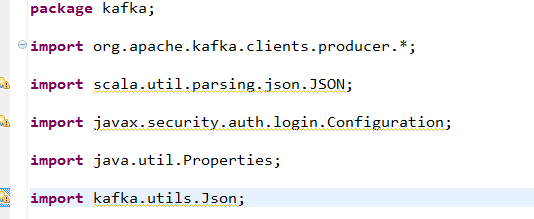
11.12

kafka生产数据



kafka中需要导入的包都是从/opt/software/kafka\_2.11-0.10.1.0/libs 中的jar包出导入 bulid path 会生成

API，即应用程序编程接口，它记录了java语言中海量的API，主要包括类的继承结构、成员变量（即属性）和成员方法（即方法）、构造方法、静态成员的详细说明和描述信息。

* **方法：一段可以被重复使用的代码块**

public static 返回值类型 方法名称（[数据类型 参数名称,…]）{

[return [返回值];]

}

* **类与对象:**

**一个类可以同时产生多个对象，对象的所有行为都受到类的控制，在类之中有两个重要的组成部分（一个类里面包括两部分：属性+方法）：**

**1.属性：每一个对象都有自己的属性内容，所谓的属性实际上就是一个变量；**

**2.方法：是公共的行为，表示某些固定的操作，如：说话**

**单独有类是不可以被使用的，所有的类都一定要通过对象产生后才可以使用**

**(即：要想用类中的属性和方法，一定要通过声明并实例化对象才行)**

* 声明并实例化对象：类名称 对象名称 = new 类名称（）
* 分布进行
  + 声明对象：类名称 对象名称 = null ；
  + 实例化对象：对象名称 = new 类名称（）

当一个类产生了实例化对象之后，就可以通过如下两种方式调用类中的属性及方法：

对象.属性：表示要访问类中的属性内容，如per.age = 20 ;

对象.方法（）：表示调用类中的方法，如per.tell() ;

* 数组：与类无关，和类一样也是引用数据类型，数组的长度是固定的，数组用来存放基本类型的数据

声明并开辟数组：

* 数据类型 数组名称[] = new 数据类型 [长度];
* 数据类型 [] 数组名称 = new 数据类型 [长度];

数组的静态初始化：

简便语法：数据类型 数组名称 []={值，值，值，…};

完整语法：数据类型 数组名称 []=new 数据类型 []{值，值，值，…};

如：int data [] = new int [3] ;

data [0] = 10 ;

data [1] = 20 ;

data [2] = 30 ;

再如：**int** a[] = {1,2,3,4,5};

System.*out*.println(a.length); ==>5 (数组长度固定！)

System.*out*.println(a[2]); ==>3

以kafka中producerDemo.java为例：

import java.util.Properties; //则可以直接使用这个api中的类

private static Producer<String, byte[]> producer;

java中ExcelToExcel 代码：

jxl-2.6.12.jar 包要放到Referenced Libraries下 解决import jxl.xxx包不存在问题

File in\_excel = new File(in\_excel\_name);

File 属于一个类 ，属于java.io.File包

对某个类产生实例化对象后，我们就能让这个对象使用属于这个类的方法和属性，即对象.方法();

数据类型分为：基本数据类型(8种)和引用数据类型（类、接口、数组）

实例化对象的数据类型是其对应的引用数据类型（某个类或数组…）

多态性：

public class **FileInputStream**

extends [InputStream](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/InputStream.html" \o "java.io 中的类)

子类**FileInputStream**继承父类[InputStream](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/InputStream.html" \o "java.io 中的类)全部的变量和方法（即子类具有父类的所有属性、成员和方法）；也就是说，任何在父类[InputStream](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/InputStream.html" \o "java.io 中的类)上的合法操作在子类**FileInputStream**上也合法，如果父类InputStream有raiseSalary()和fire()两个方法，那么子类**FileInputStream的实例化对象也可调用。**

cmd 进入windows命令行

F: 进入F盘 dir (ls) cd

javac Test.java 先编译程序（与myeclipse中代码不同 不带package xxx;这些包）

====>生成Test.class 字节码文件

java Test 解释程序（不加.class）

String in\_excel\_name = "032\_WZ产品\_数据中心\_资源库数据结构规范\_海量资源库\_V1.6.0-2018.09.27.xls";

File in\_excel = **new** File(in\_excel\_name);

System.*out*.println(in\_excel); //in\_excel是File(文件)类型，但in\_excel值等于上面赋予的值

String in\_excel\_name = "fenghuo.xls";

File in\_excel = **new** File(in\_excel\_name);

相等于：

File in\_excel = **new** File("fenghuo.xls");

in\_excel是文件类型，文件名是fenghuo.xls,文件路径默认在当前工程下

//通过Workbook的getSheet方法选择第一个工作表（从0开始）,即选择第一个(序号为0个)sheet, wb\_2\_sheet0代表第一个sheet

Sheet wb\_2\_sheet0 = wb\_2.getSheet(0);

Map<String, String> protocol\_map = new TreeMap<String, String>();

// TreeMap是其已知实现类

接口的对象 默认值为{}

# 1.java基本概念总结

java中的域：

所谓的域field，也就是我们常说的字段，或者说是属性，比如类的字段（属性，一个成员，它表示与对象或类关联的变量）局部的，全局的。

所谓的域就是变量

实例域

实例域就是在类内部的变量（非静态的），在类中有定义

如，在一个方法中，实例域初始化为某一状态，如下蓝色部分：

public Employee(String n, double s, int year, int month, int day){

name = n;

salary = s;

LocalDate hireDay = LocalDate.of(yaer, month, day);

}

这是Employee类的一个构造方法，在构造Employee类的对象时，构造器会运行，以便将实例域初始化为所希望的状态。

再如，在主方法main中创建Employee类实例时：

**new** Employee(“James Bond”, 10000, 1950, 1, 1)

将会把实例域设置为：

name = “James Bond”;

salary = 10000;

hireDay = LocalDate.of(1950, 1, 1);

总结：

* 实例域是在类刚开始时就被定义了，如private String name; private Double salary; private LocalDate hireDay;
* 在具体某一方法中，实例域是一个变量，而在main方法中调用这个方法时，就要将实例域设置成具体的常量。（在方法中，实例域都是变量形式存在）

创建对象形式：类名 对象名=new 类名() 解释

类名 对象名：这个的解释是创建名称为“对象名”的“类名”，该“对象名”是一个类引用变量

new：这个的解释就是实例化对象；

类名：这个的解释是无参数构造方法；

new 类名（）：就是说以无参数构造方法实例化对象

类名 对象名=new 类名()：用指针的解释就是以无参数构造方法实例化对象并将这个对象的内存引用赋给“类名” 类的 “对象名” 引用变量

# 2.类相关总结

## 2.1 File类

我们想要实现IO的操作，就必须知道硬盘上文件的表现形式；

而java就提供了一个类File供我们使用

File:文件和目录（文件夹）路径名的抽象表达形式

构造方法：

File(String pathname):根据一个路径得到File对象(字符串类型)

File(String parent,String child):根据一个目录和一个子文件/目录得到File对象

File(File parent,String child):根据一个父File对象和一个子文件/目录得到File对象

如果String类型的pathname没有跟上绝对路径，则写的字符串（文件名）会默认在当前工程目录下创建该文件

File ws = **new** File("1120.txt");

System.*out*.println(ws);

ws.createNewFile();

===>会在F:\Workspaces\Test 目录下生成1120.txt文件

**try** {

File ws = **new** File("f:\\1120.txt");

System.*out*.println(ws);

ws.createNewFile();

===>会在F:\ 目录下生成1120.txt文件

String targetFilePath = "F:\\Workspaces\\Test\\test.bcp";

File targetFile = **new** File(targetFilePath);

System.*out*.println(targetFile.getParentFile().exists());

**if** (targetFile.getParentFile().exists() == **false**) {

targetFile.getParentFile().mkdirs(); //mkdir()只建立最下级一层的文件夹，如果各级文件夹均不存在的话，mkdirs()则是建立多级文件夹，

}

===另一种写法：

**if** (!targetFile.getParentFile().exists()) {

}

**if（为false），则。。。 没有else也可以！！**

### 2.1.1 FileInputStream与FileOutStream类

**使用FileInputStream类向文件中将内容读出来、使用FileOutStream类写入数据，**但都存在一点不足，即这两类都只提供了对字节或字节数组的读取方法。（而汉字在文件如txt中占用两个字节，如果使用字节流，读取不好可能会出现乱码现象）

FileOutputStream方法：

**[write](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/FileOutputStream.html" \l "write(byte[]))**(byte[] b)：将b.length个字节从指定 byte 数组写入此文件输出流中

**[write](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/FileOutputStream.html" \l "write(int))**(int b)：将指定字节写入此文件输出流。

案例二：

File f = **new** File("ceshi.txt");

**try** {

//f.createNewFile(); //可省略，FileOutputStream可自动创建

FileOutputStream fileoutputstream = **new** FileOutputStream(f) ; //创建一个向指定File对象表示的文件（即文件f）中写入数据的文件输出流

PrintStream printstream = **new** PrintStream(fileoutputstream) ; //新建一个打印对象

System.*setOut*(printstream); //重定向屏幕输出到printstream对象中,将下面System.out.println()中输出的内容不输出到console上，而是输出到printstream中

StringBuilder sb = **new** StringBuilder();

sb.append("魏松是好人").append(" ").append("\r\n").append("hello world");

System.*out*.println(sb.toString()); //try里面的System.out.println都是输出到f文件中

解释：1.先让类File产生实例化对象f；2.创建一个向对象f中**写入数据**的文件输出流，并产生实例化对象fileoutputstream；3.从文件输出流实例化对象中新建一个打印对象printstream；4.重定向屏幕输出到printstream对象中，此时会在当前工程目录下创建一个ceshi.txt文件，**且下面所有的System.out,println()都会输入到printstream对象中；**5.接下来一系列加字符串加内容的操作都是在新建的ceshi.txt文件中写入的内容，System.*out*.println(sb.toString())表示在文本ceshi.txt中输出这些字符串内容

**try 中在创建文件、创建文件输出流、创建打印对象、重定向屏幕输出，这四步后 接下来1.代码编写写入的字符串之类都通过System.*out*.println(sb.toString()) 打印到文件中！不会在Console中显示！2.或者通过文件输入流InputStream读取某个路径下文件，将文件中内容通过**StringBuilder append方法写入到创建的文件内

* **比较FileOutputStream、InputStream与FileWriter、FileReader两种方法在文件中写入数据、读取数据的方式**

1、FileOutputStream、InputStream写与读方法：

**File file = new File("word1.txt");**

**try {**

**//在建立的文件中写入数据，共四步**

**FileOutputStream out = new FileOutputStream(file);**

**byte buy[] = "wei松".getBytes();**

**out.write(buy);**

**out.close();**

**//将文件中的数据读取出来**

**InputStream is = new FileInputStream(file);**

**byte byt[] = new byte[1024]; //InputStream的read()只支持byte[]形式读取**

**int len = is.read(byt); //数组byt的长度为len**

**System.*out*.println(new String(byt,0,len)); //从第0个读取到第len个**

**is.close(); //关闭流**

**} catch (Exception e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

2、FileWriter、FileReader写与读方法：

**File file = new File("word2.txt");**

**try {**

**//在建立的文件中写入数据，共四步**

**FileWriter fw = new FileWriter(file);**

**String s = "烽火星空X5173";**

**fw.write(s);**

**fw.close();**

**//将文件中的数据读取出来**

**FileReader fr = new FileReader(file);**

**char cha[] = new char[1024]; //FileReader的read()只支持char[]形式读取**

**int r = fr.read(cha); //数组cha[]的长度为r r=9 即：烽火星空X5173**

**System.*out*.println(new String(cha,0,r)); //从第0个读取到第r个**

**fr.close(); //关闭流**

**} catch (IOException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

3、**两者比较**

* 其中，类FileOutputStream（完整写法**java.io.FileOutputStream**）所包含的方法write(),其参数只能有**[write](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/FileOutputStream.html" \l "write(byte[]))**(byte[] b)、**[write](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/FileOutputStream.html" \l "write(int))**(int b)两种形式，字符串类型无法直接写入到文件中；类Writer（完整写法**java.io.Writer**）所包含的方法write()，其参数类型可以是**[write](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/Writer.html" \l "write(char[]))**(char[] cbuf)、**[write](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/Writer.html" \l "write(int))**(int c)、**[write](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/io/Writer.html" \l "write(java.lang.String))**([String](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html" \o "java.lang 中的类) str)三种形式。
* 类InputStream读取的read()方法只支持字节byte或字节数组byte[] 形式，类FileReader读取的read()方法只支持字符char或字符数组char[]形式
* **System.*out*.println(new String(byt,0,len));相当于**

**String ws = new String(byt,0,len);**

**System.*out*.println(ws);**

**注意：new String(byt,0,len)相当于一个实例化对象ws,只不过为了方便不先创建一个实例化对象，然后在调用这个对象本身。**

例如： String str = **new** String("weisong");

System.*out*.println(str); //则输出的是weisong

再如： String str = **new** String();

System.*out*.println(str); //则输出的是空

===>**可以直接打印输出实例化对象!!**

java.io.FileOutputStream out = **new** java.io.FileOutputStream(file);

java.io.FileWriter fw = **new** java.io.FileWriter(file);

相当于：

FileWriter fw = **new** FileWriter(file);

**使用完整写法与简写，对于声明并实例化对象 效果一样!**

如果代码中调用的类是使用完整写法，则不需要import java.io.FileWriter

byte字节型 一个中文占两个字节， 但一个中文算一个字符

### 2.1.2 FileReader与FileWriter类

FileReader与FileWriter类采用字符流，读取带中文内容不会出现乱码现象。

**FileReader和FileWriter字符流对应了FileInputStream和FileOutputStream类。**FileReader流顺序地读取文件，只要不关闭流，每次调用read()方法就顺序地读取源中其余的内容，直到源的末尾或流被关闭。

FileReader流：xxx.read(s); xxx.close(); 读文件，关闭流(xxx为实例化对象，s为变量)

FileWriter流：xxx.write(s); xxx.close(); 写入文件，关闭流

例：建某文件并向其中写入一些字符串

**try** {

targetFile.createNewFile();

FileWriter fw = **new** FileWriter(targetFilePath);

String s = "烽火星空X5173";

fw.write(s);

fw.close();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

## 2.2 String类

如果检查字符串是否为空，通过String自带方法isEmpty()

如果字符串为空，则返回True

**if** (*isCNChar*(str1) == **false** && str1.isEmpty() == **false**) {

…

}**else**{

**continue**; //否则跳过本次循环，不是跳出循环

}

类String自带的方法：

trim():忽略（不计）前导空格和尾部空格

indexOf(String s):返回的是搜索的字符或字符串首次出现的位置

charAt(int index):可将指定索引处的字符返回

substring(int beginIndex, int endIndex):从字符串某一索引位置开始截取至某一索引位置结束的子串

replace(char oldChar,char newChar):将指定的字符或字符串替换成新的字符或字符串

length():得到的是字符串char字符的长度（一个中文汉字占两个字节，不管是中文还是英文一个字就是一个长度），而不是字节数

valueof(int i): 返回 int 参数的字符串表示形式。return String.valueof(i);

String s1 = “123”; //**这也是创建实例化对象s1，其值等于”123”，对象是字符串类型**

### 1.2.1 StringBuilder类和StringBuffer类

String类有一个重要的特点，那就是String的值是不可变的，这就导致**每次对String的操作都会生成新的String对象，**不仅效率低下，还会浪费有限的内存空间，那么对于经常要改变值的字符串应该怎样操作呢？

答案就是使用StringBuffer和StringBuilder类，这两个类功能（和带的方法）基本一样，区别主要在于StringBuffer类的方法是多线程安全的，而StringBuilder不是线程安全的，相比而言StringBuilder类会略快一些。

在 StringBuilder 上的主要操作是 append 和 insert 方法，可重载这些方法，以接受任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将该字符串的字符追加或插入到字符串生成器中。append 方法始终将这些字符添加到生成器的末端；而 insert 方法则在指定的点添加字符。

--简单的说，让StringBuilder类实例化的对象通过append()方法在文件中追加字符串形式的内容

最后，toString()方法返回类StringBuilder实例化对象全部追加的字符串

StringBuilder sb = **new** StringBuilder();

StringBuffer类常用方法和StringBuilder类一样，也是append()、toString()

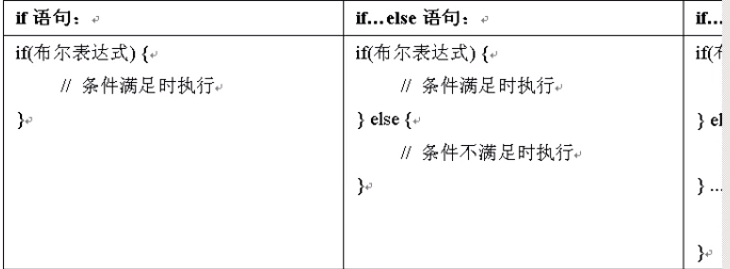
StringBuffer buffer = **new** StringBuffer();

------------------------------------------------------------------

# 3.循环相关总结

有三种：顺序、分支、循环

1.分支语句if (**括号里是true，才往下执行**)



**if** (str2.equals("string")) {

str2 = "y\_string\_id";

}**else** **if** (str2.equals("int")) {

str2 = "y\_tint\_id";

}**else** **if** (str2.equals("double")) {

str2 = "y\_double\_id";

}**else** **if** (str2.equals("tlong")) {

str2 = "y\_tlong\_id";

}**else** **if** (str2.equals("long")) {

str2 = "y\_tlong\_id";

}**else** **if** (str2.equals("geopoint")) {

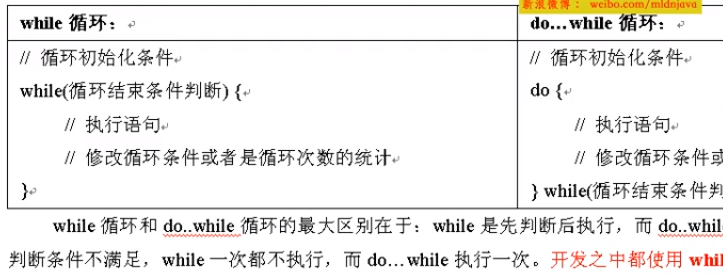
str2 = "y\_geopoint\_idm";

}

3.循环语句

主要有两类：while、for

开发中都使用while，正常人都不用do while



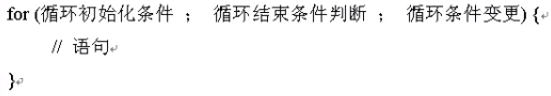
while(或者说是满足条件即进入循环体，也是条件为true才进行循环) {

//循环体，执行语句

}

注意；一定要在while循环体内设法改变循环条件为false或者使用break语句来结束循环的执行，否则将形成死循环。

for 循环



提示：关于循环的选择问题（个人总结，99%使用）

* 如果在已经明确知道循环次数的情况下，一定使用for循环；
* 如果不知道循环次数，但是知道循环结束条件，一定使用while循环；

注意：

* **for循环执行顺序：①--②--④--③ （最后执行i++，若③不满足②的条件，则最后一次执行的③不会进入到循环体④中）**
* **如果初始表达式数据类型已经被定义，则不需要在初始表达式重新加上定义的数据类型，即 int String 这种不需要在加上**

for循环第二种方式

for(循环变量类型 循环变量名称:要被遍历的对象) 循环体

这种语法，主要用于遍历数组的操作

例:

int[] integers = {1,2,3,4};

for (int i : integers){

System.out.println(i);

}

# 4.集合类（接口）总结

java.util包提供了一些集合类，集合类与数组的不同之处是，数组的长度是固定的（int data [] = new int [3]），集合的长度是可变的；数组用来存放基本类型的数据，集合用来存放对象的引用。常用的集合有List集合、Set结合和Map集合。

## 4.1 List集合

list集合包括list接口以及list接口的所有实现类。list集合中的元素允许重复，各元素的顺序就是对象插入的顺序。类似java数组，用于可通过使用索引（元素在集合中的位置）来访问集合中的元素。其中List与Set继承了Collection接口。

List接口的常用实现类有ArrayList与LinkedList。即代码中：

List<String> list = new ArrayList<>();

List<String> list = new LinkedList<>();

ArrayList类实现了可变的数组，允许保存所有元素，包括null，并可根据索引位置对集合进行快速的随机访问；缺点是向指定的索引位置插入对象或删除对象的速度较慢。

Collection接口的常用方法（List接口同样适用）

add()方法：将指定的**对象**添加到该集合中

remove()方法：将指定的对象从该集合中移除

isEmpty()方法：返回boolean值，用于判断当前集合是否为空

List.get(i):得到指定序号 i的对象值

案例1：

List<String> list = **new** ArrayList<String>();

list.add("山东省");

list.add("江苏省");

list.add("浙江省");

list.add("福建省");

list.add("广东省");

list.add("安徽省");

list.add("河南省");

list.add("河北省");

list.add("辽宁省");

list.add("吉林省");

Random rand = **new** Random();

**int** i = rand.nextInt(10); //随机返回0到10(不包括10)的数值 即0,1,2,3,4,5,6,7,8,9

**return** list.get(i);

案例2：遍历接口中的每一个对象

Iterator<String> it = list.iterator(); //遍历，创建迭代器

**while** (it.hasNext()){ //判断是否有下一个元素

String str = (String)it.next(); //获取集合中元素

System.*out*.println(str);

遍历方法二：

**for** (**int** i = 0; i < list.size(); i++){ //遍历集合中每一个对象

System.*out*.println(list.get(i));

}

## 4.2 Map集合

Map集合没有继承Collection接口，其提供的是Key到value的映射。Map中不能包含相同的key，每个key只能映射一个value。

3.1.1 Map接口的实现类

Map接口常用的实现类有HashMap和TreeMap。**建议使用HashMap类实现Map集合**，因为由HashMap类实现的Map集合添加和删除映射关系效率更高。

List、Set和Map的区别

List：是存储单列数据的集合，存储的数据是有序并且是可以重复的

Map：存储双列数据的集合，通过键值对存储数据，存储的数据是无序的，**Key值不能重复，value值可以重复，每个键最多只能映射一个值**

Set接口等同于Collection接口，不过其方法的行为有更严谨的定义。集（set）的add方法不允许增加重复的元素。set集合遍历时是无序的，与插入Set集合的顺序并不相同。

list与set不同之处是set可以除掉重复元素且遍历时是按照无序遍历的，而list是有序且可以有重复元素。

## 4.3集合（核心技术第九章）

在实现方法时，选择不同的数据结构会导致其实现风格以及性能存在很大差异。需要快速地搜索成千上万个（甚至上百万）有序的数据项吗？需要快速地在有序的序列中间插入元素或删除元素吗？需要建立键与值的关联吗？这些都需要我们利用java中集合类库来实现传统的数据结构！

集是一个集合，它可以快速地查找现有的元素。但是，要查看一个元素，需要有要查找元素的精确副本。这不是一种非常通用的查找方式。通常，我们知道某些键的信息，并想要查找与之对应的元素。映射（map）数据结构就是为此设计的。映射是用来存放键/值对，如果提供了键，就能查找到值。

### 4.3.1 Collection接口

在java类库中，集合类的基本接口是Collection接口。这个接口有两个基本方法：

public interface Colliection<E> {

Boolean add(E element);

Iterator<E> iterator();

}

add方法用于向集合中添加元素。如果添加元素确实改变了集合就返回true，如果集合没有发生变化就返回false。（因此collection集合（也就是set集合）中不允许有重复的对象）。

iterator方法用于返回一个实现了Iterator接口的对象。可以使用这个迭代器对象依次遍历集合中的元素。

* 提示：E表示泛型，比如在ArrayList中E只能是String；E-Element（在集合中使用，因为集合中存放的是元素）、T-Type（java类）、K-Key（键）、V-Value（值）、N-Number（数值类型）、？-表示不确定的java类型

### 4.3.2 迭代器

Iterator接口包含4个方法：

public interface Iterator<E> {

E next();

Boolean hasNext();

void remove();

default void forEachRemaining(Consumer<? super E> action); //jdk1.8新增

}

通过反复调用next方法，可以逐个访问集合中的每个元素。但是，如果达到了集合的末尾，next方法将抛出一个NoSuchElementException。因此，需要在调用next之前调用hasNext方法。如果迭代器对象还有多个供访问的元素，这个方法就返回true。如果想要查看集合中的所有元素，就请求一个迭代器，并在hasNext返回true时反复地调用next方法。例如：

Collection<String> c = …;

Iterator<String> iter = c.iterator();

while (iter.hasNext()) {

String element = iter.next();

do something with element

}

用“for each”循环可以更加简练地表示同样的循环操作：

for (String element : c) {

do something with element

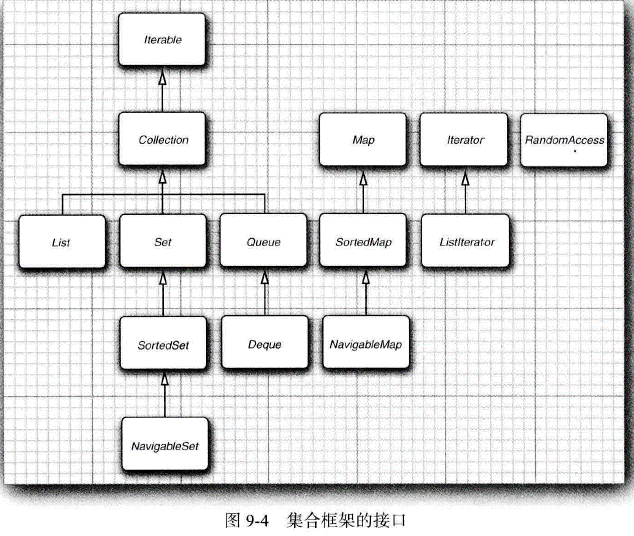
}

在java SE 8 (jdk1.8)中，甚至不用写循环，可以调用forEachRemaining方法并提供一个lambda表达式（它会处理一个元素）。将对迭代器的每一个元素调用这个lambda表达式，直到再没有元素为止。

iterator.forEachRemaining(element -> do something with element);

### 4.3.3 集合框架中的接口

Java集合框架为不同类型的集合定义了大量接口，如下图所示：



集合有两个基本接口：Collection和Map。我们已经看到，可以用以下方法在集合中插入元素：

Boolean add(E element)

不过，由于映射包含键/值对（Map），所以要用put方法来插入：

V put(K key, V value)

要从集合读取元素，可以用迭代器访问元素。不过，从映射中读取值则要使用get方法：

V get(K key)

List是一个有序集合。元素会增加到容器中的特定位置。可以采用两种方式访问元素：1、使用迭代器（必须顺序访问元素）；2、使用一个整数索引来访问（get(i)）。后一种方法称为随机访问，因为这样可以按任意顺序访问元素。

List接口定义了多个用于随机访问的方法：

void add(int index, E element)

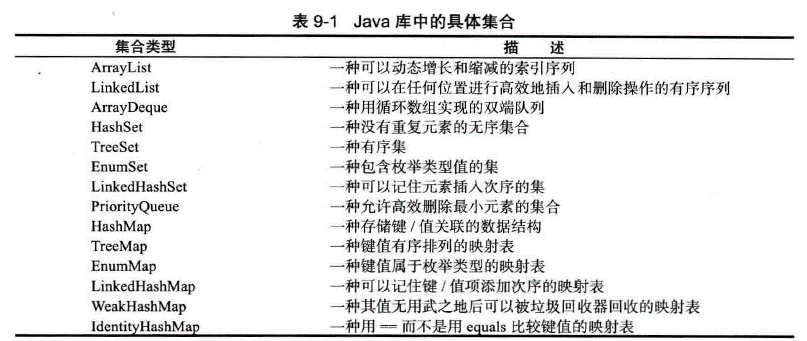
void remove(int index)

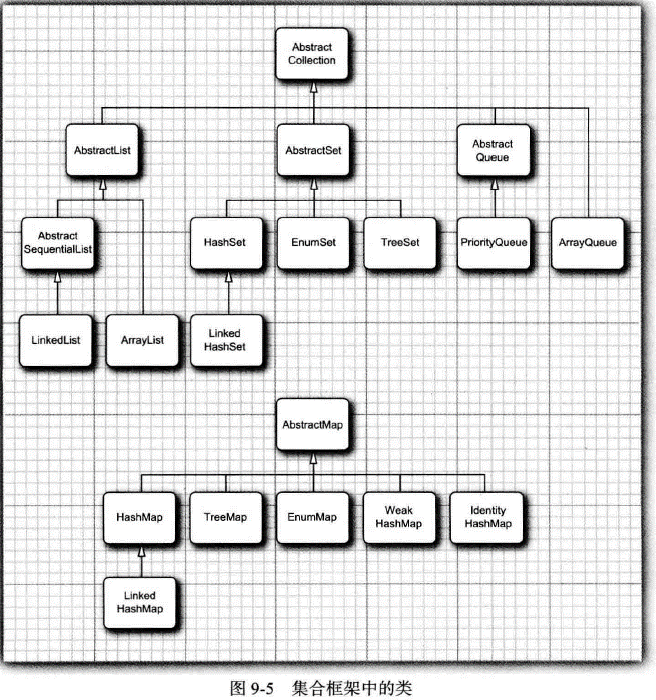
E get(int index)

E set(int index, E element)

Set接口等同于Collection接口，不过其方法的行为有更严谨的定义。集（set）的add方法不允许增加重复的元素。

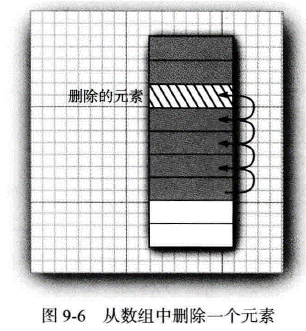
### 4.3.4 具体的集合





### 4.3.5 链表(linked list)

在本书中，有很多示例已经使用了数组以及动态的ArrayList类。然而，数组和数组列表都有一个重大的缺陷。这就是从数组的中间位置删除一个元素要付出很大的代价。其原因是**数组中处于被删除元素之后的所有元素都要向数组的前端移动**（见下图）。在数组的中间位置插入一个元素也是如此。



引入链表（linked list）就是解决这个问题。

java集合类库提供一个类LinkedList

List<String> staff = **new** ArrayList<String>();

staff.add("AAA");

staff.add("BBB");

staff.add("CCC");

Iterator iter = staff.iterator();

String first = (String) iter.next();

iter.remove();

String second = (String) iter.next();

System.*out*.println(staff.get(0));

iter.remove()将第一个对象元素删除后，则**所有元素都要向数组的前端移动，这时候查staff.get(0)变成了第二个元素**

* 散列（Hash）表/哈希表

如果我们想查看某个指定的元素，却忘记了它的位置，传统做法是访问所有元素，直到找到为止。而通过散列表（一种数据结构），可以快速地查找到所需的对象，在java中**散列表是通过链表数组实现**。

* 树集

TreeSet类与散列集十分类似，不过，它比散列集有所改进。树集是一个有序集合。可以以任意顺序将元素插入到集合中。在对集合进行遍历时，每个值将自动地按照排序后的顺序呈现。

### 4.3.6 映射

集是一个集合，它可以快速地查找现有的元素。但是，要查看一个元素，需要有要查找元素的精确副本。这不是一种非常通用的查找方式。通常，我们知道某些键的信息，并想要查找与之对应的元素。映射（map）数据结构就是为此设计的。映射是用来存放键/值对，如果提供了键，就能查找到值。

例如，有一张关于员工信息的记录表，键为员工ID，值为Employee对象。

1、基本映射操作

java类库为映射提供了两个通用的实现类：HashMap和TreeMap。这两个类都实现了Map接口。

散列映射对键进行散列。树映射用键的整体顺序对元素进行排序，并将其组织成搜索树。如果不需要按照排列顺序访问键，就最好选择散列HashMap。

* 提示：键key值不可以重复，value值可以重复

Map<String, HelloWorld> staff = **new** HashMap<String, HelloWorld>();

staff.put("123", **new** HelloWorld("weisong")); //返回value值的类型是HelloWorld 不是字符串

System.*out*.println(staff);

Map<String, String> staff1 = **new** HashMap<String, String>();

staff1.put("123", **new** String("weisong")); //返回value值的类型是字符串String

System.*out*.println(staff1);

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

map.put("NB\_TAB\_INPUTMETHOD", 1); //返回value值的类型是Integer型

System.*out*.println(map);

{123=com.fenghuo.test.HelloWorld@15db9742}

{123=weisong}

{NB\_TAB\_INPUTMETHOD=1}

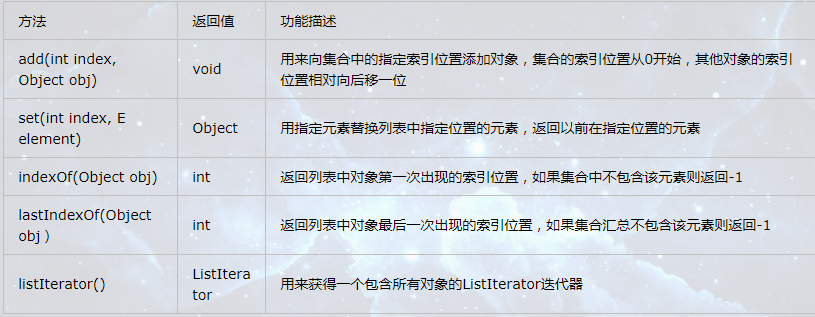
* 提示：String、Interger都属于引用数据类型，是一个类

Map<String, Integer> 第一个String代表键的数据类型，第二个Integer代表value值的数据类型

### 4.3.7 集合实例\*\*\*\*\*

1.List接口

List是列表类型，以线性方式存储对象，自身的方法都与索引有关，个别常用方法如下。



下面举一个实例，看看如何创建并添加修改集合元素。

String aString = "A", bString = "B", cString = "C", dString = "D", eString = "E";

List<String> list = **new** ArrayList<String>();

list.add(aString);

list.add(bString);

list.add(cString);

Iterator<String> iter = list.iterator(); // 创建集合迭代器

**while** (iter.hasNext()) {

System.*out*.print(iter.next() + " ");

}

System.*out*.println(); // 换行

list.set(1, eString);

Iterator<String> it = list.iterator();

**while** (it.hasNext()) {

System.*out*.print(it.next() + " ");

}

上述代码中，add()方法用于向集合中添加元素，set()方法用于修改集合中的元素，迭代器用于遍历集合中的元素并输出（会在下面的内容中涉及）。运行结果如下：



其中创建List集合对象时，“<>”中是集合汇总元素的类型，如上方的String表示集合中的元素由String字符串构成。因为List是一个接口，所以new的是接口的实现类，在Eclipse中光标放在List上按Ctrl+T就可以看见List接口的所有实现类了。

ArrayList和LinkedList区别：

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

对于随机访问get和set，ArrayList优于LinkedList，因为LinkedList要移动指针；

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

**总结：其实他们的功能相仿，只是时间快慢问题，如果不考虑返回时间问题或者数据量很小时，无脑用ArrayList，只有当数据量很大且需要大量插入或删除数据时，才选用LinkedList。**

2. Set接口

Set接口常用方法如下：



下面看一个例子，用addAll()方法把List集合对象存入到Set集合中并除掉重复值。

List<String> list = **new** ArrayList<String>();

list.add("dog");

list.add("cat");

list.add("fish");

list.add("dog"); //重复值

Set<String> set = **new** HashSet<String>();

set.addAll(list); //将list集合对象添加到set集合中

Iterator<String> it = set.iterator();

**while**(it.hasNext()){

System.*out*.print(it.next() + " ");

同理，创建Set对象集合时，Set是一个接口，new的是接口的实现类。运行结果如下：



由于Set集合中的对象是无序的，遍历Set集合的结果与插入Set集合的顺序并不相同。

3. Map接口

Map接口提供了将键映射到值的对象，一个映射不能包含重复的键，每个键最多只能映射一个值。Map接口同样提供了clear()、isEmpty()、size()等方法，还有一些常用方法如下。

|  |  |  |
| --- | --- | --- |
| 方法 | 返回值 | 功能描述 |
| put(key k,value v) | Object | 向集合中添加指定的key与value的映射关系 |
| get(Object key) | boolean | 如果存在指定的键对象，则返回该对象对应的值，否则返回null |
| values() | Collection | 返回该集合中所有值对象形成的Collection集合 |

下面看一个例子：

Map<String, String> map = **new** HashMap<String, String>();

map.put("1", "dog");

map.put("2", "cat");

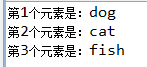
map.put("3", "fish");

**for**(**int** i = 1; i<=map.size(); i++){

System.*out*.println("第" + i + "个元素是：" + map.get(i + ""));

这里面要注意的是：1、map.get(Object key) 括号内是键的值，由上代码可知，键值类型是字符串型，因此括号内必须是字符串类型；由整型转换成字符串型只要加上+ “”；2、**i + ”” 只是代表字符串“1” “2” “3”，如果put中键值是 “10” “20” “30” ，则get中要是i + “0” （相当于“10”、“20”、“30”），要与上面键值对应！！**

运行结果如下：



创建Map集合时，Map接口的“<>”中含有两个类型，分别对应其key与value。

### 4.3.8总结

* **List接口：主要包括ArrayList()和LinkedList()两个类，其实他们的功能相仿，只是时间快慢问题，如果不考虑返回时间问题或者数据量很小时，无脑用ArrayList，只有当数据量很大且需要大量插入或删除数据时，才选用LinkedList。常用方法：add(E e)**向列表的尾部添加指定的元素，**add(int index, [E](mk:@MSITStore:D:\\+java相关\\JDK_API_1_6_zh_CN.CHM::/java/util/List.html" \o "List 中的类型参数) element)** 在列表的指定索引位置插入指定元素**添加元素**
* **Set接口：主要使用HashSet( )类，常用方法addAll(Collection col)将参数集合中的所有元素添加到集合的尾部**
* **Map接口：主要包括HashMap()和TreeMap()类，**存储双列数据的集合，通过键值对存储数据，存储的数据是无序的，**Key值不能重复，value值可以重复，每个键最多只能映射一个值, 建议使用HashMap类实现Map集合**，因为由HashMap类实现的Map集合添加和删除映射关系效率更高。常用方法：put(key k,value v)
* 三者的区别：**list是有序且可以有重复元素；set可以除掉重复元素且遍历时是按照无序遍历的；List和Set接口都是存储单列数据的集合，Map是存储双列数据的集合。**

# 5.接口相关总结

接口：**内部只存在有全局常量和抽象方法(接口只提供抽象方法，具体方法由接口的实现类重写)，接口必须有子类**

接口的子类使用implements关键字进行接口实现，在代码中**接口实现类必须要覆写接口之中的全部抽象方法。**

接口（interface）技术，主要用来描述类具有什么功能，而并不给出每个功能的具体实现。

接口中所有的抽象方法都自动地是public

java接口中只能包含public,static,final类型的成员变量和public,abstract类型的成员方法（没有方法体，不能被实例化）

接口定义的一般形式为：

[访问控制符]interface <接口名> {

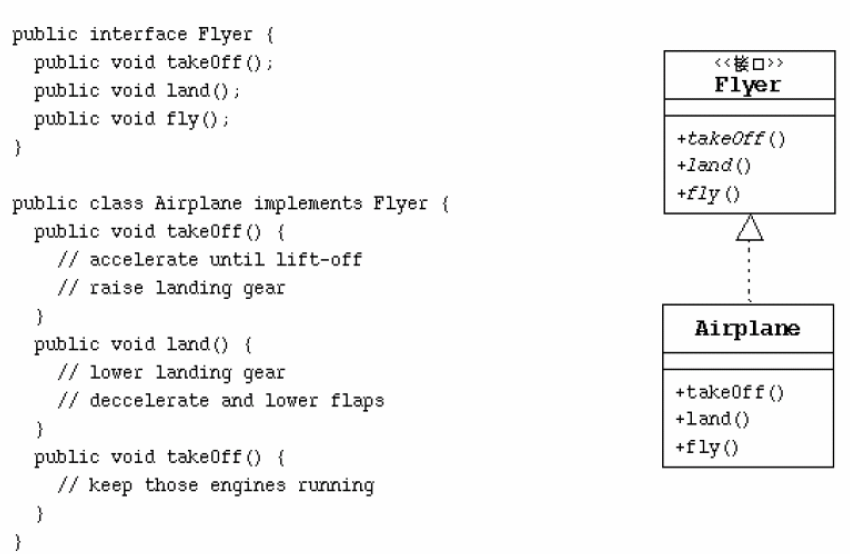
类型标识符final 符号常量名n = 常数；

返回值类型 方法名([参数列表]);

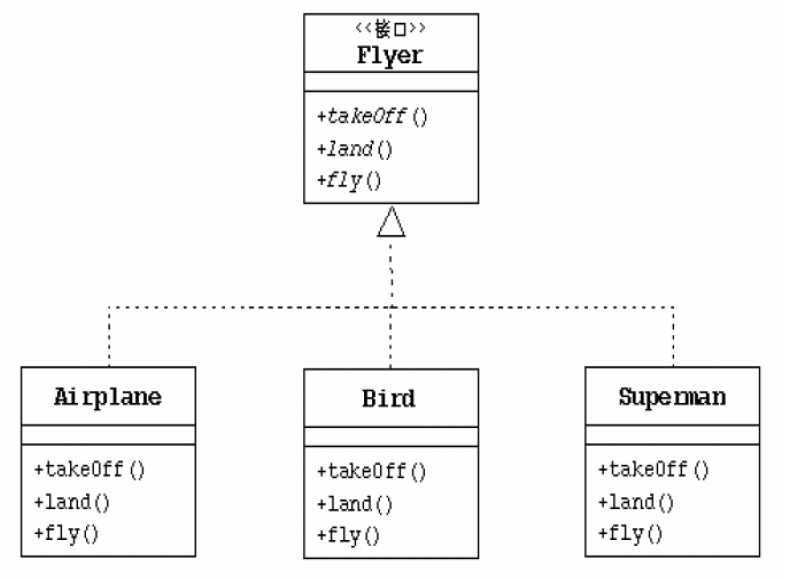
…

}

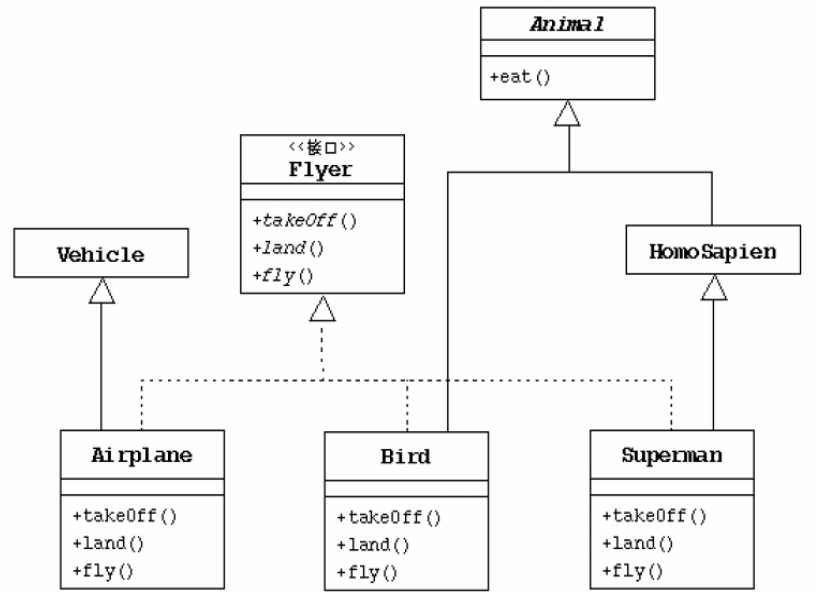
## 1.接口示例1：接口定义，接口实现类

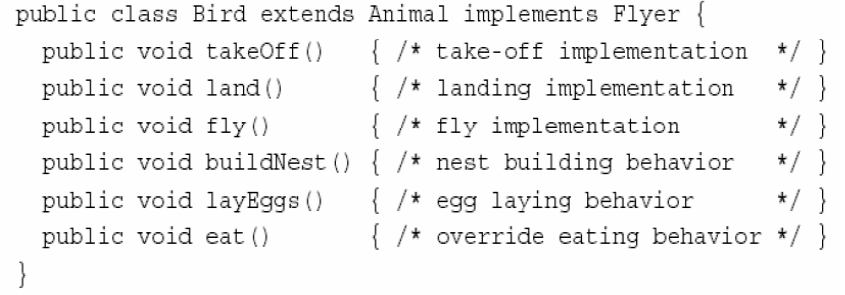


**接口示例2：一个接口可以有多个不同的实现类**



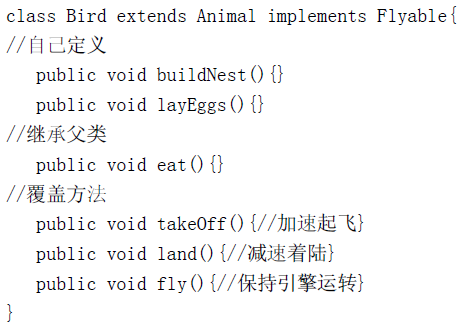
**接口示例3：一个类可以继承（extends父类），也可以实现（覆盖接口的方法）**





## 2. extends和implements

注意extends 从句放在implements 从句之前，这样一个类既具有自己定义的方法，继承父类方法，同时具有覆盖的接口方法。



## 3.接口多继承

事实上，Java的接口是可以实现多继承的（一个接口可以继承多个其他接口），类不允许多继承。

**public interface** Test **extends** A,B {

//定义

}

**interface** A{

//定义

}

**interface** B{

//定义

}

## 4.接口的基本思想

接口及相关机制的最基本作用在于：通过接口可以实现不相关类的相同行为，而不需考虑这些类之间的层次关系（因为接口只提供抽象方法，这些不相关的类都会重新覆写这些抽象方法）。根据接口可以了解对象的交互界面，而不需了解对象所属的类。

面向对象程序设计讲究“**提高内聚，降低耦合**”，那么**不同的程序模块怎么相互访问呢，就是通过接口，也就是接口是各部分对外的统一外观。**接口在Java程序设计中体现的思想就是隔离，因为接口只是描述一个统一的行为，所以开发人员在面向接口编程时并不关心具体的实现。

在接口和抽象类的选择上，必须遵守这样一个原则：行为模型应该总是通过接口而不是抽象

类定义。所以通常是：

（1）：**优先选用接口，尽量少用抽象类**。

选择抽象类的时候通常是如下情况：

（2）：需要定义子类的行为，又要为子类提供共性的功能。

java中常说的写一个接口，提供一个接口？

提供一个接口，首先要符合接口规范（接口参数、接口地址、参数类型、返回值及其类型等，也包括自己在接口里写的实现类），这样用户调用了我的接口（调用我写的实现类），就会得到接口本身具有的返回值，然后用户根据需要自己进行选择。

5.接口的特点

二、接口的特点

1、Java接口中的成员变量默认都是public,static,final类型的(都可省略),必须被显示初始化,即接口中的成员变量为常量(大写,单词之间用"\_"分隔)

2、Java接口中的方法默认都是public,abstract（抽象）类型的(都可省略),没有方法体,不能被实例化

3、Java接口中只能包含public,static,final类型的成员变量和public,abstract类型的成员方法

4、接口中没有构造方法,不能被实例化

5、一个接口不能实现(implements)另一个接口,但它可以继承多个其它的接口

6、Java接口必须通过类来实现它的抽象方法

7、当类实现了某个Java接口时,它必须实现接口中的所有抽象方法,否则这个类必须声明为抽象类

8、不允许创建接口的实例(实例化),但允许定义接口类型的引用变量,该引用变量引用实现了这个接口的类的实例

9、一个类只能继承一个直接的父类,但可以实现多个接口,间接的实现了多继承.

# 6. 继承 extends

利用继承，人们可以基于已存在的类构造一个新类。继承已存在的类就是复用（继承）这些类的方法和域（变量）。在此基础上，还可以添加一些新的方法和域，以满足新的需求。这是java程序设计中的一项核心技术。

public class Manager extends Employee{

添加新的方法和域

}

其中，已存在的类（Employee）称为超类、基类或父类；新类（Manager）称为子类、派生类或孩子类。

* 提示：尽管Employee类是一个超类，但并不是因为它优于子类或者拥有比子类更多的功能。实际上恰恰相反，子类比超类拥有的功能更加丰富。

在通过扩展超类定义子类的时候，仅需要指出子类与超类的不同之处。因此在设计类的时候，应该将通用的方法放在超类中，而将具有特殊用途的方法放在子类中，这种将通用的功能放到超类的做法，在面向对象程序设计中十分普遍。

## 1.子类覆盖父类的方法

背景：超类中的有些方法对子类并不一定使用。需要在子类中重写父类的某个方法：

在子类源文件中

public double getSalary(){

return salary + bonus ;

}

错误：因为子类中getSalary方法不能够直接访问超类的私有域（private double salary）

改进：

public double getSalary(){

double baseSalary = getSalary();

return baseSalary + bonus;

}

仍然错误：因为子类Manager中也有一个getSalary方法，会无限调用自己，直到程序崩溃。为此，可用特定的关键字super解决

最终：

public double getSalary(){

double baseSalary = super.getSalary();

return baseSalary + bonus;

}

## 2.多态

即父类中的方法、属性都可以直接拿到多个子类对象中用，这就是多态

父类 obj = **new** 子类(); //obj是子类的对象，之后可以调用父类的方法、变量

**则对象ojb的对象类型是父类型，并将调用子类()的无参构造方法！**

**但是这里面要注意的是obj对象目前还只能访问到父类里定义的属性和方法（也可以调用到子类中覆写的父类的方法，这是一种多态实现，见下文有例！），如果想要访问子类中的属性和方法，必须对obj对象进行强制类型转换，把它还原成子类的类型，就可以访问子类的属性和方法了，即：**

父类 obj = **new** 子类();

子类 m = (子类)obj;

m.xxx (xxx为子类的方法)

**举例：**

**有一个Hero类，AD接口，还有一个ADHero类，（ADHero类继承了Hero类、AD接口），还有一个 LolHerol类（含main方法）**

**package** com.fenghuo.interfacetest;

**public** **class** Hero {

String name; //定义属性，表示姓名

**float** hp; //定义属性，表示血量

**float** armor; //定义属性，表示护甲

**int** movespeed; //定义属性，表示移动速度

}

**package** com.fenghuo.interfacetest;

**public** **interface** AD {

// 物理伤害

**public** **void** physicAttack();

}

**package** com.fenghuo.interfacetest;

//ADHero是接口AD的一个具体的实现类，因此要有具体方法，变量赋值可以再主类的main方法中设置

**public** **class** ADHero **extends** Hero **implements** AD {

**public** **void** physicAttack() {

System.*out*.println(**this**.name + "进行物理攻击");

}

}

**package** com.fenghuo.interfacetest;

**public** **class** LolHero {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

AD obj = **new** ADHero();

obj.physicAttack();

**if**(obj **instanceof** ADHero){

ADHero ah = (ADHero) obj;

ah.name="亚索";

ah.physicAttack();

}

/\*ADHero ah = (ADHero) obj; //要想用类ADHero中的方法，必须要强制类型转换

ah.name="亚索";

ah.physicAttack();\*/

}

}

instanceof用于判断左边检测的对象是否是右边类或接口的实例，如果是，则返回true，继续执行{..}的内容。

**上述代码中，由于对象obj属于接口AD，只能调用AD中的方法，不能调用类ADHero继承父类Hero的变量name，因此，若想调用类ADHero中的变量name，必须对对象obj进行强制类型转型！**

**再比如：**

**package** com.fenghuo.test;

**public** **class** Parent {

String name;

**int** age;

}

**package** com.fenghuo.test;

**public** **class** Child **extends** Parent{

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Parent par = **new** Child();

par.name = "烽火";

System.*out*.println(par.name + par.searchSex());

}

String sex = "男";

**public** **void** searchSex(){

System.*out*.println(sex);

}

}

**===>此时会报错，因为对象par没有searchSex()方法，因此要想用类Child中自带的方法，必须将对象par进行强制类型转型！**

**解决方法：**

**package** com.fenghuo.test;

**public** **class** Child **extends** Parent{

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Parent par = **new** Child();

par.name = "烽火";

**if**(par **instanceof** Child){

Child chi = (Child) par; //将对象par强制转型

chi.searchSex();

System.*out*.println(chi.sex);

}

System.*out*.println(par.name);

}

String sex = "男";

**public** **void** searchSex(){

System.*out*.println(sex);

}

}

**关于向上转型，通过父类引用执行子类方法时需要注意以下两点：**

**（1）无法调用到子类中存在而父类本身不存在的方法；**

**（2）可以调用到子类中覆写了父类的方法，这是一种多态实现。（但要保证方法名、参数类型及个数必须严格一致）**

**实例：**

**package** com.fenghuo.interfacetest;

**class** Father {

**public** **void** method() {

System.*out*.print("Father's method");

}

}

**class** Son **extends** Father {

**public** **void** method() {

System.*out*.println("Son's method");

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Father obj = **new** Son();

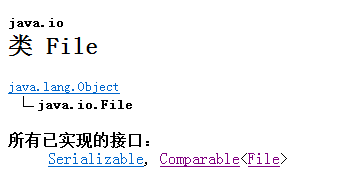
obj.method();

}

}

## 3.Object：所有类的超类

java.lang.object 是一切类的超类



3.1.Object类的toString方法

Object中有一个重要的方法，就是toString方法，它用于返回表示对象值的字符串。

## 4、强制数据类型转换instanceof

**double** x = 3.14;

**int** s = (**int**) x;

正像有时需要将浮点型数值转换成整型数值一样，有时候也可能需要将某个类的对象引用转换成另外一个类的对象引用。对象引用的转换语法与数值表达式的类型转换类似，仅需要用一对圆括号将目标类名括起来，并放置在需要转换的对象引用之前就可以了。例如：

Manager boss = (Manager) staff[1];

进行类型转换的唯一原因是：在暂时忽视对象的实际类型之后，使用对象的全部功能。

运行这个程序时，java运行时系统将报告一个错误，并产生一个ClassCastException异常。如果没有捕获这个异常，程序就会终止。因此，应该养成这样一个良好的程序设计习惯：**在进行类型转换之前，先查看一下是否能够成功地转换**。这个过程简单地使用instanceof操作符就可以实现。例如：

if (staff[1] **instanceof** Manager){

boss = (Manager) staff[1];

…

}

instanceof操作符用来确认一个对象能否成功转换成另一种数据类型；

instanceof用于判断左边检测的对象是否是右边类或接口的实例，如果是，则返回true，继续执行{..}的内容。

最后，如果这个类型转换不可能成功，编译器就不会进行这个转换。例如，下面这个类型转换：

String c = (String) staff[1];

将会产生编译错误，这是因为String不是Employee的子类。

综上所述：

* 只能在继承层次内进行类型强制转换
* 在将超类转换成子类之前，应该使用instanceof进行检查。

# 8.每日java总结

## 11.22

ExceltoBigData.java

java代码中含有传参args[0]、args[1]、args[2]…

在myeclipse中run as==>run configurations==>Arguments 多个参数以空格隔开

在linux中，javac xxx.java ==> java xxx A B C… （A、B、C为不同参数）

因为args[]数组都是String数据类型，如果我们想让某一个args[3]变成int整型（原先args[3]=1，1是字符串类型），则需要通过字符串转整型变换，即Integer.parseInt(args[3])=1，此时1是int整型

in\_type = mapping

in\_physical\_table = ydb\_physical\_table

in\_excel\_name = 032\_WZ产品\_数据中心\_资源库数据结构规范\_海量资源库\_V1.6.0\_FP-2018.09.27

in\_sheet\_index = Integer.parseInt(args[3])

out\_file\_name = weisong.txt

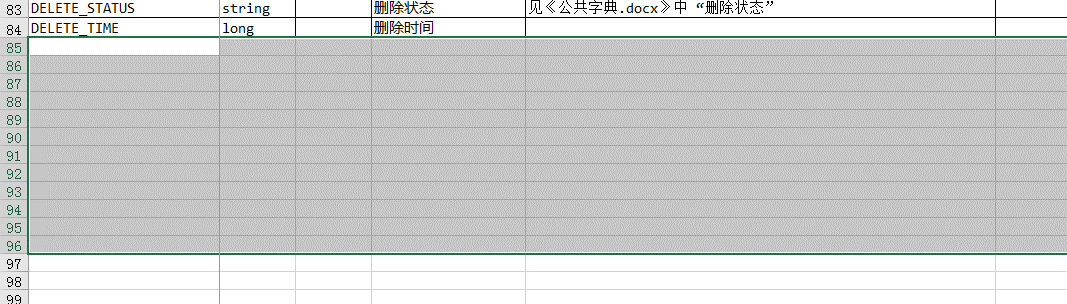
参数为：

mapping ydb\_physical\_table 032\_WZ产品\_数据中心\_资源库数据结构规范\_海量资源库\_V1.6.0\_FP-2018.09.27.xls 1 weisong.txt

sheet.getRows() : 计算总行数，即有多少行就是多少行。（但是第一行的下标是0，sheet.getCell(i,0)）

问题：若查询sheet.getRows()返回的行数超过已有的行数，说明有一些隐藏行！！！

sheet.getRows()统计的是一个sheet有多少行（空行也算）shift+ctrl+end



可以看到 这个sheet其实一共有96个有效行

可通过判断每一行的第一个cell是否为空，是空直接跳出循环

## 11.23

--产生实例化对象后就可以通过对象用该类的方法

将excel表格内容拼接到txt文本中，

问题1：保证结构对齐，字段列（30个字符），字段类型列（20个字符）

**int** field\_len = str1.length();

StringBuilder field\_space = **new** StringBuilder();

**for** (**int** k = 0; k < (30-field\_len); k++) {

field\_space.append(" ");

}

**问题2：sheet中间有某些行内容不是字段 字段类型形式，而是文字说明，应跳过这些行**

java中字符串是全英文还是中文分别判断

if continue 跳出某次循环

解决方法：

/\*\*

\* 判断字符串中是否含有中文

\*/

public static boolean isCNChar(String s){

boolean booleanValue = false;

for(int i=0; i<s.length(); i++){

char c = s.charAt(i);

if(c > 128){

booleanValue = true;

break;

}

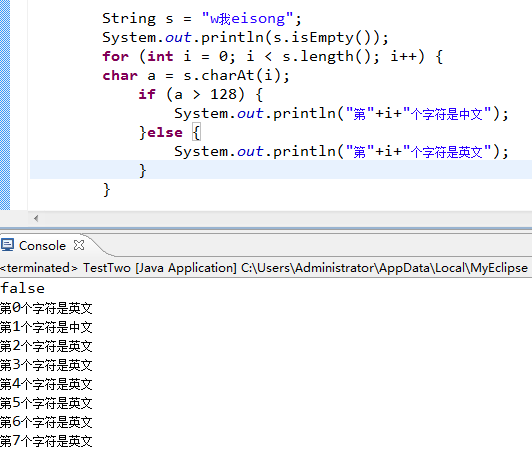
}

return booleanValue;

}

如果true，包含中文；

如果false，不包含中文



## 11.27 编译及解释程序步骤

将bcp造数据的java代码放到linux环境下，去掉代码首行的package bcp;

先编译程序，再解释程序:

javac Generatebcp.java

java Generatebcp

由于代码中所用的包都是jdk都自带lib中的包，因此不需要额外指定lib包，即

javac -Djava.ext.dirs=xxx/lib/ producerDemo.java

java -Djava.ext.dirs=xxx/lib/ producerDemo

在读取java代码中调用的包，默认从环境变量中设置的java CLASSPATH类库路径export CLASSPATH=.:$CLASSPATH:$JAVA\_HOME/lib:$JAVA\_HOME/jre/lib中读取所需的包

相当于：

javac -Djava.ext.dirs=/usr/java/jdk1.8.0\_181/lib Generatebcp.java

java -Djava.ext.dirs=/usr/java/jdk1.8.0\_181/lib Generatebcp

生成某一随机数的方法：

**public** **static** String getRandom(){

Random rand = **new** Random();

**int** i = rand.nextInt(100);

**return** String.*valueOf*(i);

}

## 11.28

给变量赋予某一随机数方法：

**int** r1 = 1000000 + **new** java.util.Random().nextInt(9000000); //nextInt方法：返回一个伪随机数，它是取自此随机数生成器序列的、在 0（包括）和指定值（不包括）之间均匀分布的 int 值。

## 11.29 int和String型互转方法

关于int型变量转成String型 ==>中间+**""**

**int** r1 = 1000000 + **new** java.util.Random().nextInt(9000000);

**int** r2 = 1000000 + **new** java.util.Random().nextInt(9000000);

**String input = r1 + "" + r2;**

r1,r2间必须加"",如果直接r1 + r2，则会报错

即：int型转为String型，多个int值之间必须要加 **+ ""**

将字符串str转换成int型形式：

int a = Integer.parseInt(str);

如果想将某个字符ch转换成int型形式：

int a = Integer.parseInt(ch + “”); //需要改写成 字符+“”==字符串 形式

“”表示一个空串

* 产生一个随机数方法：

**int** ws = **new** java.util.Random().nextInt(10);

System.*out*.println("ws=" + ws);

方法二：

Random rand = **new** Random();

**int** i = rand.nextInt(100); //随机产生[0,99]任一数值

* 在某一组数组中随机取出某一元素：

String range[]={"6","7","8","9","10","11","12"};

**int** numQQ=Integer.*parseInt*(*randomOne*(range));

System.*out*.println("numQQ = " + numQQ);

**public** **static** String randomOne(String s[]) {

**return** s[**new** Random().nextInt(s.length - 1)];

}

String result="";

result+=rand.nextInt(10)+1; //相当于result = result + (rand.nextInt(10)+1),括号里算一个字符串，整体加到result中,result最后得到的是多个值的拼接！

byte字节型 一个中文占两个字节， 但一个中文算一个字符，这就是InputStream与FileReader的区别，一个读写字节、一个读写字符

## 12.3 理解多态（向上转型）

myEclipse中，同一个包package下有多个类，在一个类T1中如果要调用其他类T2（必须要在同一个包内）中的方法，要先T2 对象 = new T2() 实例化对象，然后通过该对象.方法() 调用该方法才行！

多态的定义：父类的引用指向子类的对象

父类的引用：一是指父类变量指向的那个内存空间；

二也是指父类自由的类型信息表，这个表示任何对象都具备的，和继承多态无关；

子类的对象：一是指子类的那个堆内存空间；

二也是指子类的那个类型信息表

**对象类型的转换**

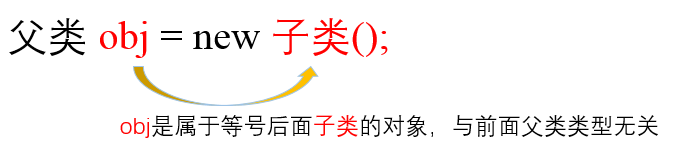
1.向上转型：将子类对象看作是父类对象被称为“向上转型”，换句话说是子类对象赋值给父类类型的变量，这种技术被称为“向上转型”；

例如：子类—平行四边形parallelogram 父类—四边形quadrangle

public class Parallelogram extends Quadrangle{

父类（四边形）的一个方法draw()，则在子类（声明继承该父类）中含有

子类 obj = **new** 子类(); 因为子类是父类中的一种，所以子类中的对象obj可以看作是父类的一个对象，这就相当于：父类 obj = **new** 子类(); 这就是“向上转型”



这样，平行四边形（子类的一种）的对象可以作为父类中draw()方法的参数，梯形（另一种子类）对象同样也可以作为draw()方法的参数。

父类（四边形类）的draw()方法中根据不同的图形对象进行不同的处理，就可以做到父类中定义一个方法完成各个子类的功能，这样可以使同一份代码毫无差别地运用到不同类型之上，这就是多态机制的基本思想。(即父类中的方法、属性都可以直接拿到多个子类对象中用，这就是多态,但是！！！对象obj无法调用子类中的属性和方法，如果该对象想调用子类中的属性和方法，必须将其强制类型转换！)

**！！！！！！！！！！！obj的引用数据类型就是父类！！！！！！！！！！！！！！！**

即父类中的方法、属性都可以直接拿到多个子类对象中用，这就是多态

父类 obj = **new** 子类(); //obj是子类的对象，之后可以调用父类的方法、变量

**则对象ojb的对象类型是父类型，并将调用子类()的无参构造方法！**

**但是这里面要注意的是obj对象目前还只能访问到父类里定义的属性和方法（也可以调用到子类中覆写的父类的方法，这是一种多态实现，见下文有例！），如果想要访问（调用）子类中的属性和方法，必须对obj对象进行强制类型转换，把它还原成子类的类型，就可以访问子类的属性和方法了，即：**

父类 obj = **new** 子类();

子类 m = (子类)obj;

m.xxx (xxx为子类的方法)

**举例：**

**有一个Hero类，AD接口，还有一个ADHero类，（ADHero类继承了Hero类、AD接口），还有一个 LolHerol类（含main方法）**

**package** com.fenghuo.interfacetest;

**public** **class** Hero {

String name; //定义属性，表示姓名

**float** hp; //定义属性，表示血量

**float** armor; //定义属性，表示护甲

**int** movespeed; //定义属性，表示移动速度

}

**package** com.fenghuo.interfacetest;

**public** **interface** AD {

// 物理伤害

**public** **void** physicAttack();

}

**package** com.fenghuo.interfacetest;

//ADHero是接口AD的一个具体的实现类，因此要有具体方法，变量赋值可以再主类的main方法中设置

**public** **class** ADHero **extends** Hero **implements** AD {

**public** **void** physicAttack() {

System.*out*.println(**this**.name + "进行物理攻击");

}

}

**package** com.fenghuo.interfacetest;

**public** **class** LolHero {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

AD obj = **new** ADHero();

obj.physicAttack();

**if**(obj **instanceof** ADHero){

ADHero ah = (ADHero) obj;

ah.name="亚索";

ah.physicAttack();

}

/\*ADHero ah = (ADHero) obj; //要想用类ADHero中的方法，必须要强制类型转换

ah.name="亚索";

ah.physicAttack();\*/

}

}

instanceof用于判断左边检测的对象是否是右边类或接口的实例，如果是，则返回true，继续执行{..}的内容。

**上述代码中，由于对象obj属于接口AD，只能调用AD中的方法，不能调用类ADHero继承父类Hero的变量name，因此，若想调用类ADHero中的变量name，必须对对象obj进行强制类型转型！**

**再比如：**

**package** com.fenghuo.test;

**public** **class** Parent {

String name;

**int** age;

}

**package** com.fenghuo.test;

**public** **class** Child **extends** Parent{

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Parent par = **new** Child();

par.name = "烽火";

System.*out*.println(par.name + par.searchSex());

}

String sex = "男";

**public** **void** searchSex(){

System.*out*.println(sex);

}

}

**===>此时会报错，因为对象par没有searchSex()方法，因此要想用类Child中自带的方法，必须将对象par进行强制类型转型！**

**解决方法：**

**package** com.fenghuo.test;

**public** **class** Child **extends** Parent{

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Parent par = **new** Child();

par.name = "烽火";

**if**(par **instanceof** Child){

Child chi = (Child) par; //将对象par强制转型

chi.searchSex();

System.*out*.println(chi.sex);

}

System.*out*.println(par.name);

}

String sex = "男";

**public** **void** searchSex(){

System.*out*.println(sex);

}

}

**关于向上转型，通过父类引用执行子类方法时需要注意以下两点：**

**（1）无法调用到子类中存在而父类本身不存在的方法；**

**（2）可以调用到子类中覆写了父类的方法，这是一种多态实现。（但要保证方法名、参数类型及个数必须严格一致）**

**实例：**

**package** com.fenghuo.interfacetest;

**class** Father {

**public** **void** method() {

System.*out*.print("Father's method");

}

}

**class** Son **extends** Father {

**public** **void** method() {

System.*out*.println("Son's method");

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

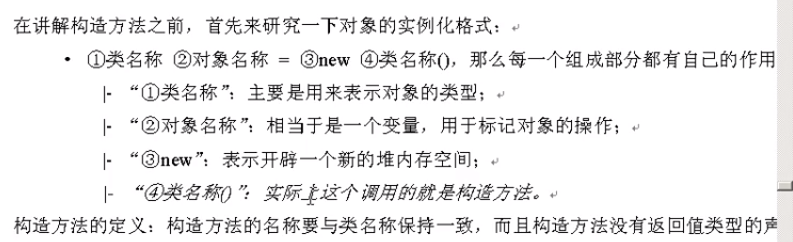
// **TODO** Auto-generated method stub

Father obj = **new** Son();

obj.method();

}

}



例如：

**package** com.fenghuo.test;

**class** People{

**public** People(){

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

**public** **class** TestJava {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

People peo = **new** People();

}

}

**重要tips**：该对象类型是最左边的People，new表示为对象peo开辟一个新的堆内存空间，**最后People()是调用类People的无参构造方法**，如果类People中没写无参构造方法（每个类默认都有一个无参构造方法），也没有影响，如果类People含有具体的无参构造方法，则实例化对象并调用无参构造方法后，将返回无参构造方法中的值。

## 12.4 理解类库、类

理解jar包

java压缩归档，即JAR文件—它允许设计者将所有必要的模块都封装到一起，供用户统一下载。

import java.util.\*; //导入完整的实用工具（Utility）库下面**所有的类（只是类）**

import java.util.Vector; //单独导入Vector类

“类”（如java.util）由一系列类文件构成，每个类文件都有一个public类，每个类文件都有独自的.java和.class文件。

在一个\*.java文件（源码文件）里面只能够有一个public class定义（如果有多个无法区分类名称），而在一个源码文件中剩下的类，它们并非是“公共”的（非public），而是它们是由用于主public类的“支撑”类组成；**源代码的文件名必须与公共类的名字相同，并用.java作为扩展名。**

编译一个.java文件时，我们会得到一个名字完全相同的输出文件；但对于.java文件中的每个类，它们都有一个.class扩展名生成。因此，我们最终可以由一个.java文件里获得数量众多的.class文件。

java的工作方式：**一个有效的程序就是一系列.class文件，它们可以封装和压缩到一个jar文件里；java解释器负责对这些文件的寻找、装载和解释。**

类库是由N个包构成的，N>=1

java通过创建包时构建的分级文件结构，从而避免package的名称重复，名称冲突。

## 12.5 自定义工具库、导入、导出jar包

自定义工具库

目的；给System.out.println()创建一个别名，减少重复键入的代码量；

在一个工程src目录下，在包com.fenghuo.tools下创建一个类P

**package** com.fenghuo.tools;

**public** **class** P {

**public** **static** **void** rintln(String s) {

System.*out*.println(s);

}

}

在同一工程下(不同子文件夹不影响)，另一个.java源码文件中通过**import** com.fenghuo.tools.P;导入这个类P，**即可调用类P中的方法**

**package** com.fenghuo.test;

**import** com.fenghuo.tools.P;

**public** **class** HelloWorld {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.*out*.println("hello world");

P.*rintln*("weisong"); //调用com.fenghuo.tools.P类中的rintln()方法

}

}

问题：当测试在别的工程下导入这个类，则无法识别？

答：其实这个.java类文件所在的目录必须从某个CLASSPATH位置开始。然后继续com/fenghuo/tools，才可在自己系统的任何地方使用P.class文件；换句话说，当我们每次创建一个包后，都在为包取名时间接地指定了一个目录结构，这个包必须存在于由它的名字规定的目录内，而且这个目录必须能从CLASSPATH开始搜索并发现。

解决：A工程要调用B工程中的包（包含的类）

方法一：A-->properties-->java build path-->project-->add B

方法二：把工程B打成jar包，然后把jar包引入到工程A中，就可以直接写

import A了

具体：1、右键工程B-->export-->Java中JAR file---> ~~~ 将jar包放到某一路径下；或者通过cmd，jar cvf jar的包名.jar 源码.java

2、将工程A中导入jar包，通常我们导入的外部包存放在lib文件夹下，依次点击[New]-->[Floder]，建立lib目录；将jar包拖到lib文件夹或复制jar包到lib，然后在[Libraries]中[add JARs…]； 或者直接Add External JARs… 会自动生成Referenced Libraries(外部导入的jar包会自动放到里面)

Eclipse增加JDK及不同工程使用不同JDK方法：

eclipse 通过Preferences-->Java-->Installed JREs导入本地装的jdk，针对不同的工程可以装jdk1.7 1.8等多个版本使用

然后针对具体某个工程更改jdk（JRE System Library）-->Libraries-->Edit-->Alternate JRE

所以在使用新的Eclipse，首先要把本地安装的JDK导进去

## 12.6 private(封装性)访问修饰符

private访问修饰符声明的方法A：

对于在本.java文件中：在同一个类可以用于直接访问，不同的类不可以直接访问调用private声明的方法A；

对于不是同一个.java文件、但在同一个包中，或者不同包中的子类的对象都不可以直接访问调用方法A

但在其他的工程，如果导入了.java文件的jar包，直接import这个类（import com.fenghuo.tools.P;），即使这个类的方法是private，也是可以访问调用的。

也就是说，只要import com.fenghuo.tools.P; 不管P类中的方法是public还是private都也是访问、调用该类中的方法。

所以一般对于private属性的变量，往往都需要编写setter、getter方法。以private String name为例：

* setter：public void setName(String n);
* getter：public String getName();

在源码中，其实一个类（封装变量和方法）就是一个新的数据类型，

封装性是面向对象之中第一大重要特征，其主要的意义在于类内部的操作对外部不可见。

private 声明：只可以在定义的类中使用，其他类中对象调用不可用

**package** com.fenghuo.test;

**class** Person { // 定义类，类名称首字母大写

**private** String name; // 定义属性，表示姓名

**private** **int** age; // 定义属性，表示年龄

**public** **void** setName(String n) {

name = n;

}

**public** **void** setAge(**int** a) {

age = a;

}

**public** String getName(){

**return** name;

}

**public** **int** getAge(){

**return** age;

}

**public** **void** tell(){

System.*out*.println("姓名："+name+"，年龄"+age);

}

}

**public** **class** TestDemo {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Person per = **new** Person();

per.setName("张三");

per.setAge(25);

per.tell();

System.*out*.println(per.getName());

System.*out*.println(per.getAge());

}

}

注：per.tell()会带哦用tell()方法，再从中调用上方赋值的name和age

返回结果：

姓名：张三，年龄25

张三

25

## 12.7 理解构造方法

常量

在java中，利用关键字final指示常量。例如：

final int a = 22; //一旦这个变量a被赋值后，就不能再更改了。

源代码写好或有调整时，可通过Source-->Format修正格式

主类中main方法调用其他类的变量和方法时，要先在main设置好变量数组或变量值，然后通过调用其他类实例化对象.方法()的方式调用出具体的变量值

**public** **class** EmployeeTest {

**public** **static** **void** main(String[] args) {

Employee[] staff = **new** Employee[3];

staff[0] = **new** Employee("Carl Cracker", 75000, 1987, 12, 15);

staff[1] = **new** Employee("Harry Hacker", 50000, 1989, 10, 1);

staff[2] = **new** Employee("Tony Tester", 40000, 1990, 3, 15);

**for** (Employee e : staff) {

e.raiseSalary(5);

}

**for** (Employee e : staff) {

System.*out*.println("name=" + e.getName() + ",salary="

+ e.getSalary() + ",hireDay=" + e.getHireDay());

}

}

}

在一个源代码文件.java中调用同一个包其他类，不需要import

构造方法与其他方法的不同在于：

构造方法总是伴随着new操作符的执行被调用，而不能对一个已经存在的对象调用构造器来达到重新设置实例域的目的。（换句话说，**构造方法总是使用在类实例化对象是new实例中**，即：）

类 对象 = new 类（**构造方法**）；

构造方法例子：

**public** Employee(String n, **double** s, **int** year, **int** month, **int** day) { // 类Employee的一种构造方法/构造器

name = n;

salary = s;

hireDay = LocalDate.*of*(year, month, day);

}

如果public Employee(){..}是这种无参构造器的话，则这个构造器将所有的实例域设置为默认值，即name = “”; salary=0; hireDay=LocalDate.now();

* 提示：不管是怎样的构造方法，其目的之一就是将每个实例域设置为一个有意义的初值！构造方法是不能有返回值的！

## 12.8 隐式参数与显式参数，this关键字

方法用于操作**对象**以及存取它们的实例域。例如，方法：

**public** **void** raiseSalary(**double** byPercent) {

double raise = salary \* byPercent / 100;

salary += raise;

}

* 提示：形参：在方法定义中用到，如上面的byPercent；实参：在调用方法时实际传入的参数，如e.raiseSalary(5)，5就是一个实参。

将调用这个方法的对象的salary实例域设置为新值。看看下面这个调用：

**number007**.raiseSalary(5);

它的结果将number007.salary域的值增加5%。具体的说，这个调用将执行下面指令：

double raise = **number007**.salary \* 5 /100;

**number007**.salary += raise;

raiseSalary方法有两个参数。第一个参数称为隐式（implicit）参数，是出现在方法名前的Employee类对象（即**number007**）。第二个参数位于方法名后面括号中的数值，这是一个显式参数（显式参数就是形参）。

在每一个方法中，关键字this表示隐式参数。如果需要的话，可以用下列方法编写raiseSalary方法：

**public** **void** raiseSalary(**double** byPercent) {

double raise = **this**.salary \* byPercent / 100;

**this**.salary += raise;

}

有些程序员更偏爱这样的风格，因为这样可以将实例域与局部变量明显地区分开。实例域是类中被定义的类内部的变量（实例域是变量，具体的值在构造方法中给出形参，然后调用这个方法时才写明具体的值），在类内部写方法的时候，如果要调用类定义的变量（即实例域），加上this关键字好区分与方法中的局部变量！

**this表示：在之后调用类这个方法的类的实例化对象将会调用this.xxx中的xxx的实例域；简而言之，this是为了明确表示对象将会调取了类中哪些实例域。**

**再比如：**

**有一个Hero类，还有一个ADHero类，（ADHero类继承了Hero类），还有一个 LolHerol类（含main方法）**

**package** com.fenghuo.interfacetest;

**public** **class** Hero {

String name; //定义属性，表示姓名

**float** hp; //定义属性，表示血量

**float** armor; //定义属性，表示护甲

**int** movespeed; //定义属性，表示移动速度

}

**package** com.fenghuo.interfacetest;

//ADHero是接口AD的一个具体的实现类，因此要有具体方法，变量赋值可以再主类的main方法中设置

**public** **class** ADHero **extends** Hero **implements** AD {

**public** **void** physicAttack() {

System.*out*.println(**this**.name + "进行物理攻击");

}

}

**package** com.fenghuo.interfacetest;

**public** **class** LolHero {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ADHero ah = **new** ADHero();

ah.name="亚索";

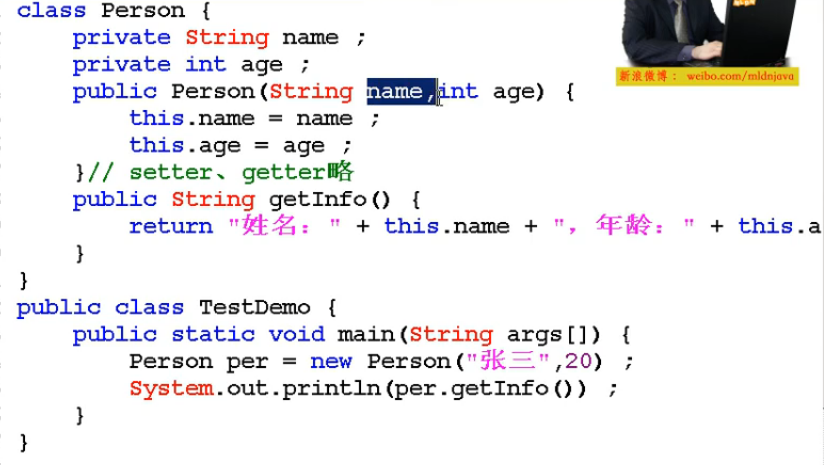
ah.physicAttack();

}

}

**注意：这里面在主类的main方法中当我们对ADHero类实例化对象后，可以对ADHero类中的变量进行赋值、方法进行调用。ah.name=… 所以在ADHero类中变量前就可以加上this. 表示在实例化对象后会调取这个实例域！同时方便ADHero类中this.实例域和局部变量进行区分。**

**再比如：**



## 12.10 静态域

main方法为何都被标记static修饰符？

1、静态域

例：给Employee类添加一个实例域id和一个静态域nextid:

class Employee {

private static int nextid = 1;

private int id;

}

现在，每一个雇员对象都有一个自己的id域，但这个类的所有实例将共享一个nextid域。换句话说，如果有1000个Employee类的对象，则有1000个实例域id。但是，只有一个静态域nextid。即使没有一个雇员对象，静态域nextid也存在。它属于类，而不属于任何独立的对象。

2、静态常量

静态变量使用的比较少，但静态常量却使用的比较多。如：

public class Math {

public static final double PI = 3.1415926;

}

在程序中，可以采用Math.PI的形式获得这个常量。

3、静态方法

注意：不需要使用对象调用静态方法。例如，不需要构造Math类对象就可以调用Math.pow方法。

main方法也是一个静态方法。事实上，在启动程序时还没有任何一个对象，静态的main方法将执行并创建程序所需要的对象。

* 提示：每一个类都可以有一个main方法。这是一个常用于对类进行单元测试的技巧。如果这个类是一个更大型应用程序的一部分，则有main方法也不会对更大的程序执行时产生影响，因为测试的类的main方法永远不会被执行！

## 12.11 包、类路径

1、包

java允许使用包（package）将类组织起来。标准的java类库分布在多个包中，包括java.lang、java.util和java.net等。

使用包的主要原因是确保类名的唯一性。事实上，为了保证包名的绝对唯一性，Sun公司建议将公司的因特网域名（这显然是独一无二的）以逆序的形式作为包名，并且对于不同的项目使用不同的子包。例如：fenghuo.com域名，则建立包是com.fenghuo。这个包还可以根据项目不同进一步划分成子包，如com.fenghuo.test

2、将类放入包中

要想将一个类放入包中，就必须将包名放在源文件的开头，如：

**package** com.fenghuo.test;

如果源文件开头没有package语句，这个源文件中的类就会被放置在一个默认包（default package）中

3、类路径

前面已经看到，类存储在文件系统的子目录中。类的路径必须与包名匹配。

另外，类文件也可以存储在JAR（java归档）文件中。在一个JAR文件中，可以包含多个压缩形式的类文件和子目录，这样既节省又可以改善性能。jdk也提供了许多的JAR文件，例如，在jre/lib/rt.jar中包含数千个类库文件。（JDK安装目录下有一个jre目录，里面有两个文件夹bin和lib，可以认为bin里的就是jvm，**lib中则是jvm工作所需要的类库**。）

设置类路径：

最好采用-classpath选项指定类路径

java –classpath /home/user/classdir:.:/home/user/archive.jar MyProg

或者通过设置CLASSPATH环境变量完成这个操作。

#set java

export JAVA\_HOME=/usr/java/jdk1.7.0\_71

export CLASSPATH=.:$CLASSPATH:$JAVA\_HOME/lib:$JAVA\_HOME/jre/lib

export PATH=$JAVA\_HOME/bin:$PATH

java的环境变量中:

CLASSPATH配置的是java类库的位置（$JAVA\_HOME/jre/lib）以及执行字节码文件时查到文件的路径（.:$CLASSPATH:$JAVA\_HOME/lib），即类库加载的路径

PATH配置的是执行java程序省略写$JAVA\_HOME的绝对路径

* 提示：在命令中输入#java –version 能返回版本，这是因为在环境变量中设置好了路径PATH=$JAVA\_HOME/bin:$PATH，则输入java时默认会执行的是路径下的java脚本，即/usr/java/jdk1.7.0\_71/bin/java –version；（环境变量的功能就是将一个命令名不需要输出其绝对路径，而可以直接通过这个命令名形式直接调用命令！ 环境变量已经将命令搜索的路径全列到了环境变量中，省事！）

## 12.13 原始类型和封装类

java提供两种不同的类型：原始类型（内置类型）和引用类型。比如int是java的原始数据类型，Integer是java为inter提供的封装类。java为每个原始类型提供了封装类，常见的原始与对应的封装类如下：

|  |  |
| --- | --- |
| 原始类型 | 封装类 |
| boolean | Boolean |
| char | Character |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |

引用类型和原始类型的区别：

1.两者的初始化方式不同

int i = 5; //原始类型

Integer j = new Integer(10); 或者Integer j = 10 ; String k = “ws”; //引用对象，两种方式都可以

String province = getProvince();

使用原始类型无须调用new，也无须创建对象，而使用引用类型必须创建对象。

2.原始类型是类，引用类型是对象

原始类型大小比较用“==”，引用类型大小比较用“equals”

3.引用类型可以被序列化，原始类型不行

4.引用类型提供的方法可以灵活转换，可以扩展，原始类型不行

5.在集合类中只能使用引用类型，不能使用原始类型

6.原始类型没有null概念，引用类型（类、数组、接口）默认值是null

## 12.18 代码的逻辑流程

* 代码的开发思路、逻辑流程和写代码的顺序：

写代码的顺序最好和实际操作的逻辑一样，先做什么后做什么，这样能最大程度的保证逻辑不会出什么问题。前提是你要熟悉业务逻辑。

还有开发的时候不要局限于一个方法的实现代码逻辑，**先从整体上把方法都写出来，比如这个方法要做什么，只管写一个方法就行，具体怎么做可以再框架都搭完了再写，**更好的就是全部用接口，接口全部写好了之后再写实现类，这样更好。

还有一个注意点，方法里面的代码最好不要太长，最好不要超过一个屏幕，不然看起来很累，可以把一个方法分成好几个方法来写，那样BUG也会减少很多，逻辑性强。还有就是最好不要循环嵌套，嵌套个两三层后出问题就麻烦了，如果一定要嵌套就分出去一个方法就好。

还有就是要多集成，能通用的方法就通用写掉，写成工具类，如果有框架了，多多熟悉原本就有的工具类，都自己写的话代码多不说，还很容易出错，要学会活用别人的代码。

## 12.20 接口作用理解

问题：java程序中接口只定义了常量和空方法（抽象方法），空的方法有什么用呢？还要在具体实现类中写方法体，那还要接口干什么？

在项目中，有一些功能，不仅仅是一个类去实现，即有很多地方都能用到，大家需要统一标准。所以需要统一标准（统一只能实现哪些方法），所以需要用接口，这样大家同时实现一个接口，在我们各自的实现类中将接口中的抽象方法（根据自己的需求）具体化，这样有利于统一规范。在接口中只定义了基本的方法名，具体的实现要交给实现类。

接口就是个招牌，比如接口是KFC，看到了这个接口，就知道这个店里有炸鸡腿（接口实现方法）

Java的接口是可以实现多继承的（一个接口可以继承多个其他接口），类不允许多继承。如果把不设计成接口A、接口B等而是设计成类A、类B等，在实际自定义类文件中不可以同时继承多个类A,B，这样无法实现A、B中的功能，但可以多继承A,B就可以实现A、B中的功能了。所以，对于那种功能单一（如响铃、猫眼），又需要拿来作为一个附件附加到基本类上的类，我们就把它定义成接口。

一个类描述了一个实体，这个实体可能是一个复杂的对象，它的动作很多，如果把这些动作分类，用接口a定义其中的某一组动作，接口b定义其中的另外一组动作，这样的结构，比较清楚。这种方式具备了多继承的优点，避免了多继承的缺陷，实际上在历史上，接口在很大程度上，是为了解决多继承带来的种种问题而设计出来的。

接口定义后的主类，当我们需要拓展（添加）其他类后，主类仍然可以拿来就用，无需进行更多的修改，此时就可以显示出接口的作用了。如果没有用接口，在主方法中要改的东西就很多，需要不同定义实例化对象。

定义好接口后，如果有A类implement这个接口，则这个类A就是这个接口的一个实现类，一个接口可以有无数个实现类，**在这个实现类A中要给变量赋值，根据需求重写接口中的方法，**最后在主类中当对这个类A实例化对象时，直接对象.方法(); 在主类main方法中不必在设置变量赋值，因为赋值的变量对类A没有任何影响。

接口的作用：

1、精简程序结构，免除重复定义

比如，有两个及上的的类拥有相同的方法，但是实现功能不一样，就可以定义一个接口，将这个方法提炼出来，在需要使用该方法的类中去实现，就免除了多个类定义系统方法的麻烦。

2、拓展程序功能，应对需求变化

在主类基础上直接添加新的类（功能），无需进行更多的修改。

## 12.25 public中的内部类问题

如下例：

**public** **class** PersonInterface {

**public** **static** **void** main(String[] args) {

Student s = **new** Student();

s.eat();

}

**class** Student **implements** Person {

**public** **void** eat() {

System.*out*.println("学生去食堂吃饭！");

}

}

上面语句报错信息如下：



原因：类Student是属于PersonInterface的内部类，内部类是动态的（无static关键字修饰），类似于普通的实例变量，而main方法是静态的，不可以直接调用类的实例变量，所以创建的内部类也要是**static**静态的才行。

解决方法有以下四种：

1、将内部类定义为**static**，即静态类

2、将Student s = new Student();改为

PersonInterface pi = new PersonInterface();

Student s = pi.new Student();

3、将类Student写在另一个.java源码中，main方法调用另一个源码中的类

4、将内部类Student和内部接口放在外部，即在**public** **class** PersonInterface {…}之外创建类

**public** **class** PersonInterface {

**public** **static** **void** main(String[] args) {

Student s = **new** Student();

s.eat();

}

**static class** Student **implements** Person {

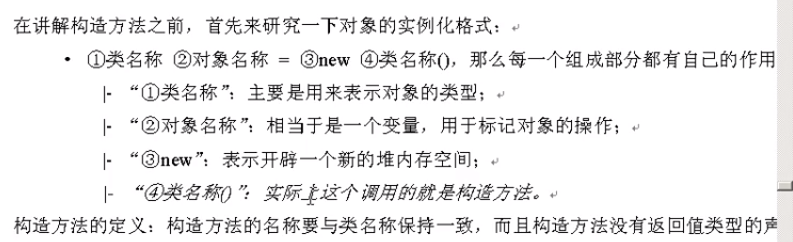
**public** **void** eat() {

System.*out*.println("学生去食堂吃饭！");

}

## 12.26 类创建实例化对象开辟内存

**所有的引用数据类型，如果要想进行使用，那么必须通过关键字“new”来进行内存的分配操作（开辟内存）**



例如：

**package** com.fenghuo.test;

**class** People{

**public** People(){

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

**public** **class** TestJava {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

People peo = **new** People();

}

}

注：该对象类型是最左边的People类型，new表示为对象peo开辟一个新的堆内存空间，**最后People()表示调用类People的无参构造方法**，如果类People中没写无参构造方法（每个类默认都有一个无参构造方法），也没有影响，如果类People含有具体的无参构造方法，则实例化对象并调用无参构造方法后，将返回无参构造方法中的值。但是如果此时类之中已经明确的定义了构造方法的话，则此默认构造方法将不会被自动生成。

## 12.27 关于main主方法

此时的程序只能说是完成了功能，但这个代码并不好，实际上主方法是一切操作的起点，说白了，**所有的客户操作都是在主方法上进行的**。所以在以后的开发之中，一定要秉持一个原则“**主方法之中的代码越少越好**，最好都简单的像1—+1应用那样”。**所以这个时候就可以利用方法来简化代码。**主方法客户端只需要调用方法就可以完成指定的功能。

利用方法减少客户端代码，在java之中有一个命名规范，如果说某一个方法返回的数据类型是boolean，一般此方法名称都会以“isXxx”的形式命名。

## 12.28 java中变量、方法声明的格式

java类中**属性**的声明采用如下格式：

**访问修饰符 修饰符 类型 属性名称=初始值；**

访问修饰符：可以使用四种不同的访问修饰符中的一种，包括public(公共的)、 protected（受保护的），无修饰符和 private（私有的）。public 访问修饰符表示属性可以 从任何其它代码调用。private 表示属性只可以由该类中的其它方法来调用。protected 将 在以后的课程中讨论。

修饰符：是对属性特性的描述，例如后面会学习到的：static、final 等等。（可省略）

类型：属性的数据类型，可以是任意的类型。

属性名称：任何合法标识符

Java 类中**方法**的声明采用以下格式：

**访问修饰符 修饰符 返回值类型 方法名称 (参数列表) throws 异常列表 {方法体}**

返回值类型：表示方法返回值的类型。如果方法不返回任何值，它必须声明为 void(空)。 Java 技术对返回值是很严格的，例如，如果声明某方法返回一个 int 值，那么方法必须从所有可能的返回路径中返回一个 int 值（只能在等待返回该 int 值的上下文中被调用。）

参数列表：允许将参数值传递到方法中。列举的元素由逗号分开，而每一个元素包含一个类型和一个标识符。在下面的方法中只有一个形式参数，用 int 类型和标识符 days 来声 明：public void test(int days){}

throws 异常列表：子句导致一个运行时错误（异常）被报告到调用的方法中，以便以 合适的方式处理它。

## 12.29 ASCII编码、Unicode、UTF-8详解

1. ASCII

我们知道，计算机内部，所有信息最终都是一个二进制值。每一个二进制位（bit）有0和1两种状态，因此八个二进制位(八位二进制)就可以组合出256种状态，这被称为一个字节（byte）。也就是说，一个字节一共可以用来表示256种不同的状态，每一个状态对应一个符号，就是256个符号，从00000000到11111111。

上个世纪60年代，美国制定了一套字符编码，对英语字符与二进制位之间的关系，做了统一规定。这被称为 ASCII 码，一直沿用至今。

ASCII 码一共规定了128个字符的编码，比如空格SPACE是32（二进制00100000），大写的字母A是65（二进制01000001）。这128个符号（包括32个不能打印出来的控制符号），只占用了一个字节的后面7位，最前面的一位统一规定为0。

总结：一个字节(byte, 1KB=1024Byte)由八位二进制组成，每个二进制位(bit)有0和1两种状态，即1个字节共有256种不同状态。而ASCII码规定了128个字符的编码，比如空格SPACE是32（二进制00100000），大写的字母A是65（二进制01000001）,只占用了一个字节的后面7位，最前面的一位统一规定为0.因此，英语字母都是单字节，ASCII码都是单字节的，因此英文字母都是ASCII码（A-Z：65-90，a-z：97-122）

1. 非ASCII编码

英语用128个符号编码就够了，但是用来表示其他语言，128个符号是不够的。比如，在法语中，字母上方有注音符号，它就无法用 ASCII 码表示。于是，一些欧洲国家就决定，利用字节中闲置的最高位编入新的符号。比如，法语中的é的编码为130（二进制10000010）。这样一来，这些欧洲国家使用的编码体系，可以表示最多256个符号。

至于亚洲国家的文字，使用的符号就更多了，汉字就多达10万左右。一个字节只能表示256种符号，肯定是不够的，就必须使用多个字节表达一个符号。比如，简体中文常见的编码方式是 GB2312，使用两个字节表示一个汉字，所以理论上最多可以表示 256 x 256 = 65536 个符号。虽然都是用多个字节表示一个符号，但是GB类的汉字编码与后文的 Unicode 和 UTF-8 是毫无关系的。

1. Unicode

可以想象，如果有一种编码，将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的编码，那么乱码问题就会消失。这就是 Unicode，就像它的名字都表示的，这是一种所有符号的编码。

Unicode 当然是一个很大的集合，现在的规模可以容纳100多万个符号。每个符号的编码都不一样，比如，U+0639表示阿拉伯字母Ain，U+0041表示英语的大写字母A，U+4E25表示汉字严。具体的符号对应表，可以查询unicode.org，或者专门的汉字对应表。

1. Unicode的问题

需要注意的是，Unicode 只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储。

比如，汉字严的 Unicode 是十六进制数4E25，转换成二进制数足足有15位（100111000100101），也就是说，这个符号的表示至少需要2个字节。表示其他更大的符号，可能需要3个字节或者4个字节，甚至更多。

这里就有两个严重的问题，第一个问题是，如何才能区别 Unicode 和 ASCII ？计算机怎么知道三个字节表示一个符号，而不是分别表示三个符号呢？第二个问题是，我们已经知道，英文字母只用一个字节表示就够了，如果 Unicode 统一规定，每个符号用三个或四个字节表示，那么每个英文字母前都必然有二到三个字节是0，这对于存储来说是极大的浪费，文本文件的大小会因此大出二三倍，这是无法接受的。

它们造成的结果是：1）出现了 Unicode 的多种存储方式，也就是说有许多种不同的二进制格式，可以用来表示 Unicode。2）Unicode 在很长一段时间内无法推广，直到互联网的出现。

5 UTF-8

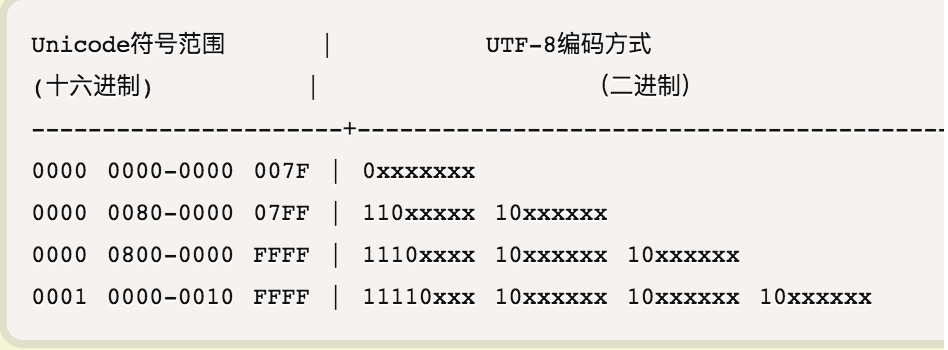
互联网的普及，强烈要求出现一种统一的编码方式。UTF-8 就是在互联网上使用最广的一种 Unicode 的实现方式。

UTF-8 最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号(即1~4个字节表示一个字符)，根据不同的符号而变化字节长度。

UTF-8 的编码规则很简单，只有二条：

1）对于单字节的符号，字节的第一位设为0，后面7位为这个符号的 Unicode 码。因此对于英语字母，UTF-8 编码和 ASCII 码是相同的。

2）对于n字节的符号（n > 1），即n个字符表示1个字符，第一个字节的前n位都设为1，第n + 1位设为0，后面字节的前两位一律设为10。剩下的没有提及的二进制位，全部为这个符号的 Unicode 码。



跟据上表，解读 UTF-8 编码非常简单。如果一个字节的第一位是0，则这个字节单独就是一个字符；如果第一位是1，则连续有多少个1，就表示当前字符占用多少个字节。

下面，还是以汉字严为例，演示如何实现 UTF-8 编码。

严的 Unicode 是4E25（100111000100101），根据上表，可以发现4E25处在第三行的范围内（0000 0800 - 0000 FFFF），因此严的 UTF-8 编码需要三个字节，即格式是1110xxxx 10xxxxxx 10xxxxxx。然后，从严的最后一个二进制位开始，依次从后向前填入格式中的x，多出的位补0。这样就得到了，严的 UTF-8 编码是11100100 10111000 10100101，转换成十六进制就是E4B8A5

如果都是英文字母的话，UTF-8和ASCII码一样，都只占用了1个字节，这就是UTF-8的好处，根据不同的字符其字节长度可变，从而节省存储空间。

1. Unicode和UTF-8之间的转换

自行百度

总结：

现在，捋一捋ASCII编码和Unicode编码的区别：ASCII编码是1个字节，而Unicode编码通常是2个字节。

字母A用ASCII编码是十进制的65，二进制的01000001；

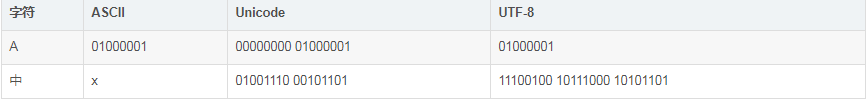
字符0用ASCII编码是十进制的48，二进制的00110000，注意字符'0'和整数0是不同的；

汉字中已经超出了ASCII编码的范围，用Unicode编码是十进制的20013，二进制的01001110 00101101。

你可以猜测，如果把ASCII编码的A用Unicode编码，只需要在前面补0就可以，因此，A的Unicode编码是00000000 01000001。

新的问题又出现了：如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算。

所以，本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的UTF-8编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。如果你要传输的文本包含大量英文字符，用UTF-8编码就能节省空间：

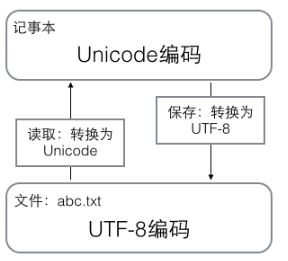


从上面的表格还可以发现，UTF-8编码有一个额外的好处，就是ASCII编码实际上可以被看成是UTF-8编码的一部分，所以，大量只支持ASCII编码的历史遗留软件可以在UTF-8编码下继续工作

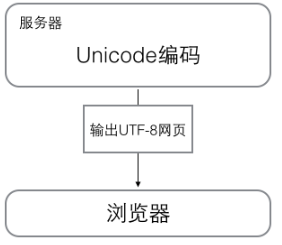
搞清楚了ASCII、Unicode和UTF-8的关系，我们就可以总结一下现在计算机系统通用的字符编码工作方式：

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件：



浏览网页的时候，服务器会把动态生成的Unicode内容转换为UTF-8再传输到浏览器：



所以你看到很多网页的源码上会有类似<meta charset="UTF-8" />的信息，表示该网页正是用的UTF-8编码。

全文总结：

* 一个字节(byte, 1KB=1024Byte)由八位二进制组成，即1个字节共有256种不同状态。而ASCII码规定了128个字符的编码，只占用了一个字节的后面7位，最前面的一位统一规定为0。（这是二进制下的ASCII，对应十进制下的ASCII是从0-127）因此，英语字母都是单字节，ASCII码都是单字节的，因此英文字母都是ASCII码（A-Z：65-90，a-z：97-122）；
* 如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算；
* 本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的UTF-8编码。如果都是英文字母的话，UTF-8和ASCII码一样，都只占用了1个字节，这就是UTF-8的好处，根据不同的字符其字节长度可变，从而节省存储空间。（对于符号是英文字母的话，UTF-8就是ASCII码，因为只需用单一字节）；
* 在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

**对于编码的自己理解：**

1、对于计算机来说，所有的信息都是二进制值。每一个二进制位(bit)都是两种状态，即0和1。而对于计算机，它进行存储容量的基本计量单位是字节（Byte,简称B）。而一个字节是由8个二进制位组成，即一个字节等于8个比特(1Byte = 8bit)，比如一个字节可以是01101010；这样，一个字节可以有256个不同的状态。而美国制定了一套字符编码即ASCII码，ASCII码一共规定了128个字符的编码（只占用了一个字节的后面7位，最前面的一位统一规定为0），这样对于英文字母，都属于单字节的ASCII码。

2、而对于其他国家的语言，显然ASCII码无法满足需求。这时候就出现了unicode编码，将世界上所有的符号都纳入其中，每一个符号都给予一个独一无二的编码。但是，它有一个问题，由于不同字符有的占用了四个字节，有的字符只需占用两个字节，而unicode编码无法识别到底是几个字节表示一个字符，如果unicode规定每个字符都占用四个字节表示，那么对英文字母必然有三个字节都是0，这样对于存储来说是极大的浪费，同时也影响传输，这就是unicode一个大问题：没有解决存储问题。

3、UTF-8是unicode使用最广的一种实现方式，它是一种可变长的编码方式。它可以使用1~4个字节表示一个字符，根据不同的字符而变化字节长度！这样对于中文字符，UTF-8一般需要占用三个字节，对于英文字符，UTF-8只需要占用1个字节，就变成了ASCII码。

4、在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

## 1.3

IDE（integrated Development Environment）集成开发环境，是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具。

常见的IDE有Eclipse、IDEA

## 1.4 java三大特性：封装、继承、多态

OOP(Object-Oriented Programming)面向对象编程

**封装**是在抽象基础上决定信息是否公开，以及公开等级，核心问题是以什么样的方式暴露哪些信息。。封装使面向对象的世界变得单纯，对象之间的关系变得简单，各人自扫门前雪，精合度变弱，有利于维护。设计模式七大原则之一的迪米特法则就是对于封装的具体要求，即A 模块使用B 模块的某个接口行为，对B 模块中除此行为之外的其他信息知道得尽可能少。比如。耳塞的插孔就是提供声音输出的行为接口，只需关心这个插孔是否有相应的耳塞标记，是否是圆形的，有没有声音即可，至于内部CPU 如何运算音频信息，以及各个电窑如何协同工作，根本不需要去关注，这使模块之间的协作只需忠于接口、忠于功能实现即可。封装这件事情是由俭入奢易，由奢入俭难。属性值的访问与修改需要使用相应的getter/setter 方法，而不是直接对public 的属性进行读取和修改。

**继承**：is-a关系即“D is a B”：概念体D物是概念体B物的特殊化，B则是D的一般化。继承的使用成本很低，一个关键字就可以使用别人的方法，似乎更加轻量简单。想复用别人的代码，跳至脑海的第一反应是继承它，所以继承像抗生素一样窑易被滥用，我们传递的理念是谨慎使用继承，认清继承滥用的危害性，即方法污染和方法爆炸。方法污染是指父类具备的行为，通过继承传递给子类，子类并不具备执行此行为的能力，比如鸟会飞，驼鸟继承鸟，发现飞不了，这就是方法污染。子类继承父类，则说明子类对象可以调用父类对象的－切行为。在这样的情况下，总不能在继承时，添加注释说明哪几个父类方法不能在子类中执行，更不能覆写这些无法执行的父类方法，抛出异常，以阻止别人的调用。方法爆炸是指继承树不断扩大，底层类拥有的方法虽然都能够执行，但是由于方法众多，其中部分方法并非与当前类的功能定位相关，很窑易在实际编程中产生选择困难症。比如某些综合功能的类，经过多次继承后达到上百个方法， 造成了方法爆炸，因而带来使用不便和安全隐患。在实际故障中， 因为方法爆炸，父类的某些方法签名和子类非常相似，在IDE 中，输人类名＋点之后，在自动提示的极为相似的方法签名中选择错误，导致线上异常。综上所述，提倡组合优先原则来扩展类的能力，即优先采用组合或聚合的类关系来复用其他类的能力，而不是继承。

**多态**是以上述的三个面向对象特性为基础，根据运行时的实际对象类型，同一个方法产生不同的运行结果，使同一个行为具有不同的表现形式。“override”和“overload”，“override”译成“覆写”，是子类实现接口，或者继承父类时，保持方法名完全相同，实现不同的方法体，是垂直方向上行为的不同实现。“ove rload ”译成“重载”，方法名称是相同的，但是参数类型或参数个数是不相同的，是水平方向上行为的不同实现。多态是指在编译层面无法确定最终调用的方法体， 以覆写为基础来实现面向对象特性，在运行期由JVM进行动态绑定， 调用合适的覆写方法体来执行。重载是编译期确定方法调用， 属于静态绑定，本质上重载的结果是完全不同的方法，所以本书认为**多态专指覆写**。自然界的多态最典型例子就是碳家族，据说某化学家告诉他女朋友将在她的生日晚会上送她一块碳，女朋友当然不高兴，可收到的却是5 克拉的钻石。钻石就是碳元素在不断进化过程中的一种多态表现。严格意义上来说，多态并不是面向对象的一种特质，而是一种由继承行为衍生而来的进化能力而已。

向上转型：子类对象变成父类对象，语法：父类 父类对象 = 子类对象，自动；

向下转型：父类对象变成子类对象，语法：子类 子类对象 = （子类）父类实例，强制；

对于多态，总结一为：

1、定义一个父类类型的引用指向一个子类的对象既可以使用子类强大的功能，又可以抽取父类的共性；

2、父类类型的引用可以调用父类中定义的所有属性和方法，而对于子类中定义而父类中没有的方法，它是无可奈何的；

3、父类中的一个方法只有在父类中定义而在子类中没有覆写的情况下，才可以被父类类型的引用调用；

4、对于父类中定义的方法，如果子类中覆写了该方法，那么父类类型的引用将会调用子类中的这个方法，这就是动态链接。

对于多态，可以总结二为：

1、使用父类类型的引用指向子类的对象；

2、该引用只能调用父类中定义的方法和变量；

3、如果子类中覆写了父类中的一个方法，那么在调用这个方法的时候，将会调用子类中的这个方法；（动态链接、动态调用）；

4、变量不能被覆写，“覆写”的概念只针对方法，如果在子类中覆写了父类中的变量，那么在编译时会报错。

总结：

* 继承的好处是代码复用，继承的子类自动拥有父类中的所有属性和方法。所以继承已存在的类就是复用这些类的方法和域。在此基础上，子类还可以添加一些新的方法和域，以满足新的需求，这是java程序设计中的一项核心技术。**为什么要继承：继承就是为了代码的复用或者方法的覆写或重载还可以在此基础上子类添加新的方法和变量。**
* 多态（向上转型）就是用父类对象的类型信息表去调用子类类型信息表中和父类相同的成员方法。所谓相同，就是子类中覆写父类方法。（对象会调用到子类中覆写的方法及结果）父类信息表中没有的成员方法，就调用不到。还有一种向下转型，就是用子类的类型信息表代替父类的类型信息表，就可以完全的让位子类对象中所有的变量和方法。(参考上方多态总结一二)

## 1.5 super和this关键字

super可以理解为是指向自己超（父）类对象的一个指针，而这个超类指的是离自己最近的一个父类。

super也有三种用法：

1.普通的直接引用

与this类似，super相当于是指向当前对象的父类，这样就可以用super.xxx来引用父类的成员。

2.子类中的成员变量或方法与父类中的成员变量或方法同名

3.引用构造函数

super（参数）：调用父类中的某一个构造函数（应该为构造函数中的第一条语句）。

this（参数）：调用本类中另一种形式的构造函数（应该为构造函数中的第一条语句）。

super和this的异同：

* super（参数）：调用基类中的某一个构造函数（应该为构造函数中的第一条语句）
* this（参数）：调用本类中另一种形成的构造函数（应该为构造函数中的第一条语句）
* super:　它引用当前对象的直接父类中的成员（用来访问直接父类中被隐藏的父类中成员数据或函数，基类与派生类中有相同成员定义时如：super.变量名    super.成员函数据名（实参）
* this：它代表当前对象名（在程序中易产生二义性之处，应使用this来指明当前对象；如果函数的形参与类中的成员数据同名，这时需用this来指明成员变量名）
* 调用super()必须写在子类构造方法的第一行，否则编译不通过。每个子类构造方法的第一条语句，都是隐含地调用super()，如果父类没有这种形式的构造函数，那么在编译的时候就会报错。
* super()和this()类似,区别是，super()从子类中调用父类的构造方法，this()在同一类内调用其它方法。
* super()和this()均需放在构造方法内第一行。
* 尽管可以用this调用一个构造器，但却不能调用两个。
* this和super不能同时出现在一个构造函数里面，因为this必然会调用其它的构造函数，其它的构造函数必然也会有super语句的存在，所以在同一个构造函数里面有相同的语句，就失去了语句的意义，编译器也不会通过。
* this()和super()都指的是对象，所以，均不可以在static环境中使用。包括：static变量,static方法，static语句块。
* 从本质上讲，this是一个指向本对象的指针, 然而super是一个Java关键字。
  1. java jxl

sheet.getRows()是统计单个sheet具有的行数（excel行数-1 因为jxl中行数是从0开始算）

而统计sheet工作表个数 有多少个工作表就是多少

例如：一共有215个sheet 则正确写法是 从0开始 一直到215

for (in\_excel\_index = 0; in\_excel\_index < 215; in\_excel\_index++) {

3.15