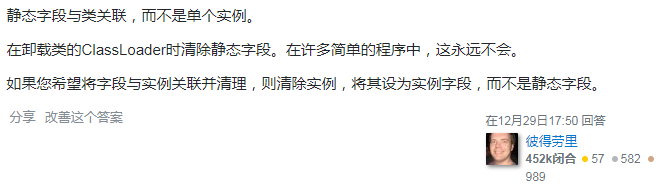
4. **静态存储区(static storage)**。”静态(static)”在这里表示”在一个固定的位置”(尽管它同样在RAM中).静态存储区包括:对于程序的整个运行期间都有效的数据.你能够使用关键字static来指定特殊的对象为静态的.但是Java对象自身却从来不放置在静态存储区中.

5. **常量存储区(constant storage)**.常量通常被直接放置在程序代码当中.这是安全的,因为它们永远不能被改变.在某些时候多个常量能够自己分隔开,这样在嵌入式系统中可以被任意地放置在只读存储器(ROM)中。

6. **非RAM存储区(non-RAM storage)**。如果数据完全存在于程序之外,在程序未被运行时数据也能存在,而不受程序的控制.对于此类情况,有两个主要的例子来说明:一,对象流(streamed objects),这种对象通常以字节流的形式被送到其他的机器去.二,持久化对象(persistent objects),这种对象存放在磁盘上,即便是程序中止以后它们仍然保持原有状态.这些存储器都是将对象以某种形式保存在其他的介质中,然后在需要的时候再把它恢复为常规的基于RAM的对象.Java提供了对轻量级持久性的支持.在未来的Java版本中,Java将提供更加完备的持久性的支持。

[Java存储区详解](https://www.cnblogs.com/kakusousou/p/4001582.html)

Java中的垃圾收集器是否可以处理java中的静态变量或方法？

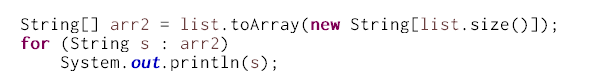


答案是不会的。因为它是同类关联的，属于Class实例。除非

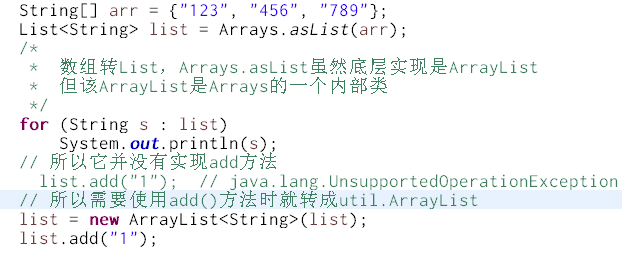
# [垃圾收集器是否可以处理java中的静态变量或方法？](https://stackoverflow.com/questions/13126833/does-the-garbage-collector-work-on-static-variables-or-methods-in-java)

List转数组，数组转List

List转数组



数组转List

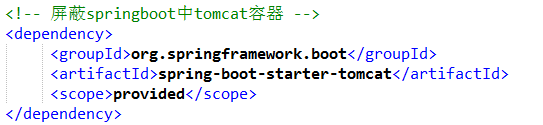


SpringBoot部署到独立Tomcat中，三步走

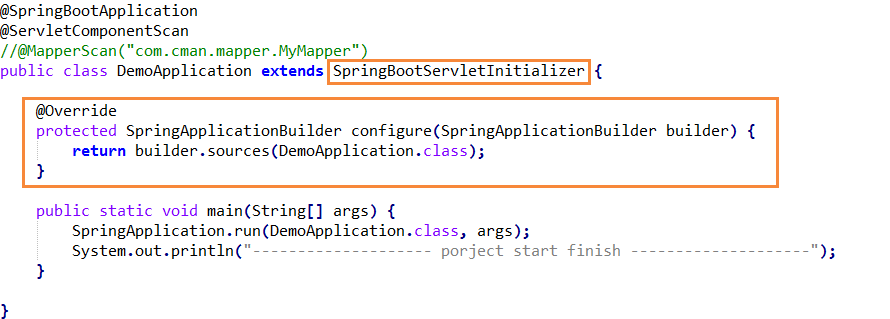
1、修改pom文件，改变打包方式（变成war包）；



2、添加tomcat依赖，作用域改为private，从而屏蔽springboot内置的Tomcat；



3、启动类中继承SpringBootServletInitializer，并且重写其configure(SpringApplicationBuilder)方法



实现Excel的导入导出操作

1、使用apache的POI包

功能强大，操作复杂，可支持复杂的公式运算，支持单元格的样式等设置。不仅支持excel，还支持work，pdf等等文件的操作。

2、使用JXL

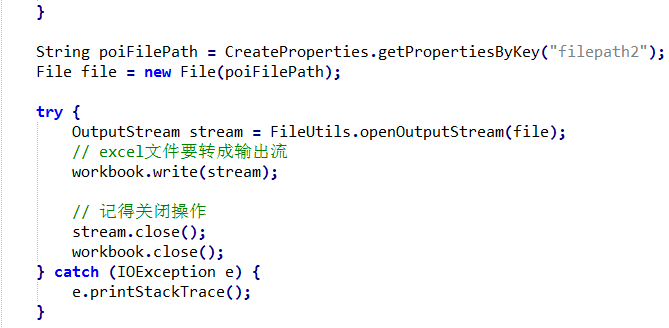
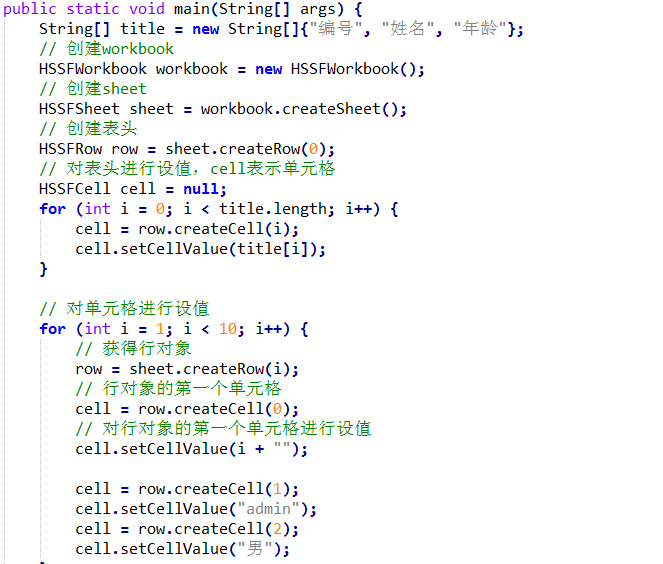
功能比POI略差，操作简单，不能支持复杂的公式运算，支持单元格的样式等设置

3、使用FastExcel

功能简单，操作简单，不支持单元格的样式等设置

1、POI实现Excel导出，以Stream（流）的方式进行输出

创建workbook对象（创建Excel）- 创建sheet – 得到row – 得到cell

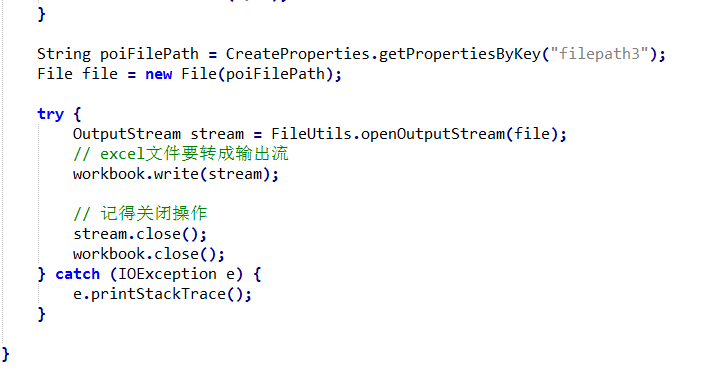


2、POI实现Excel导入，以Stream的方式进行读取

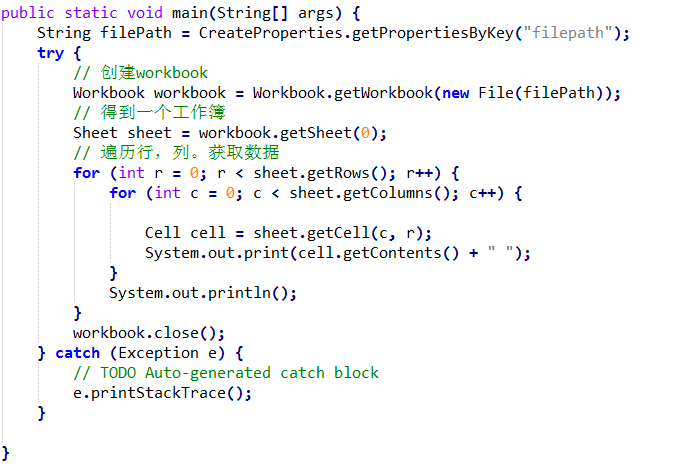
创建workbook对象（创建Excel）- 创建sheet – 得到row – 得到cell



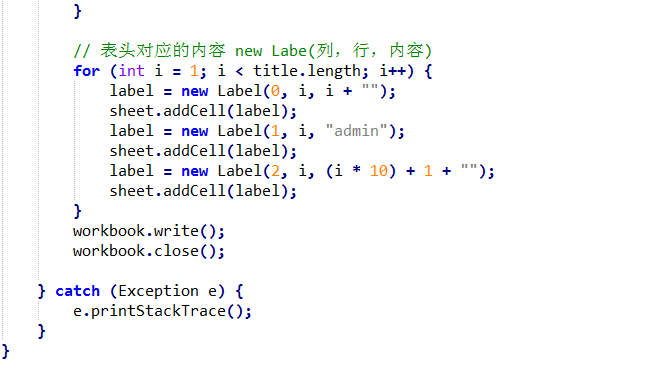
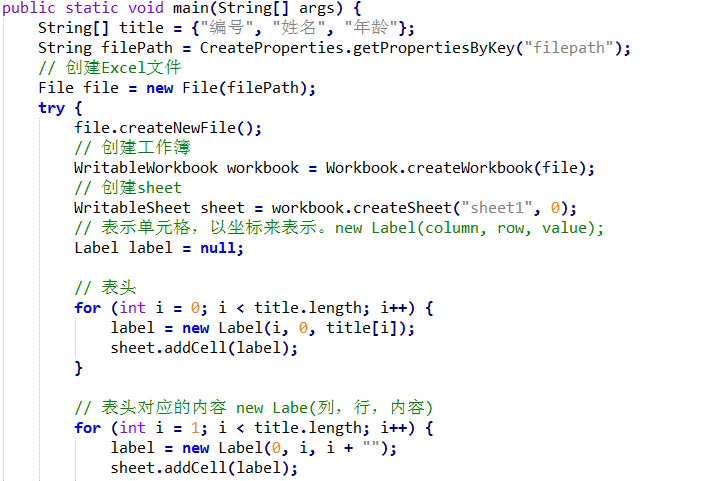
针对2007及更高版本的xlsx，HSSFWorkbook换成XSSFWorkbook，HSSFSheet，HSSFRow，HSSFCell等去掉HSSF。



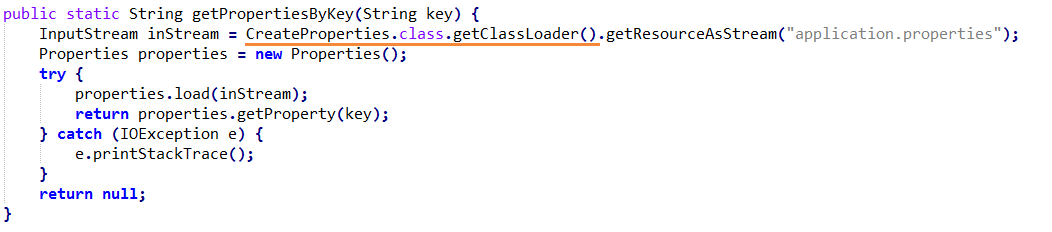
3、JXL实现导入



4、JXL实现导出



读取properties文件，application.properties我放到了src目录下



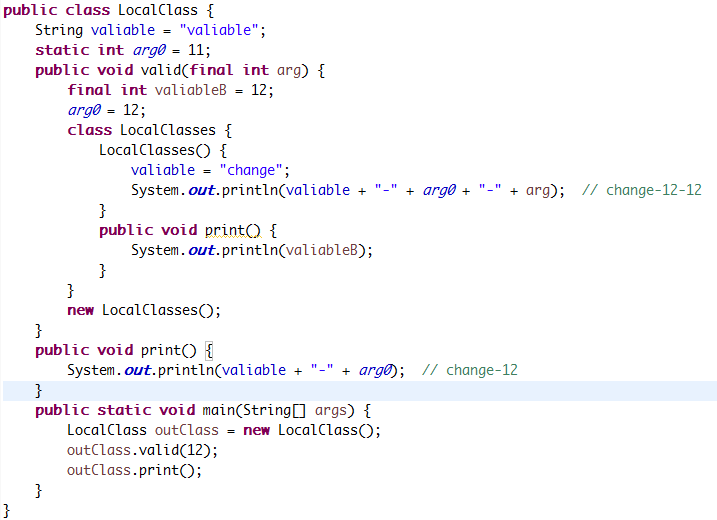
java内部类

内部类、本地内部类、匿名内部类

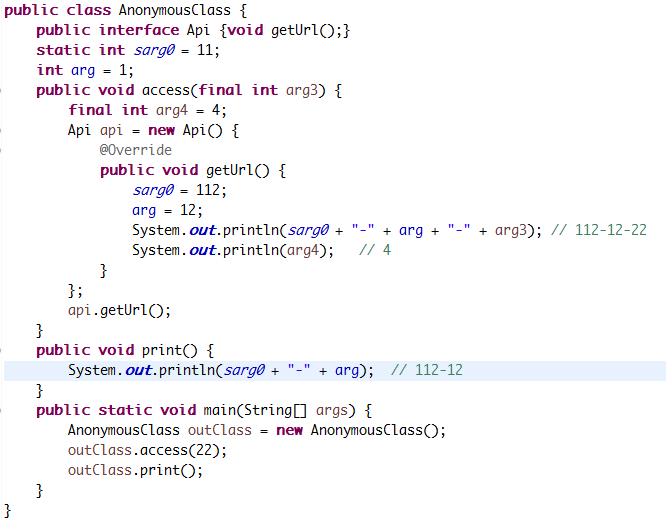
**内部类**可以访问外部类的成员变量、静态成员变量。可以对外部类的变量进行覆盖定义，但是这将是指向一个新的内存地址。如果内部类要使用static，将成为静态内部类。



**本地类**，定义在块（方法、for、if等）中的类，它跟内部类的主要区别是**访问方法的局部变量跟参数**。在java8之前，访问局部变量必须使用final关键字，指定该变量是最终或有效最终的。java8乃至更高的版本就可以省略final。本地类不能存在成员接口并且是非静态的，即不允许在方法中使用static关键字，除非是定义静态常量变量。

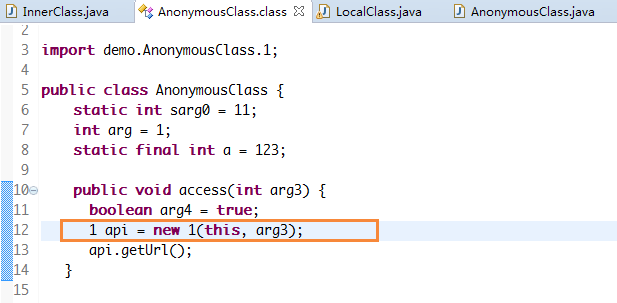
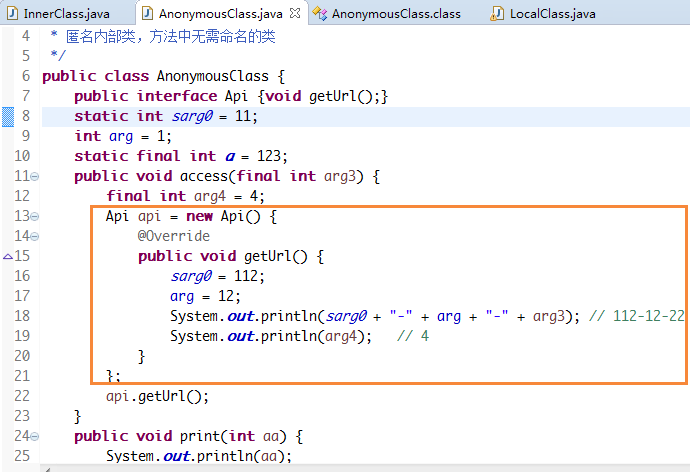
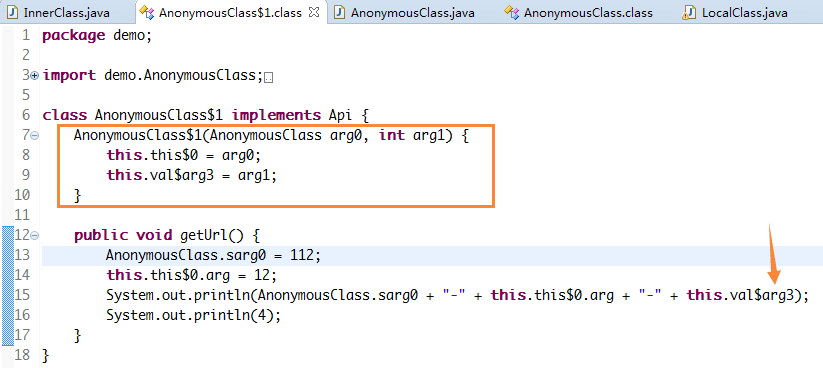


匿名内部类，跟本地类基本相似，如果你只调用一次本地类的话，那么就使用匿名内部类吧，但值得注意的是，本地类是类声明，匿名内部类是表达式，而且不能拥有构造器。



那么，为什么本地内部类跟匿名内部类访问局部变量时要加入final关键字呢？

内部类不是直接调用方法的参数，而是利用自身的构造器把需要用到的局部变量拷贝一份，成为自己的成员变量。那么这样在内部类中的属性和外部方法的参数两者从外表上看是同一个东西，但实际上却不是。所以为了保持数据的一致性，使用final关键字来保证形参地址的不变。

[匿名内部类 使用外部的变量为什么要声明成final](https://www.cnblogs.com/qinjiayan/p/4918154.html)

arg指的是实参，param指的是形参

# [parameter和argument的区别](https://zhidao.baidu.com/question/2116957332636950747.html)

# [自定义Exception异常](https://www.cnblogs.com/aeolian/p/10449506.html)

1、继承Exception类

2、构造方法中调用父类构造方式，可以自定义输入其他信息，如异常描述等



dto 与 entity

前者用于数据的传输，如传递多个参数时，可以不选择map，而是新建dto包建立类来管理属性，可读性、易维护

后者一般用于与数据库建立映射关系，通过对应数据库中的某张表

泛型符号含义

* E - Element (used extensively by the Java Collections Framework)
* K - Key
* N - Number
* T - Type
* V - Value
* S,U,V etc. - 2nd, 3rd, 4th types