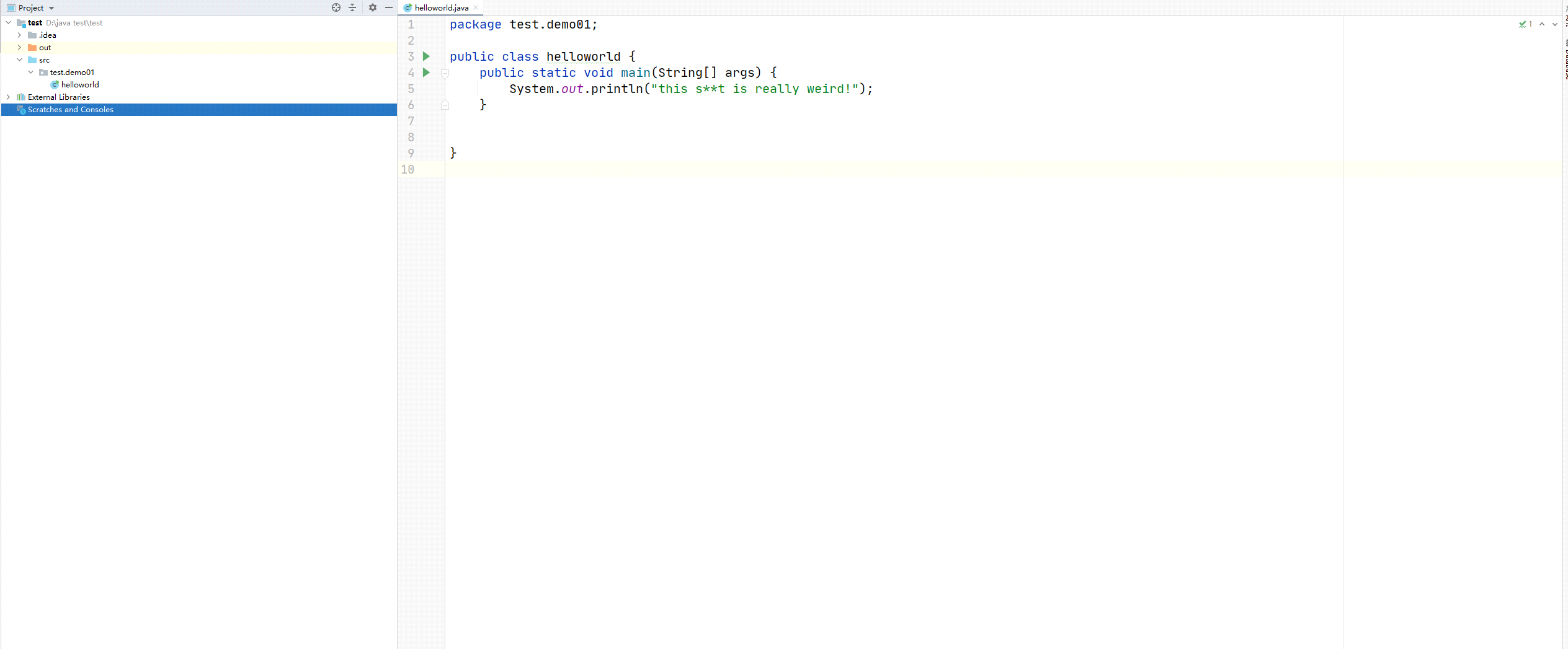
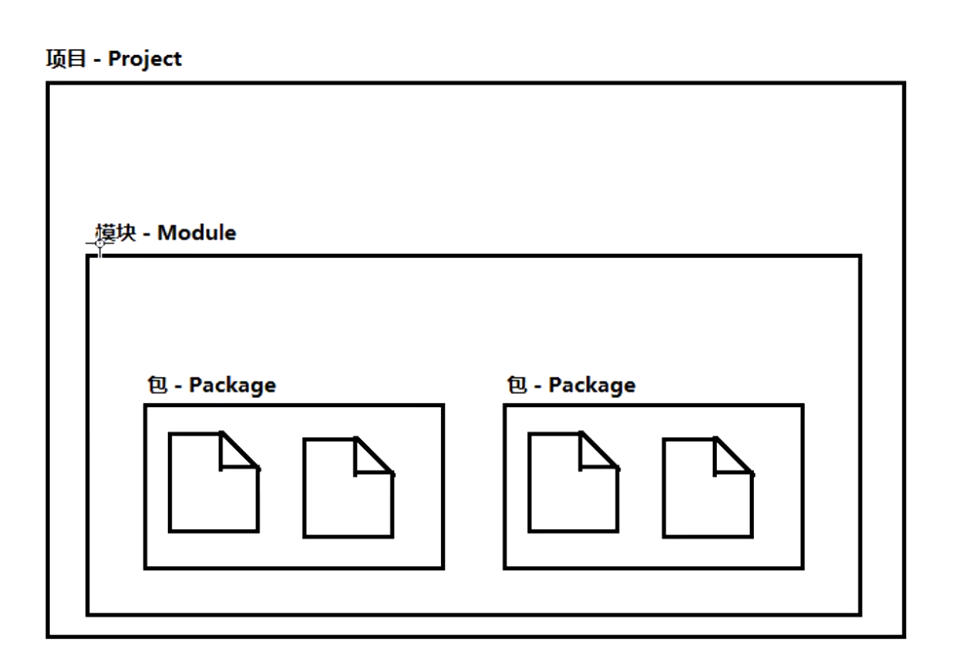
* java同csharp一样，是完全面向对象的语言。所有的方法都应写于类体中。
* idea的项目结构：

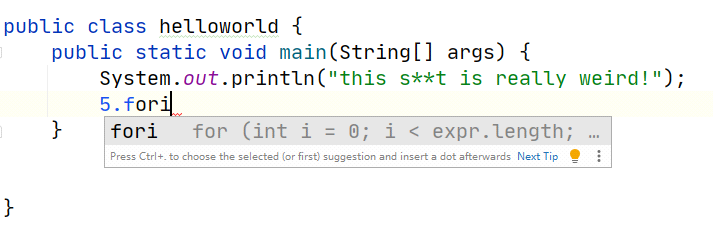


Class建于package内。方法（包括main主入口）写在类体内。

* 常用快捷键：
* 导入modele

首先将文件夹放在project目录下；file-project structure-modules-+号-import modules

* 快捷输入循环： 5.fori



* 函数重载overload：

和c++基本相同。

多个方法的名称一样，但参数列表不一样。

重载与以下因素无关：参数名称（废话）、方法返回值类型.

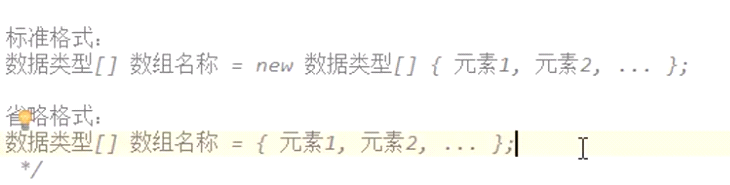
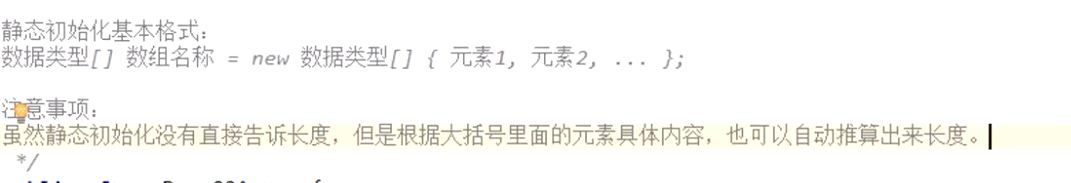
脑部为啥一个print（）随便放参数啥都能输出。

* 数组：

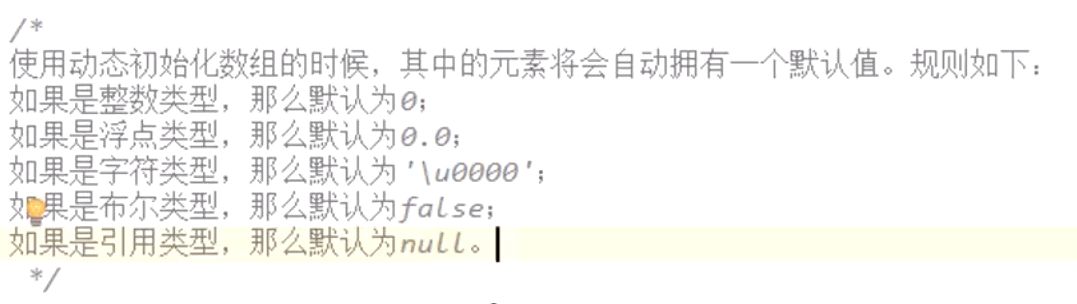
和c++一样的静态数组，无法改变长度。存储和引用方法相同。

初始化语句略有不同：

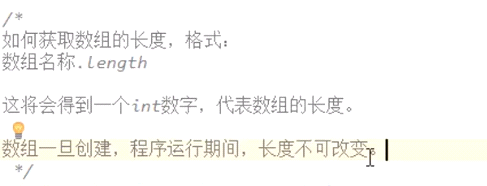


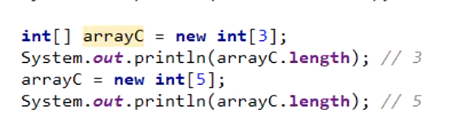


* 数组的default：



* 获取数组长度：

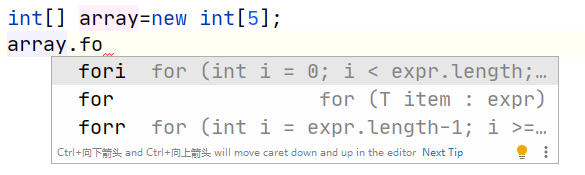


易错点：

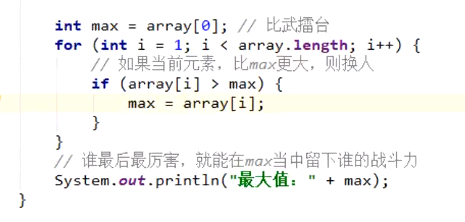
这种情况下数组长度好似改变了，其实不然；这其实是两个数组（两次new），只不过第一个数组叫arrayC，第二个数组抢了第一个数组的名字。

数组的遍历：

数组名称.fori 即可



数组求最值：（没有库函数）

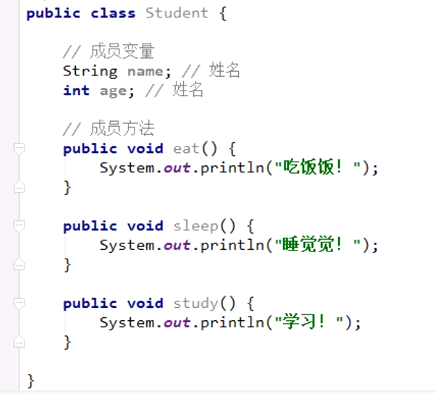


* 输入输出：

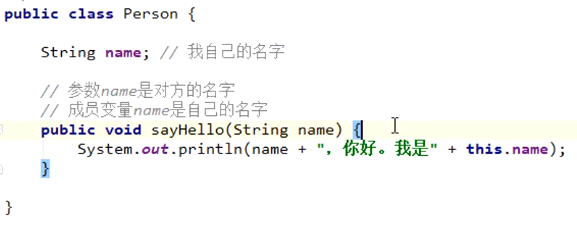


* 类的定义：

成员方法一般不public

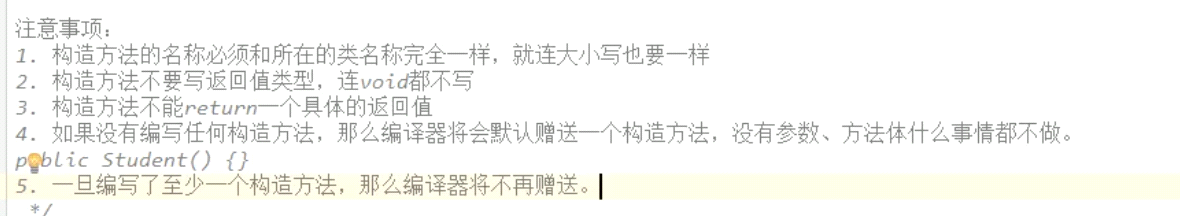


This指针：同c++



This肯定是写在方法内部的，this指的作用域就是本方法所在类。

构造方法：和c++构造函数一样。

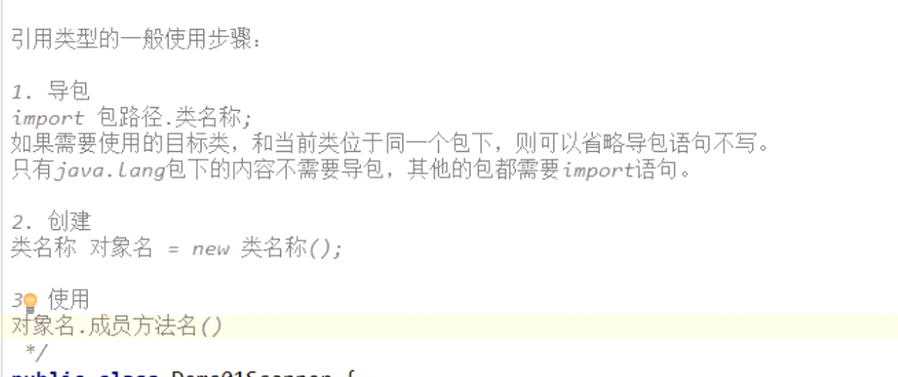


* 常用API



Api就是一大堆类、一大堆方法。文档就是字典。文档的话可以先看1.6中文版。

使用api的方法：

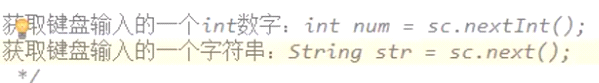


Scanner：

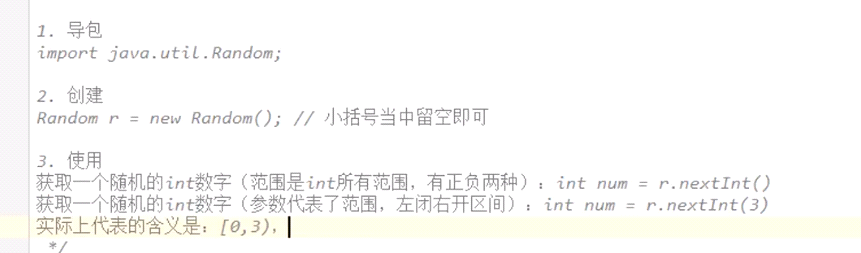




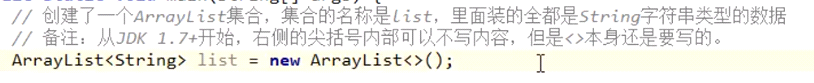
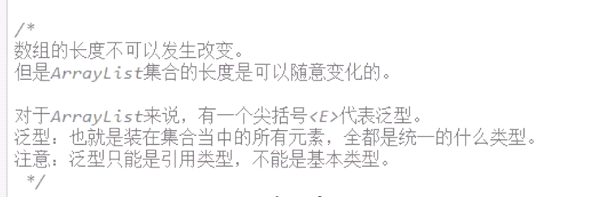
查api文档得知：

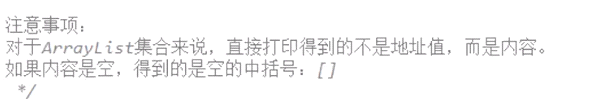


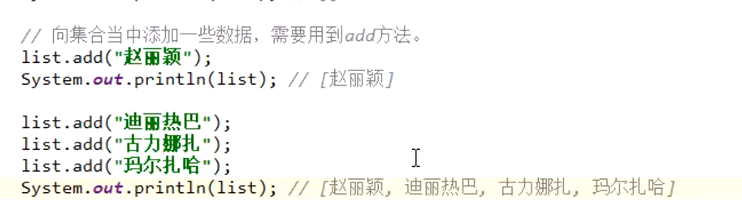
Random：产生随机数



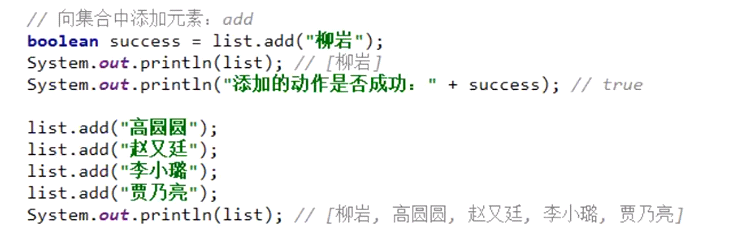
* ArrayList类：类似c++中的vector，顺序表实现长度可变的数组

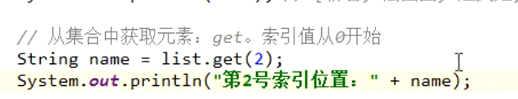


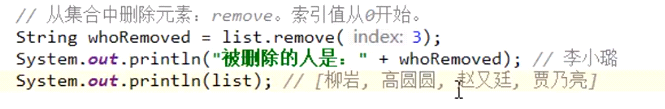


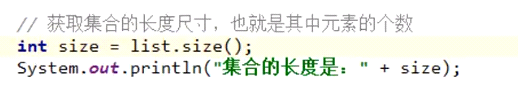


常用方法：

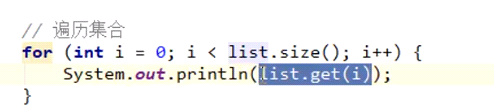




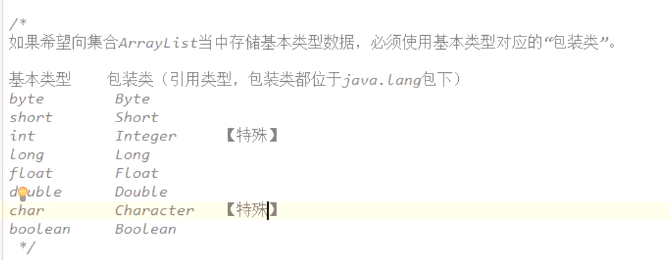


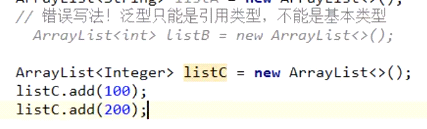


遍历list：名称.i 即可



如何使用list存放基本类型数据：

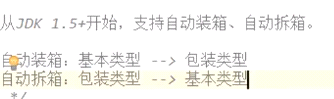




Integer直接当作int用就可以。例如：

int num=listC.get(1);//200

这是因为：

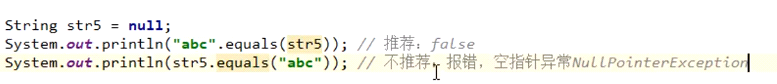


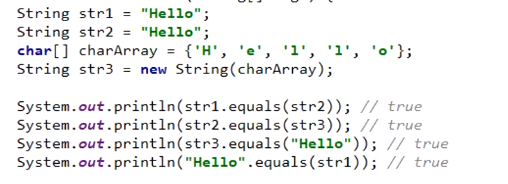
* 字符串相关方法：

字符串比较：

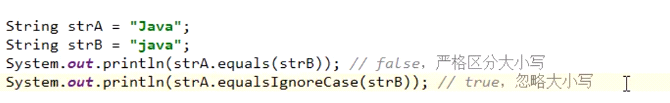
如果直接用==进行比较，比的不是内容是否相同，而是地址是否相同。

想要比较内容，应该用 .equals()：

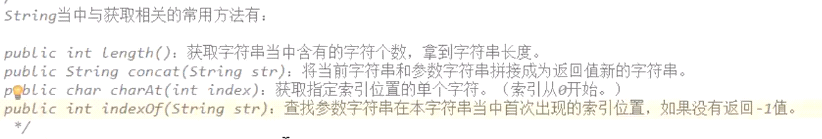




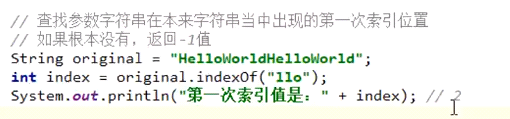
关于大小写：



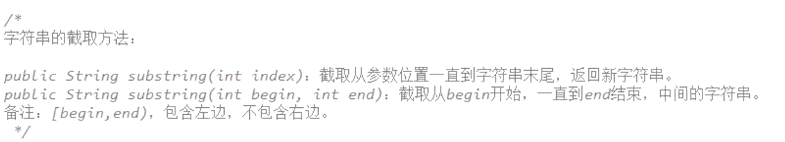
字符串的获取：





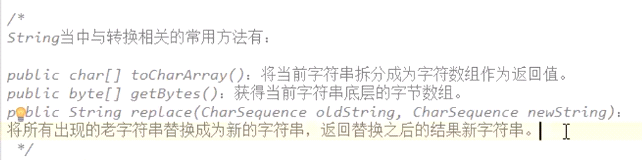


字符串的截取：



Tips：alt+enter，可以自动建立接受的容器。输入list.size（）,按alt+enter，左边自动出现int num；

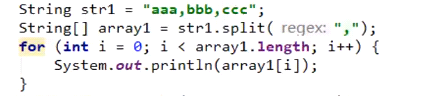
字符串的转换：

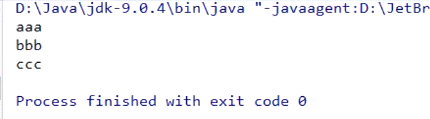


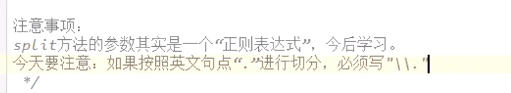
Charsequeence就是可以接受字符串类型

字符串切割：等于c#的split





输出

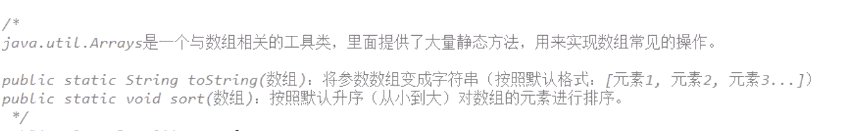


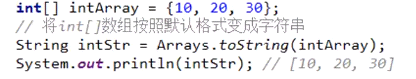
.在正则表达式中有特殊含义，必须将其转义

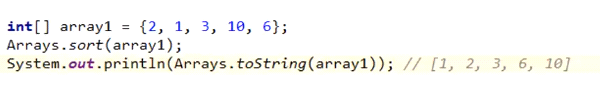
* Array工具类

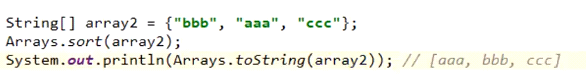
注意：静态方法不用new，直接就能.方法！！！

常用的两个方法：







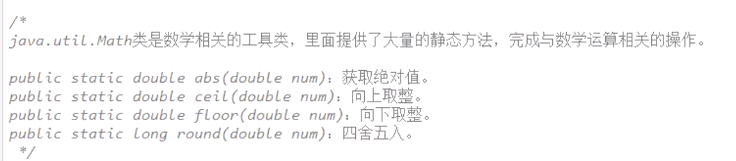


排序备注：



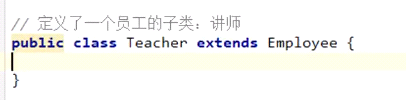
Tips：查看数组用tostring就可以直接查看，省得还得用for循环输出

* Math工具类

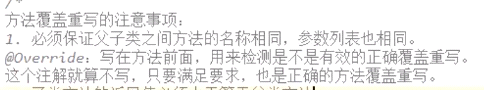


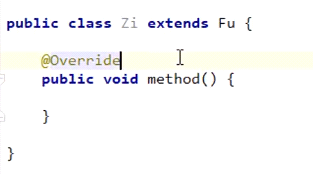
* 类的继承：与c++基本相同

关键字：estends



继承中的override：和c++相同



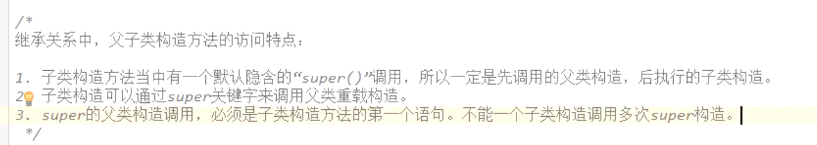


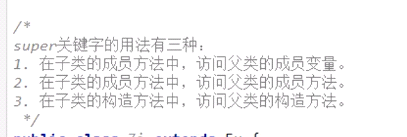
如果是正确重载，不会报错；否则@override会报错

另：子类中的重载函数和父类中的对应函数的返回值可以不一样，但必须是父类返回值的子集。比如父类是public object method（）；子类可以是public string method();因为string是method的子类

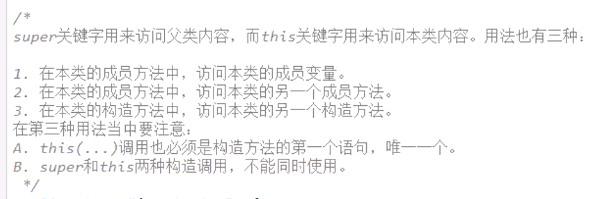
Super指针：

为了实现代码的重用，子类中的覆盖函数中和父类相同的部分不用再写一遍，直接super.方法名（）即可；但是，即使你不写super，默认也会有一个隐含的super（），那就是父类的构造函数！new一个子类，其实先执行了父类的构造函数

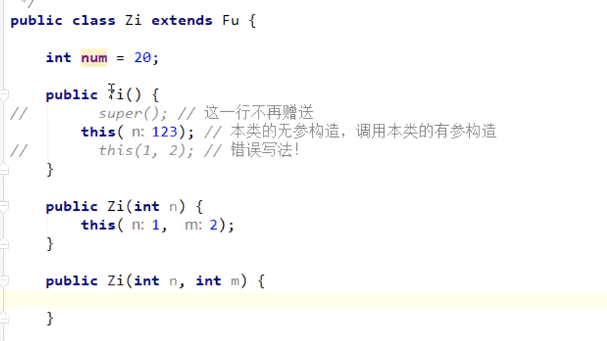




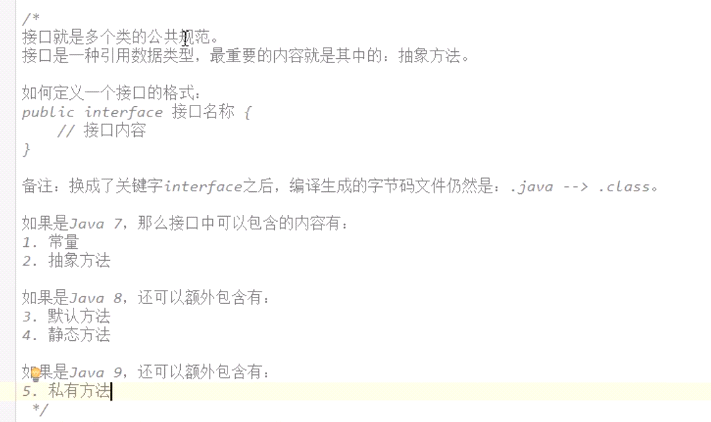
This指针，和super指针并列，super用于调父类，this就是调自己



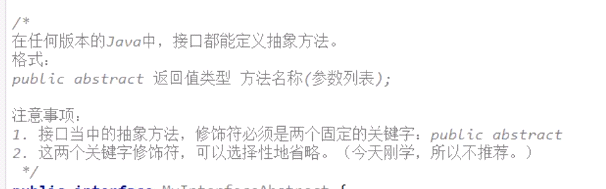
第三种用法：



* 关于接口interface



创建一个interface：

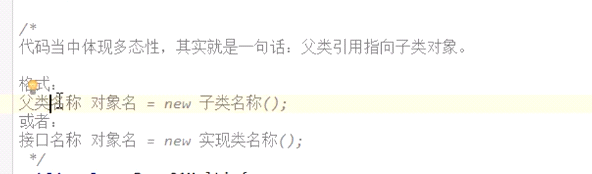


接口想要使用，需要一个类将其方法全部实现。用关键字implement。Tips：输入类名和implement后，输入alt+回车，可以自动在类中实现接口的全部方法。

接口的总结：

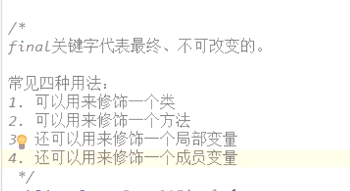


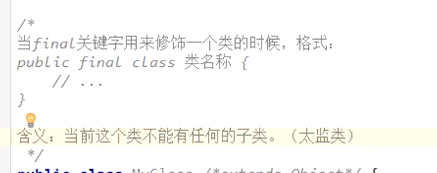
* 多态性

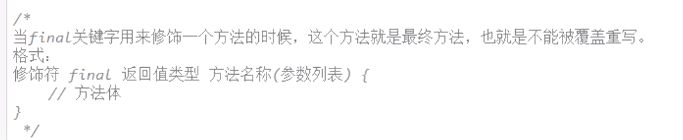


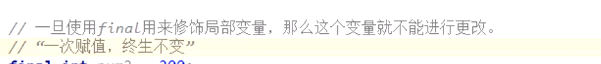
New出来的子类可以当作父类用（左父右子就是多态！更多内容可以参考c#note相关内容）

* Final关键字：









* 权限关系：



（其实protected只有继承时才会使用）

Tips：新建一个类后，alt+i直接空参构造函数；alt+ins直接全参构造函数（其实不止全参构造函数，还有很多常用的重写方法）；

#object类

是一切类的基类。当你自己写一个类时，其实默认继承了object。

也就是说，一切类都可以使用object类中的方法。

Tostring（）方法：

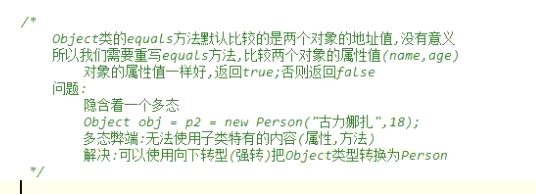
直接打印，出来的是地址值；所以应当重写一下同string（）方法。

快捷重写方法：alt+insert，在里面选即可。

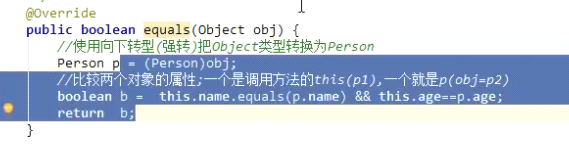
Equals（）方法：

默认的object.eauals()比较的是地址值，所以自己创建类的时候也需要将其重写。

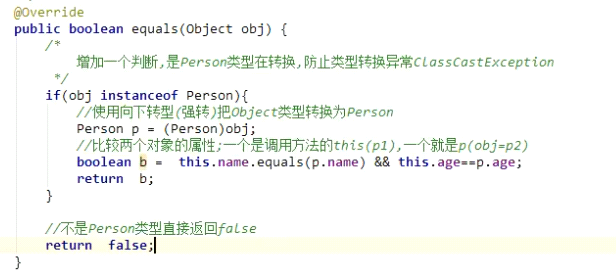
需要注意的是：



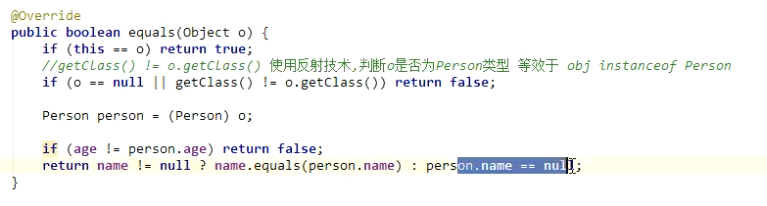
Like this：



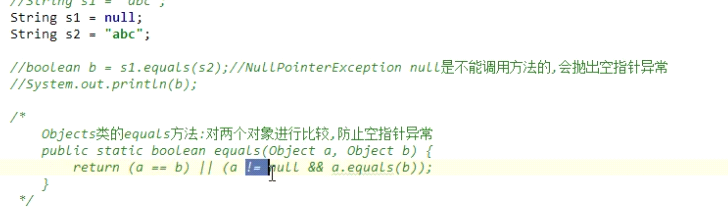
当然了，直接强转容易出错，所以可以加一个判断：



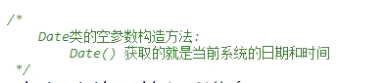
来看看java默认的重写equal()idea默认版：



Objects类中的equals（）容忍空指针，object类不行

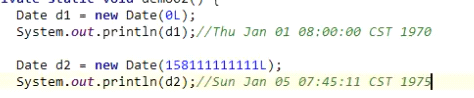


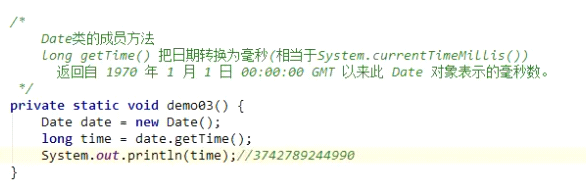
#Date类：util包里



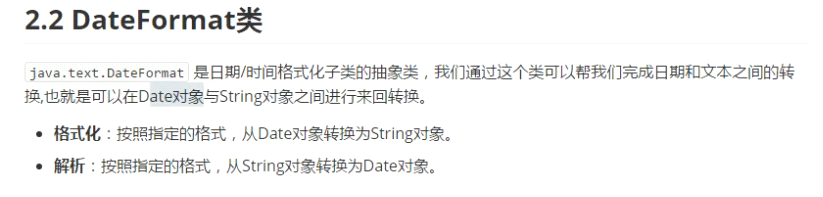


毫秒值的原点是：1970年格林尼治00：00；like this：





#DateFormat类

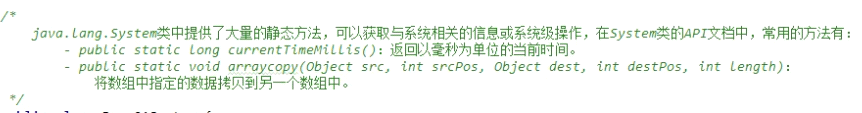




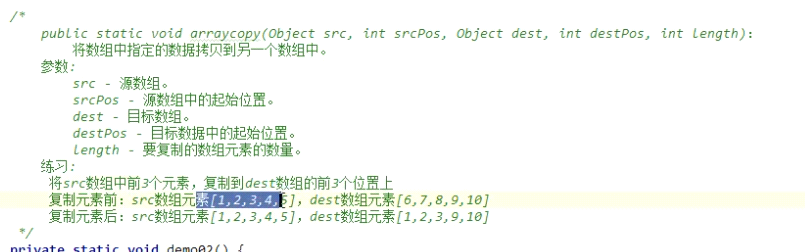
是个抽象类，所以都具体成一个子类。

……时间相关的还是先跳过吧，好像没啥用。

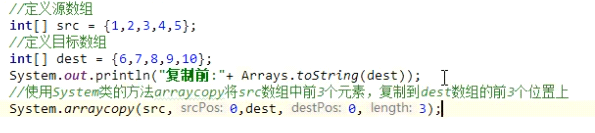
# system类



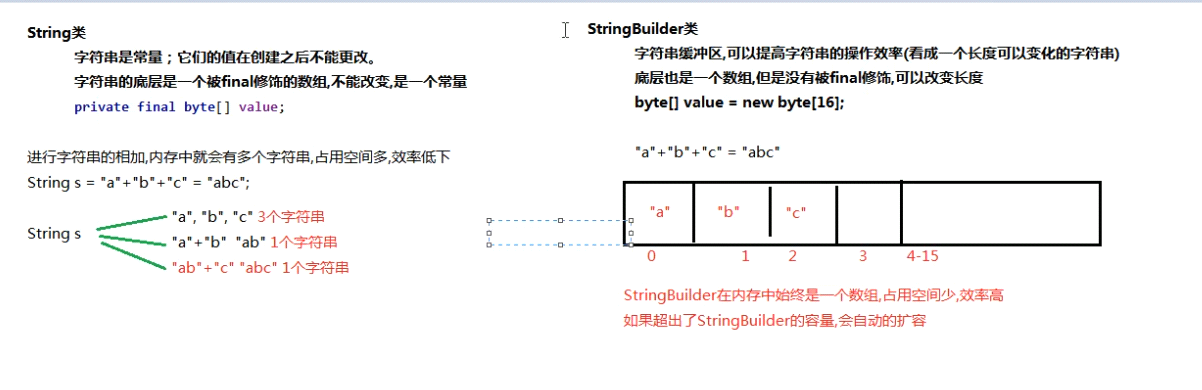




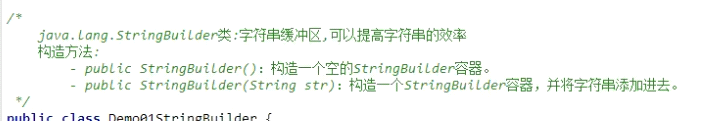
做法就是：

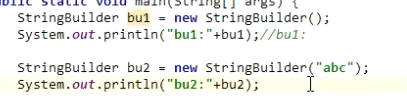


#string builder类



构造方法：

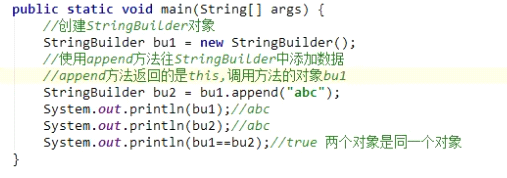




输出：

成员方法：





这个方法是引用传递，所以其实不需要用一个容器变量来接受它，直接用就可以。

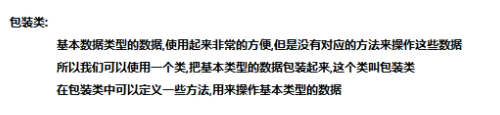
Like this：



输出：

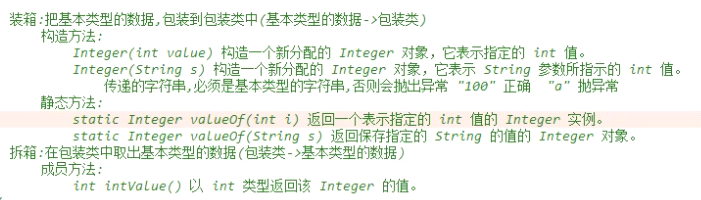
当然了，str和sb是可以互相转换的，str想变成sb，调用sb的带参构造函数即可；sb想变str，tostring（）即可，

#包装类





Integer与int：

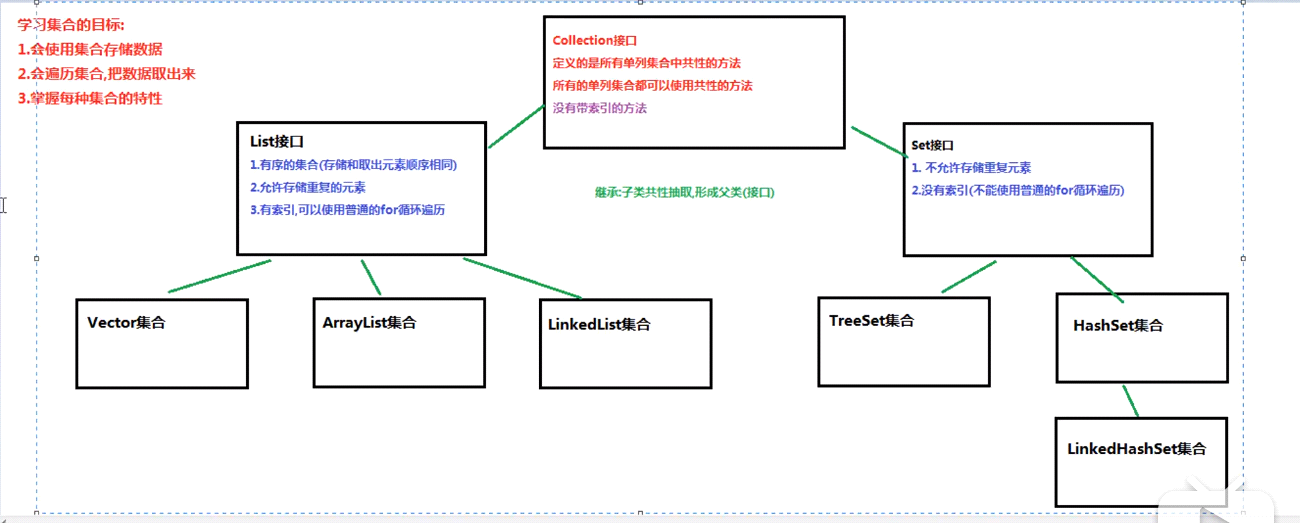


注意到，装箱就是把基础类型数据变成箱类型数据（比如Integer），这不应该是Integer类构造函数做的事情吗，为什么不用呢？因为java官方认定箱类构造函数过时了，所以初始化都用成员函数点valueof（）

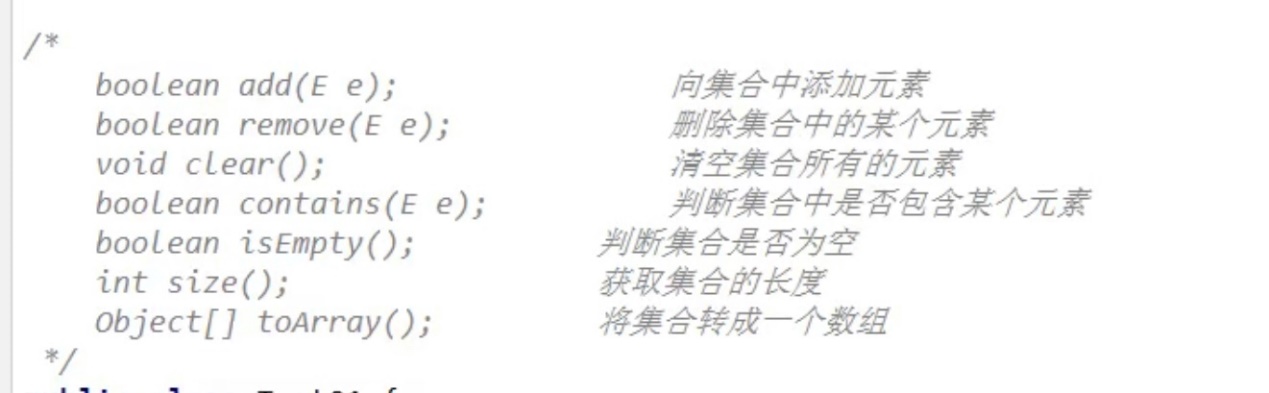
但实际上，jdk已经支持自动装箱拆箱了，所以Integer可以完全当int来用。

#collection集合

先来看看java集合的框架：

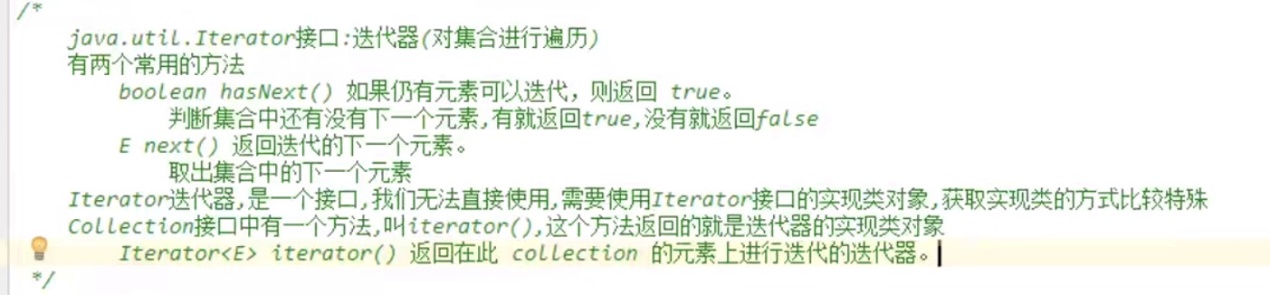


collection集合中的常用方法：（这些方法是一切集合都要实现的，务必熟悉）

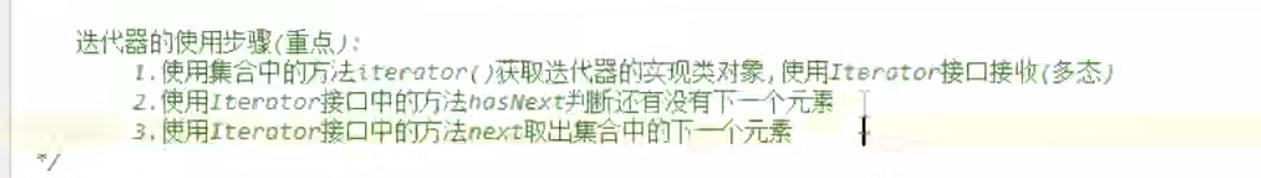


# 迭代器iterator（通用的取出集合中元素的方法。否则每个集合都要专门定义一个遍历方法，太过复杂）

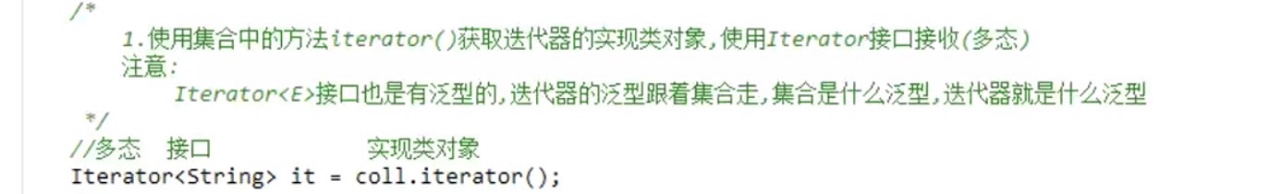
两个常用方法（iretator是个接口，所以首先要在实现类初始化，获取一个迭代器）：



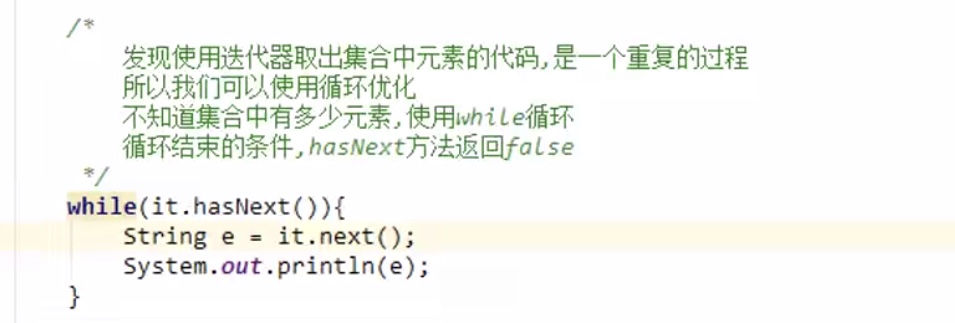
迭代器的使用步骤：



第一步，使用之前要先用实现类（几乎所有的类都实现了iterator接口）将其实体化：

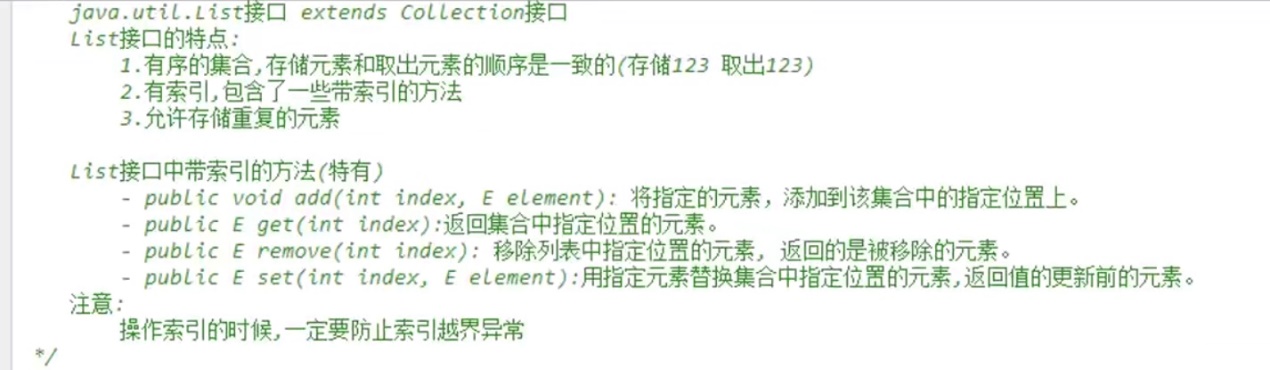


取出集合元素是一个重复的步骤，所以可以使用循环取出（可以获取集合大小，也可以直接while（）循环条件用hasnext判断）：

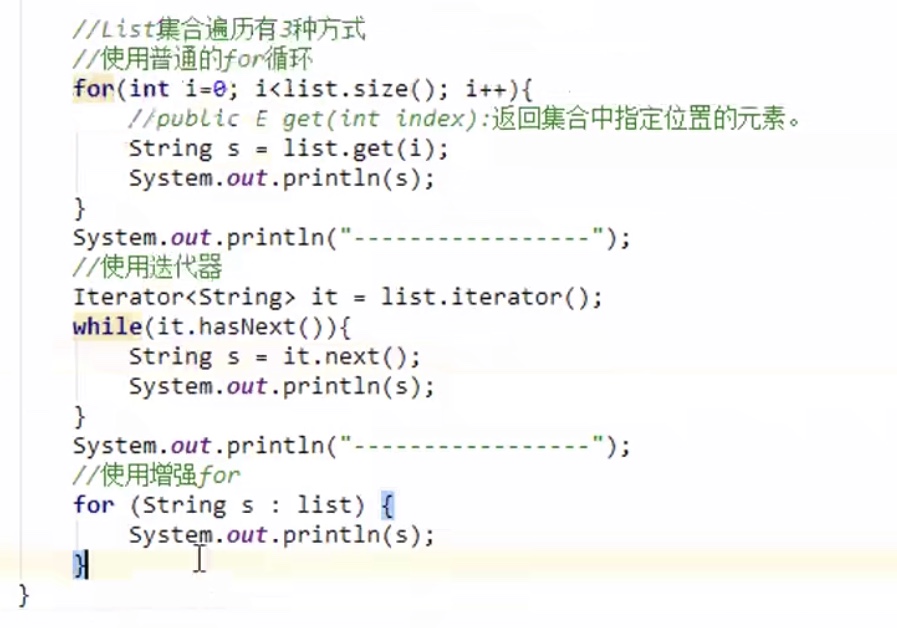


#List集合（有索引 有顺序 允许存储重复元素；实现类主要是：arrlist，linkedlist，vector）

List集合中带索引的方法（list特有）：



list集合的遍历（三种方法）：



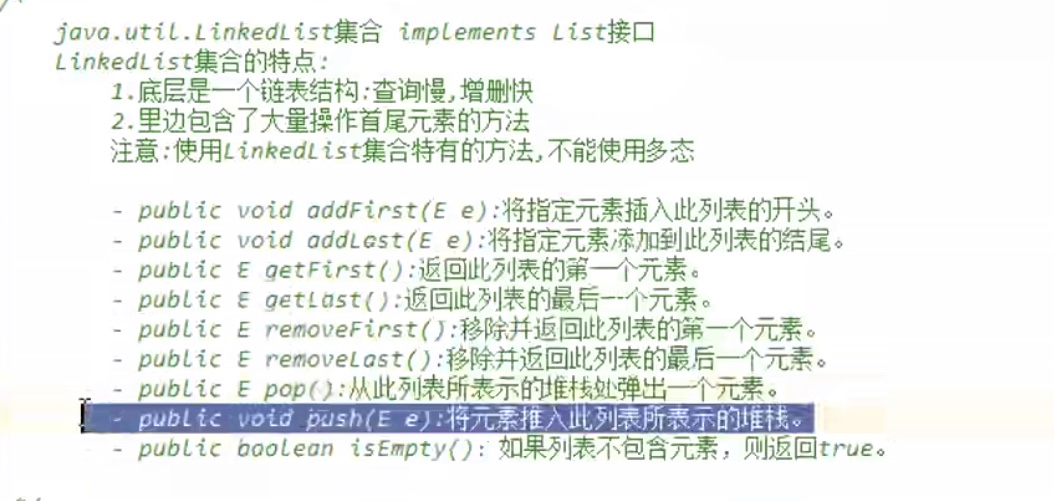
1.arraylist集合

它的底层增删元素是需要复制数组实现的，因此它的查找很快（数组地址连续），但是增删元素很慢。

2.linkedlist集合

底层是链表实现的（所以查询慢，但是增删快）。

特点及常用方法：



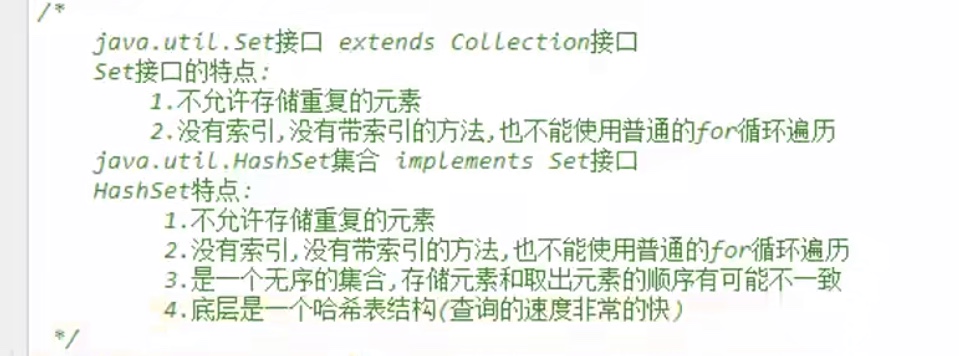
3.vector集合

早已过时，了解即可。

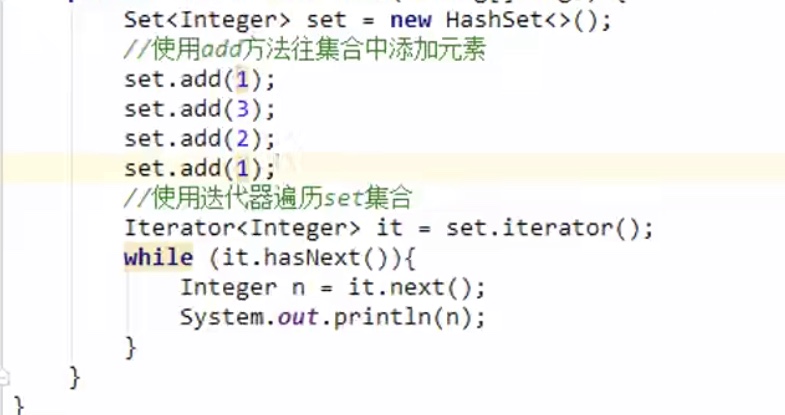
#set集合（它是没有索引的，因此它的方法基本和collection是一样的）

两个主要实现类：hashset集合和treeset集合

特点：



有重复元素，遍历可以发现：只输出123！

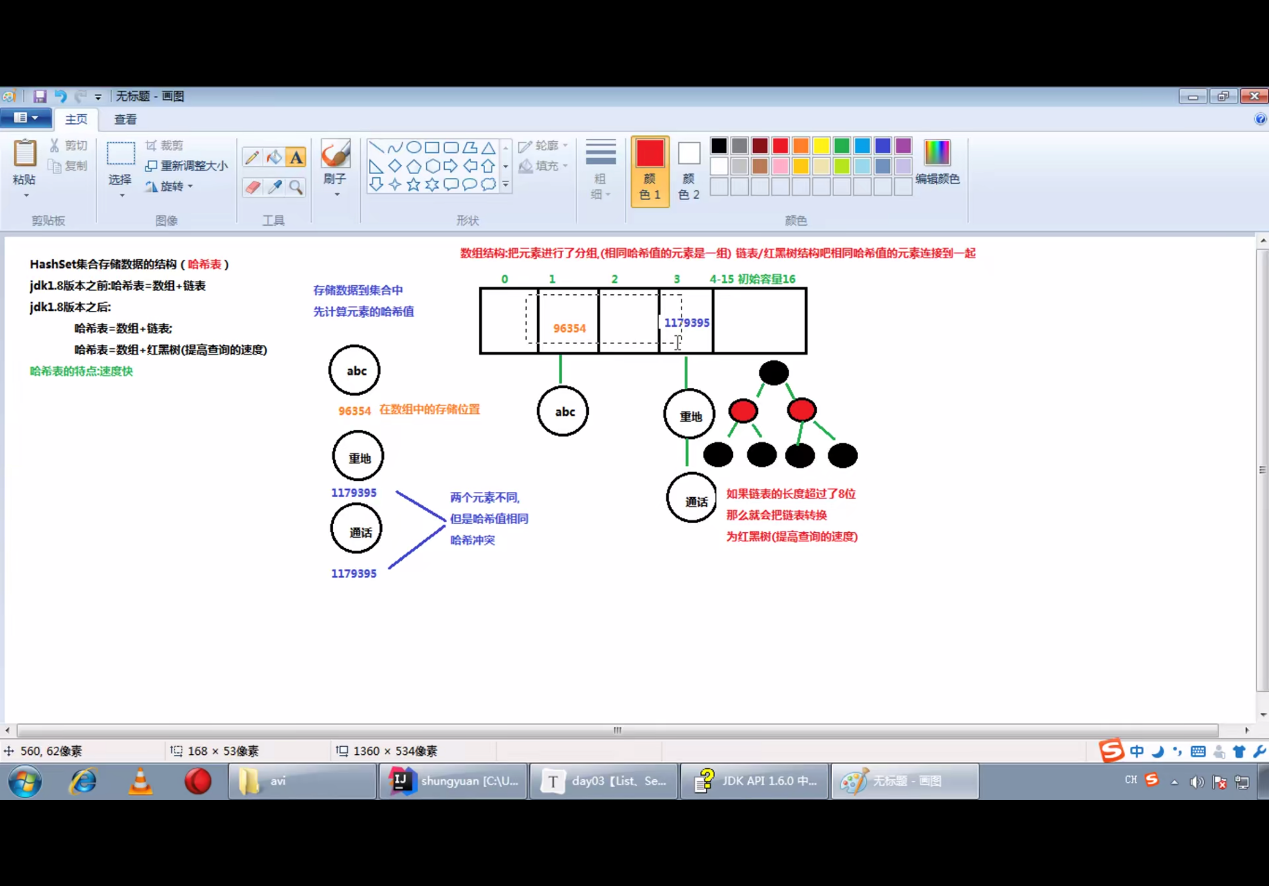


只输出123！所以说set中不能有重复元素，根本存不进去。

在学习hashset集合前，先来看看hash值是什么：

所以说默认的tostring（）中直接打印的地址就是某对象的哈希值（不是真实的地址啦）。

1.hashset集合



综上，当一个哈希值下挂着的元素超过八个，系统就会把它们变成红黑树，目的是提高查询速度。

hashset具体的创建过程：

