4.数据类型与表达式

整数：byte，short，int，long，分别对应1,2,4,8字节长度，范围与C/C++的一致，例如，byte为-128-127.

byte a = 12;

int b = 123456;

Java中没有无符号类型；

浮点类型：

float，double，和C/C++中一样

输出整数与小数：

直接与字符串连载一起即可，Java会自动把变量转成字符串。

               int a = 10;

               double b = 123.456;

              System. out .println("a=" +a +",b=" +b );

布尔类型：

Java里的布尔类型boolean，有两个字面常量true，false。

例如：boolean checked = true;

Java里的boolean类型与整型不通用！

在C/c++里：

int a = 12;

if(a) {}

在Java里必须这么写：

int a = 12;

if(a != 0)//括号里必须是Boolean类型

{

}

表达式：和C++基本一致

1.算术与赋值

a+b

5%3

a+=3；

2.关系表达式，值是Boolean类型

a>b

3.条件表达式

cond ? expr1:expr2

cond:必须是Boolean类型

若cond为true，则取expr1；否则取expr2.

4.逻辑表达式

a && b

a || b

! a

注：a，b必须是Boolean类型的

5.语句

if/switch/for/while与C/c++相同，但强制用Boolean表达式。

               if (true )

              {

                     System. out .println("ok" );

              }

               else

               {

                     System. out .println( "not nok");

              }

               int ch = 1;

               switch (ch )

              {

               case 1:

                     System. out .println("Monday" );

                      break ;

               case 2:

                     System. out .println("Th" );

                      break ;

              }

               for (int i =0;i <10;i ++)

              {

                     System. out .println(i );

              }

               int n = 0;

               while (n <10)

              {

                     System. out .println("got" +n );

                      n+=2;

              }

6.新建一个类

先新建一个包（package），然后在左边的列表中右击这个package，选择new，选择class，然后命名，点击Finnish。

创建一个类的对象：

注意：所有的对象都必须new出来；

          java对象实际上是指针；

          用点号 . 不用箭头->

下面是一个package my下的一个类，名字是Example，这是个独立的\*.Java文件，文件名就是类名。每次新建类都是独立的一个\*.Java文件，这个文件里这只能有这一个类。而且类名必须与文件名相同，每个XXX.java被编译成同名的XXX.class。如果要在建一个类，必须在“New|Class”

package my;

public class Example

{

        private int id = 0;//添加了一个成员变量,访问修饰符直接放在前面，每一个都要

        public int numb = 1;

        public int getid()//成员函数

       {

               return this .id ;//java中this总是使用。，不再使用-》

       }

        public void setId( int id)

       {

               this .id = id ;

       }

        public void print()

       {

              System. out .println("example(id=" +id +")" );

       }

}

使用类的时候，new一个对象就可以了，例子见下一节。

类中函数和变量的调用与C/C++相似，但是只用点号.

7.类的构造函数

下边这个类只是比上一届的多两个构造函数。

package my;

public class Example

{

        private int id = 0;//添加了一个成员变量,访问修饰符直接放在前面，每一个都要

        public int numb = 1;

        //构造函数：与类同名，可以重载，没有返回值;在new的时候被隐式调用。

        public Example()

       {

               id = 0;

              System. out .println("consructor 1" );

       }

        public Example(int id ) //构造函数的重载

       {

               this .id = id ;

              System. out .println("constructor 2" );

       }

        public int getid()//成员函数

       {

               return this .id ;//java中this总是使用。，不再使用-》

       }

        public void setId( int id)

       {

               this .id = id ;

       }

        public void print()

       {

              System. out .println("example(id=" +id +")" );

       }

}

这是主函数，包括类的创建，带参数的构造函数的调用，类中函数的调用。

package my;

public class Hello

{

        public static void main(String[] args)

       {

               // TODO Auto-generated method stub

              Example ex = new Example(12);//创建类的对象，对象是一个指针

               //ex.setId(112345);

               ex.print();

       }

}

Java中没有析构函数，如果需要析构，则自己添加一个普通的成员函数、然后显式调用。

8.垃圾回收机制

Garbage Collection，简称GC，垃圾回收机制

Java中的对象，只需要new，不需要delete。这是因为味儿了简化程序员的负担，创建了垃圾回收机制：当一个对象不再被使用时，将被系统自动回收。

确切的说，当一个对象不再被引用（指向）后，改对象被GC回收。

比如：

public void test()

{

     Example ex = new Example();

     ex.print;

}

当test()退出后，里面创建的对象失去引用。（变量ex出了作用域）

只要对象还被引用（指向），就不会被回收。当不被指向的时候，这个对象也就不能被使用了，因为没有标识，所以此时系统就回收。

例1：

public Example test()

{

     Example ex = new Example();

     return ex;

}

public void other()

{

     Example aaa = test();

     //此时，aaa指向了这个对象，所以该对象不会被回收

}

例2：

Example ex1 = new Example();

Example ex2 = ex1;//ex1，ex2都指向了这个对象；

ex1 = null;//此时，仅有ex2指向了这个对象；

ex2 = null;//此时，没有指针指向这个对象了，他将被回收。

垃圾回收机制是一个自动机制，程序员甚至无需知道他的存在。

这也解释了为何没有析构函数：因为对象的销毁是交给GC来做的，在他的销毁这个对象的时候，不能保证析构函数还能正确运行。

总之，省去了内存管理的麻烦。

同时，也不能非常随意的new出大量对象，因为GC需要时间，不同的创建对象，会使得GC来不及回收。

9.单步调试

指定main函数：运行和调试的时候均需要指定main函数。

在同一个java项目里，可以有多个main函数；

当运行程序时，需要指定入口；

方法一：切到main函数所在源文件，右键Run As|Java Application

方法二：从工具栏里运行。

单步调试：

1.双击，下断点，debug；

2.F6，step over；

3.用鼠标直接查看变量的值。

调试状态下可以直接编辑代码，也可以进入Java状态进行编辑。

注释：选中，Ctrl+/；取消注释同上。

Java的单元测试非常方便，直接在本文件里写一个main函数，对本类进行测试即可。

10.数组对象

java里的数组是个对象，所有的对象都必须new来创建。

int[] a = new int[10];

int[] src ={1,2,3,4,5};

所有的对象都只管new，不管delete，由gc自动回收。

数组的下标从0开始，使用下标访问数组，下标不能超出范围，否则出错。

数组对象的长度信息：

int[] a = new int[10];

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

{

}

a.length即数组对象a的长度，可以看成是数组对象的一个属性，一个成员变量。

当数组元素类型为class类型时，需要注意。

//创建一个数组，含10个null指针

Example[] exs = new Example[10];

//创建10个对象

for(int i=0;i<exs.length;i++)//遍历，对数组元素进行赋值

{

     exs[i] = new Example();

     exs[i].setId(10);

}

数组的拷贝：

java里很少进行对象的拷贝，一般不需要。

数组一般有基本类型和class类型。

一般只对基本类型（char，short，int，...double...）的数组进行拷贝。

int[] src = {1,2,3,4,5,6};

int[] dst = new int[3];

System.arraycopy(src,0,dst,0,3);

拷贝函数原型：

System.arraycopy(src , srcOff , dst , dstOff , length)

参数分别是：拷贝的源，从源的第几个元素开始，拷贝的接收者，从接收者的第几个位置开始接收，拷贝的长度。

小节：

1.一定要把数组当成一个对象；

2.数组对象可以直接得到长度；

3.当数组类型为class类型时，要注意数组里存放的实质上是指针。

11.字符串对象

Java里用String类型来表示字符串；

例如：

String text = "hello,world";

String test2 = new String("hello,woeld");

//通常使用第一种方式

String是一个类，有很多函数，length

注：

1.java里的字符就是字符；

2.无论中文字符和英文字符，都按照1个计数。

3.字符串操作总是“创建一个新字符串，而不是修改原字符串”

              String text = "hello world" ;

              String text2 = new String("hello,world" );

               //长度

               int len = text .length();

               //拼接

              String text3 = text + ",123" ;

               //子串

              String text4 = text .substring(1, 4); //ell

               //替换

              String text5 = text .replace( "world", "china" );

               //将数字转成字符串

               int a = 123;

               double b = 123.456;

              String text6 = String.valueOf( a);

              String text7 = String.valueOf( b);

               //格式化字符串

              String text8 = String.format( "a=%d,b=%.2f,%s" , a ,b ,text );

               //转义字符

              String text9 = "\"ok\",he saied" ;

               //其他转义字符和C++中相同。

12.static和final

static静态：

静态即全局，在程序的任何地方都可以使用；

静态成员变量：即全局变量；

静态成员函数：即全局函数；

含义和C++里的static一样；

static成员的调用：

调用static成员变量

Const.bufsize

Const.local

调用static成员函数

Const.key();

直接用类名加点号加对象调用，不要实例化。

public class Const

{

        public static int bufsize = 1024;

        public static String local = "127.0.0.1";

        public static String key()

       {

               return "12345678" ;

       }

}

上边这个类中的成员变量和函数都是静态的，可以直接在其他的类和main函数中使用。

下边是在其他的类中对上述的类的静态成员和函数进行调用。

        public static void main(String[] args)

       {

               // TODO Auto-generated method stub

              Const. bufsize = 111;

              System. out .println("now bufsize = " + Const. bufsize);

              Const. local = "0.0.0.00" ;

              System. out .println("now local = " +Const. local);

              Const. key();

       }

final：常量，不能够被赋值

final int a = 123;

public static final int maxsize = 1999;//全局常量

13.类的继承-extends

父类

class Base

{

}

子类

class Child extends Base

{

}

父类的public/protected成员被自动继承，private成员不被继承；

被继承的成员可以被子类直接使用，和自己的成员一样。

子类可以重写父类中的函数；

所有类只能继承一个父类，不支持多重继承；

所有的类都是Object的直接或间接子类，如果一个类没有指定父类，默认继承于Object类。

所以每个类都有一些公用的，继承于Object的方法。

eg：父类：

package my;

public class Base

{

        public int id = 0;

        private String tempname = "temp" ;

        public Base(int n )

       {

               this .id = n ;

       }

        public void print()

       {

              System. out .println("base:id =" + id );

       }

        public void print( int value)

       {

              System. out .println("base:value = " + value );

       }

}

子类：

package my;

public class Child extends Base

{

        public Child()

       {       //子类中调用父类的函数，使用super即可

               super (123);//调用父类的构造函数

       }

        public void print()//重写了父类中的print函数，使用此类调用时，调用这个函数，不再调用父类的函数

       {

              System. out .println("child:..." );

               super .print();//调用父类的函数

       }

}

main函数：

package my;

public class Hello

{

        public static void main(String[] args)

       {

               // TODO Auto-generated method stub

              Child c = new Child();

               c.print();

               //父类指针指向子类对象；

              Base b = new Child();

               b.print(); //实际上调用的是子类的print

       }

}

14.抽象类：abstract class

1.类名前加abstract；

2.函数名前加abstract，不写函数体；

3.并不要求一定要加abstract的函数，部分abstract，部分非abstract也是可以的，也就是说在抽象类中就把函数实现，不用在子类中实现；

4.抽象类不能够被实例化；

为什么不能实例化？

抽象类是抽象的，只是对事物的性质进行抽象。它规定了他的子类应该有什么。具体如何实现，需要子类自己去实现。

即使语法支持实例化，也没有什么意义，它只是描述了一类事物的共性，不是具体的某个事物，他需要另外定义一个子类，实现所有规定要实现的函数，这是子类必须做到的。我们可以使用子类来实现我们的需要。

抽象类的子类定义后，可以直接使用函数补足把父类需要实现的函数写出来，然后补足函数体就可以了。方法是鼠标放在子类定义语句的子类名上，此时自动弹出需要补足的函数，单机即可；另一种方法是，在子类中右键，选择“Source”-“Override/Implement Methods...”

下例是一个抽象类，没有写实现的子类。

package my;

public abstract class Something

{

        public abstract void whatever();

}

15.类的分包管理（package）

包（package）：当一个项目中的类越来越多时，需要对其进行分包管理。

一个package实际上对应一个分级目录，就是一个队里的文件夹。

在package里添加类，选择package其实就是选择把源文件放在哪个子目录下。

如果要使用其他的包里的类，有两种方法：

1.声明类的时候使用 包名.类名 对象名 = new 包名.类名

然后就可以使用类中的函数了。

2.使用import语法，在main函数之前，加上import语句：

import 包名.类名；

使用import之后，就可以直接用类名，而不需要指定package路径了。

import的两种写法：

1.指定import一个类

import 包名.类名；

2.import该目录下的所有类

import 类名.\*；

最后，import语句不需要自己手写，只需要实例化一个其他包的类，然后把鼠标放在类名上，软件自动弹出是否要import包，然后单击就可以了。

如果一个类没有放在任何package下面，则认为其放在默认的包下面，叫做default。

包下面还可以继续建包。

16.库的使用（Jar）

开发出一个有用的类，想给别人使用，但是不想把源代码给别人，刻意把编译出的class文件打包成\*.jar格包，成为jar包。将jar包发给别人使用。避免将源代码别人看到。

1.导出一个库；

2.在项目中引入别的库；

导出jar文件：

1.右键Refresh（F5），刷新/同步目录；

2.菜单Project|Clean...，重新编译生成；

3.确认无编译错误，bin\目录下的\*.class都已生成；

4.右键点击要导出的包Export|Java|Jar file...

观察jar包中的内容：

直接把\*.jar改成\*.rar，可以发现：

1.本质是一个压缩文件；

2.内容就是bin下的相应目录和\*.class文件；

把此\*.jar文件，称为库文件（Library）。

使用jar文件：

1.在项目管理器中，项目的下边添加一个文件夹，例如叫libs；

2.将jar文件拷贝到libs目录；

3.右键点jar文件，build path | Add to build path;

之后，jar文件会出现在Referenced Libraries文件夹下，可以看到文件中包含的类和方法，可以直接使用。

下边是使用的案例，实例化类后，在点出import。

package test;

import my.Base;

public class hallo

{

        public static void main(String[] args)

       {

               // TODO Auto-generated method stub

              Base aa = new Base(123);//和在本地写的类一样使用，构造函数有参数的，不要忘记参数

               aa.print();

       }

}

管理jar文件：

右键Build Path | Configure Build Path... | Libraries

在这个界面里，可以管理本项目所引用的库；

1.添加Add JARS；

2.删除 Remove；

17.Eclipse的常用快捷操作

注释：多行：/\* \*/；单行：//；快捷键：选中要注释的内容，按Ctrl+/

格式化代码：右键Source | Format 或 Ctrl + Shift + F；将选中的代码整理成标准格式。

显示下拉列表提示：Eclipse默认是在输入点号的时候，显示下拉列表提示的。使用Alt + /可以在输入一般的时候打开提示，补足要输入的东西。

重命名：在Eclipse中可以对class，package，函数名等进行重命名，在要改名的目标上右击，Refactor | Rename；函数改名的话，先选中函数再右击。Eclipse会自动把所有该改的地方都自动修改。

刷新 | 同步：尽量在Eclipse界面下操作，不要在文件管理器（本地磁盘）中操作，否则会不同步。方法为：右键Refresh 或 F5 ；建议：拷贝文件、包、jar文件...所有操作在Eclipse下进行。

自动编译：默认编译模式是 Build AutoMatically，即保存文件时自动编译该\*.java文件，生成新的\*.class文件。

重新编译：Project | Clean...；将自动清空bin目录下的所有文件，然后重新编译。

getter/setter：成员变量的getter和setter可以自动添加，不必手工去写。右键 Source | Generate Getters and Setters...  例如，有一个变量名叫id，则自动添加函数getId()和setId();

public class getset

{

        public int id ;

        private String name ;

        public int getId()

       {

               return id ;

       }

        public void setId( int id)

       {

               this .id = id ;

       }

        public String getName()

       {

               return name ;

       }

        public void setName(String name)

       {

               this .name = name ;

       }

}

Override / Im 父类函数：重写父类的方法，就是说当你觉得父类的那个方法不适合子类使用的时候，可以重新写，以满足子类的需求，而父类的实际方法并不发生改变。方法是：右键 Source | Override / Implement ...

TODO任务：//TODO：备忘性质的注释语句；如果一个注释以//TODO开头，则它将被Eclipse列入任务列表。其作用是：提醒程序员这里需要完善，等写得差不多了，再回头来修改这里的代码。菜单Window | Show View | Tasks.

18.异常机制（1）

地位：

1.Java彻底地、完全地支持了异常机制；

2.无处不在、必须掌握。

异常机制：就是对异常情况的处理。

使用方法：先对函数添加一个声明，表示我可能会抛出异常，然后把可能发生的异常if出来，如果发生了异常，则用throw语句把异常抛出。然后在使用这个函数的时候，使用try{} catch{}语句。

try{}:正常处理流程；Catch{}：异常处理流程（错误已发生，该怎么办）

如果一切正常，则运行完try{}中的代码后，忽略catch{}，继续运行后面的代码。

如果函数有异常发生，则立即从try{}中退出，并跳转到相应的catch{}中运行。

package my;

public class Test1

{

        //求数组最大值

        public int getMax( int[] number) throws Exception     //添加声明

       {

               if (number .length == 0)

              {

                     Exception e = new Exception("长度不能为0" );

                      throw e ;

              }

               int max = number [0];

               for (int i =1;i <number .length ;i ++)

              {

                      if (number [i ]>max )

                     {

                            max = number[ i];

                     }

              }

               return max ;

       }

}

package my;

public class Hello

{

        public static void main(String[] args)

       {

               // TODO Auto-generated method stub

              Test1 t = new Test1();

               int [] data = new int[0];

               try

              {

                      int max = t .getMax(data );

                     System. out .println("result: " + max );

              }

               catch (Exception e )//如果是异常e，则执行下面的操作，如果不是，继续向下寻找。

              {

                      //异常处理

                     System. out .println("出错： " + e .getMessage());//异常的getMessage方法，就是把异常的text输出出来

              }

       }

}

throw语句用于抛出一个异常对象，该对象必须是Throwable的子类。

通常我们使用Exception类（Throwable的一个子类），或者创建一个Exception的子类，该类用于描述错误的具体信息。

Throw语句有双重作用：

Exception e = new Exception("...描述...");

throw e;

1.抛出一个异常对象；

2.退出当前函数。

19.异常机制2

异常机制与if判断：如果不用异常机制，而是使用if判断，也可以完成相同的功能。

异常机制的优点：

1. 错误的自描述
2. 增加业务流程的可读性；（看到这样的语句就知道是异常机制了）
3. 减轻程序员的负担，可以将注意力集中到正常业务流程上面。

通过下面这个例子，可以学习一下异常机智的形式。

  package my;

public class AfDate

{

        public int year, month, day;

        // 规定格式: yyyy-mm-dd

        public void parse(String dateStr) throws Exception

       {

               // 检测非法字符

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

              {

                      char ch = dateStr.charAt(i);

                      if (ch >= '0' && ch <= '9')

                     {

                     }

                      else if (ch == '-' )

                     {

                     }

                      else

                     {

                            throw new Exception( "非法字符: pos=" + i);

                     }

              }

               // 检测分隔符

               int p1 = dateStr.indexOf("-" );

               if (p1 < 0)

              {

                      throw new Exception( "格式不对: 找不到第1个横杠!" );

              }

               int p2 = dateStr.indexOf("-" , p1+1);

               if (p2 < 0)

              {

                      throw new Exception( "格式不对: 找不到第2个横杠!" );

              }

               // 检测年月日的长度

              String s1 = dateStr.substring(0, p1);

              String s2 = dateStr.substring(p1+1, p2);

              String s3 = dateStr.substring(p2+1);

               if (s1.length() != 4)

              {

                      throw new Exception( "格式不对: 年份错误!" );

              }

               if (s2.length() != 1 && s2.length() != 2)

              {

                      throw new Exception( "格式不对: 月份错误!" );

              }

               if (s3.length() != 1 && s3.length() != 2)

              {

                      throw new Exception( "格式不对: 日份错误!" );

              }

               this .year = Integer.valueOf(s1);

               this .month = Integer.valueOf(s2);

               this .day = Integer.valueOf(s3);

               // 有效性检测

               if (month <0 || month > 12)

              {

                      throw new Exception( "数据无效: 月份不能为" + month);

              }

               if (day <0 || day > 31)

              {

                      throw new Exception( "数据无效: 日期不能为" + day);

              }

       }

        @Override

        public String toString()

       {

               return year + "年" + month + "月" + day + "日" ;

       }

}

20.异常机制3

自定义异常类：继承于Exception；异常类要能够描述错误信息。比如，非法字符异常IllegalCharException应该把非法字符的位置带上。就是不同的异常抛出不同的异常类，把错误提示或操作放在异常类中，这样可以减少主程序的代码量，同时可读性更好。

例子见示例源码：

throws用于声明本函数可能产生的异常的种类，各异常类以逗号分隔。

void parse(String s) throws IllegalCharException,BadFromatException

{

}

异常的匹配：

根据抛出的异常对象的类型，从前往后匹配catch语句。

-若匹配成功，则其他的catch不在匹配；

-若全不匹配，则继承向上层函数抛出。

怎样才算是匹配成功呢？

如果抛出的异常与catch的异常相同，厚实catch的异常的子类，则匹配成功；由于Exception类是所有异常的父类，所以用这个肯定能捕获所有异常（相当于default处理）。

向上层抛出：

函数可以不处理异常，交给上层函数处理。也就是说，try{} catch{}不放在本函数中，可以放在main（）函数中。

package test3;

public class Test3

{

        public void   test1()  throws Exception

       {

            throw new Exception( "test");

       }

        public void test2()  throws Exception

       {

            test1();  // 有异常抛出，但不在本函数中处理

       }

        public static void main(String[] args)

       {

              Test3 t = new Test3();

               try

              {

                     t.test2();

              }

               catch (Exception e)

              {

                     e.printStackTrace();

              }

       }

}

finally：

无论try发生什么，finally中的代码总是执行。主要用来做一些善后清理工作，比如保存，关闭文件之类的。

1. try中正常运行；
2. try中return；
3. try中有异常抛出。

以上者三种情况，finally都会执行。

21.异常机制4

语法支持的异常：java对异常支持的很彻底，其本身自带了很多异常类，如：

java.lang.\*下面，语法自带的异常类：

数组越界：ArrayIndexOutOfBoundsException

除0异常：ArithmeticException

空指针异常：NullPointerException，对空指针进行操作的时候，会报出该异常。

当程序中有这些异常，运行时，编译器自动会报出这些异常。

由于java从语法级别对异常进行了支持，使得各种错误都能够被捕捉。

也就是说，只要在代码中加了try{}catch{}，所有的异常都可以被捕捉出来，然后执行catch中的代码。

package my;

public class Test

{

        public static void main(String[] args)

       {

               try {

                     Example ex = null ;

                     ex.setId(123);

              } catch (Exception e)

              {

                     System.out.println( "出错！" );

              }

              System.out.println( "程序退出" );

       }

}

22.内部类

写在一个类内部的类，成为内部类。

创建内部类对象，方法有两种，但是都需要先创建一个外部类对象。

第一种：先在外部类的内部使用函数创建内部类，然后在使用的时候，调用函数，返回创建的内部类对象；

第二种：直接在外部创建内部类对象，需要用到外部类的new函数。

为了可读性，我们常用第一种方法。

package test1;

public class Example

{

        // 创建内部类的对象

        public InnerObj create()

       {

              InnerObj obj = new InnerObj();

               return obj;

       }

        ///////// 内部类 ////////////

        public class InnerObj

       {

               public int id;

               public String name;

               public void test()

              {

                     System.out.println( "id=" + id + ",name=" + name);

              }

       }

}

package test1;

public class Test1

{

        public static void main(String[] args)

       {

               // 第一种：在Example的内部创建InnerObj

              Example ex = new Example();

              Example.InnerObj obj1 = ex.create();

               // 第二种：在Example的外部创建InnerObj

               // Example.InnerObj t = new Example.InnerObj(); // 语法错误

              Example.InnerObj obj2 = ex. new InnerObj();

       }

}

外部类与内部类的关系：

内部类中有一个指针指向外部类，指针名为：外部类.this；

在内部类中，可以访问自己的成员，也可以访问外部类的成员，访问外部类的成员需要加前缀：外部类.this.  ；如果外部类成员的名字和内部类不同，可省略前缀。

package test2;

public class Example

{

        private int id;

        public Example(int id)

       {

               this .id = id;

       }

        public void print()

       {

              System.out.println( "example, id=" + id);

       }

        // 创建内部类对象

        public InnerObj create()

       {

              InnerObj obj = new InnerObj();

               return obj;

       }

        ///////// 内部类 ////////////

        public class InnerObj

       {

               public int id;

               public String name;

               public void test()

              {

//                   this.id = 111;

                     Example. this .id = 222;

                     print();

              }

       }

}

public class : 外部可见；

private class：该类被隐藏在内部，外部不可用。

23.接口

使用interface定义一个接口，然后使用implements实现这个接口。

接口是一种设计方法，跟C/c++里的回调类似。

举例：有一个工具类AfSort，他为我们提供一个函数，用于任意类型的数组进行排序。

AfSort.doSort(objs, ...)

例如，我们有一个Student数组，可以用AfSort类来帮我们完成排序。

AfSort类内部使用“选择排序法”进行数组的排序。他精通排序算法，可以帮我们完成排序，但是，有一个条件：“必须告诉他，如何比较两个对象a，b的大小。”

他内部需要对所有的元素进行反复比较，然后按从小到大进行排序。他可以对任意类型的对象进行排序，可以对Student[]排序，也可以对Teacher[]进行排序。

问题：怎么告诉它“如何比较两个对象的大小呢？”传入一个接口对象作为参数即可。

接口的意义：

你要使用某个通用的工具，该工具提供了一个接口给你接入，你必须先实现这个接口，才能使用这个工具，相当于给通用的工具增加特异性，以满足你的要求。

注意事项：

1. interface里的函数声明是public的；
2. interface里一般不写成员变量，如果有成员变量，则该变量时public static final的。

语法之所以这么规定，是因为接口是一门设计技术，在interface里添加变量并没有什么实际意义，添加private函数更是没意义。

回调：

接口本质上是一种回调机制（Callback）

调用：别人给我提供一个函数，我调用它，成为Call；

回调：别人给我提供一个函数，我调用它，他在内部又调用我的函数，成为Callback。

一个类可以实现（implements）多个接口（interface）。

通用的一个工具类：

package test2;

public class AfSort

{

        // 用选择排序法对数组排序

        public static void doSort(Object[] objs, AfCompare c)//一个参数是接口

       {

               int N = objs.length;

               for (int i=0; i < N -1; i++)

              {

                      int pos = i;

                      for (int j=i+1; j< N; j++)

                     {

                            // 找到最小值的那个元素的下标

                            // 比较大小

                            if ( c.compare(objs[j] ,objs[pos]) < 0)

                           {

                                  pos = j;

                           }

                     }

                      // 把最小值调换到最前面

                      if (pos != i)

                     {

                           Object temp = objs[pos];

                           objs[pos] = objs[i];

                           objs[i] = temp;

                     }

              }

       }

}

接口：配合工具类食用更佳

package test2;

public interface AfCompare

{

        // 比较两个对象a,b的大小

        // 若a>b返回1，若a<b返回-1

        // 若a==b返回0

        int compare(Object a, Object b);

}

数组元素类：

package test2;

public class Student

{

        int id;

       String name;

       String phone;

        public Student(int id, String name, String phone)

       {

               this .id = id;

               this .name = name;

               this .phone = phone;

       }

        @Override

        public String toString()

       {

               return "(" + id + "," + name + "," + phone + ")" ;

       }

}

使用接口：

package test2;

import java.util.ArrayList;

public class Test2

{

        public void test()

       {

              Student sts[] = new Student[4];

              sts[0] = new Student(1001, "Wang" ,  "18012980989" );

              sts[1] = new Student(1032, "Li" ,    "13812542212");

              sts[2] = new Student(1029, "Shao" ,  "18092927239" );

              sts[3] = new Student(1087, "Chen" ,  "12783903839" );

//            AfSort.doSort(sts, new SortById());

               // SortByName类是一种AfCompare ( is a kind of)

              AfSort.doSort(sts, new SortByName());

              System.out.println( "done" );

       }

        public static void main(String[] args)

       {

              Test2 t = new Test2();

              t.test();

       }

        // 按ID排序

        public class SortById implements AfCompare

       {

               @Override

               public int compare(Object a, Object b)

              {

                     Student a1 = (Student) a;

                     Student b1 = (Student) b;

                      if ( a1.id > b1.id) return 1;

                      else if ( a1.id < b1.id) return -1;

                      else return 0;

              }

       }

        // 按姓名排序

        public class SortByName implements AfCompare

       {

               @Override

               public int compare(Object a, Object b)

              {

                     Student a1 = (Student) a;

                     Student b1 = (Student) b;

                      return a1.name.compareTo(b1.name);

              }

       }

}

24.匿名类

匿名内部类：简称匿名类，是内部类的一种简化写法；基本写法如下：

BaseType obj = new BaseType()

{

     //类的定义

};

匿名类也是内部类，所以语法上和内部类一样使用：

this. 访问本类对象的成员；

OUTER.this. 访问外部类OUTER对象的成员。

匿名类在java项目里被大量使用，比较简洁，推荐使用。

等下整理应用场景和示例：

25.模板类

模板类用于表示通用的数据结构和算法，例如，定义一个类AfList，表示通用的链表（适用于任意对象的类型）

定义模板类：

class AfList<T>

{

}

使用模板类：创建AfList对象时应制定参数类型

AfList<Student> stus = new AfList<Student>();

注意事项：

1. 在java里面，模板的语法不是非常必要，因为所有的对象都是Object的子类，所以直接用Object作为元素类型就可以了。
2. Java的模板支持不如C++严谨，例如，在模板类里面，不能写：T value = new T(); //语法错误    （java的模板类是属于类，C++则属于模板）
3. 似乎在任何情况下，都没有必要自己写一个模板类。

          1） java自带了ArrayList，HashMap等模板类；

          2） 直接用Object作为元素类型就行。

具体例子见示例源码。

这是一个直接使用java官方提供的ArrayList链表模板类的例子，关于链表的可以直接使用。

package test1;

import java.util.ArrayList;

public class main

{

        public static void main(String[] args )

       {

               // TODO Auto-generated method stub

              ArrayList<Student> ss = new ArrayList<Student>();

               ss.add( new Student(1001,"wang" ,"125468" ));

               ss.add( new Student(1005,"sldk" ,"1546525" ));

               for (int i = 0;i <ss .size();i ++)

              {

                     Student s = ss.get( i);

                     System. out .println("got:" + s .toString());

              }

       }

}

26.Java命令行1

27.Java命令行2

如何使用控制台运行java程序，但是很少用，如果要用，再看吧。

java的用处：

1. 网站开发；
2. Android app
3. 大数据/云计算
4. Java官方文档

对于JDK自带的类，我们到哪里查看他的帮助文档呢？

参考《java常见问题汇总》，设置Eclipse，随后即可在Eclipse里直接查看帮助。

使用鼠标查看提示，或用F1键打开帮助窗口。

以html方式查看文档

解压缩文档文件，用浏览器打开docs\api\index.html，可以直接查看帮助。

29.枚举

Java里的三种类型定义：class、interface、enum

用enum定义枚举类型，即定义一些常量。

eg：

package my;

public enum Weekday

{

       SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

    THURSDAY, FRIDAY, SATURDAY

}

使用：

package my;

public class Hello

{

        public static void main(String[] args)

       {

               // 定义一个变量

              Weekday day = Weekday.MONDAY;

               // if判断

               if (day == Weekday.SUNDAY)

              {

                     System.out.println( "Stay at home" );

              }

               // switch, 在case语句中不加Weekday前缀

               switch (day)

              {

               case SUNDAY:

               case SATURDAY:

                     System.out.println( "have a rest" );

                      break ;

               default :

                     System.out.println( "go to work" );

                      break ;

              }

               // enum -> String

              System.out.println( "the day is : " + day.toString());

               // String -> enum

               try {

                     Weekday otherDay = Weekday.valueOf( "TUESDAY" );

                     System.out.println( "convert success! other day is : " + otherDay);

              } catch (IllegalArgumentException e)

              {

                     System.out.println( "convert failed!" );

              }

               // enum -> int

               int index = day.ordinal();

               // int -> enum

              Weekday[] options = Weekday.values();

              Weekday which = options[index];

               // 取得所有可能的值

               for (Weekday d : Weekday.values())

              {

                     System.out.print(d + ", " );

              }

       }

}

最后，java中的enum语法不太严谨，替代方法：用public static final来定义常量。

30.数据类型与转换

java里的数据类型分两类：

1. 原生类型（primitive type）

     整型：byte, short, int, long ：依次是8/16/32/64 bits

     浮点型：double，float：64/32 bits

     布尔型：Boolean

     字符型：char：16 bits；用于表示字符；

     eg：

     byte b = (byte) 12; //12是int类型，所以加个强制转换

     long c = 146841256548L; //末尾加一个L

     float e = 35.3f; // 末尾加一个f，表明是float

     char f = '我'； //可以表示单个中文字符；

      char g = 'a';

      boolean h = true;

     数据类型的转换：

     整型、浮点型:

     int a = 12345;

     byte b = (byte) a; // 数据被截断

     float c = (float) 12345.006789; // 精度丢失

     int d = (int) 1234.567; // 小数部分丢失

     char型可以整型、浮点型之间强制转换

     boolean型不可以和上述类型转换

1. 引用类型（reference types）

     class、interface、enum

引用类型 reference type

下面语句:

Example ex = new Example(123);

Example ex2 = ex;

右侧 new Example(123) 是创建了一个对象

左侧变量 ex 是这个对象的引用（指针）。

英文文档一般写作

`ex` is a reference of the object

或 `ex` refers to the object

引用类型 reference type

由class, interface, enum定义的类型，都

是引用类型。（可以理解为C/C++里的指针)

Example ex = null;

则ex是空的，没有引用（指向）任何对象

引用类型 reference type

数组

Example[] ex = new Example[10];

并没有创建10个对象，只是10个指针而已，

初始值全为null。

引用类型的互转

父类指针指向子类对象

Child ch = new Child();

Base ba = ch;

隐式转换：这种情况不需要转换，自然而然地支持

void test(Object obj)

{

A pa = (A) obj;

}

以上转换有可能支持，也有可能不支持。要在运行时决

定。

如果传入的对象是A类或A的子类，则正常运行；否则会

抛出一个异常： ClassCastException 。

所以，强制转换是否出错，要看对象的实际类型。

小结

介绍了Java里的类型与转换

对象类型不能随便转，如果类型不匹配，就会抛异

常。

怎么样才叫匹配？

A a = (A) b;

b对象必须是A或A的子类的时候，才不会出错。

31.装箱拆箱（int与Integer）

包装类Wrapper Class

     在java.lang.\*下有8个类:

     Byte Short Integer Long

     Double Float

     Character

     Boolean

注意：它们是class类型，都是Object的子类

把它们叫做包装类，是对原生类型做了简单的包

装。

包装是什么意思？

class AfInteger

{

     private int value;

     public AfInteger(int v)

     {

          this.value = v;

     }

     int intValue()

     {

          return this.value;

     }

}

包装类Wrapper Class

     为什么要提供Wrapper class?直接用原生类型不

     行吗？

     有些地方要求必须用一个Object类型 ，比如：

          ArrayList<Integer>

          Hashmap<Integer, Student>

     由于模板类要求传入Object类型，不能用int，

     所以提供一个包装类Integer

装箱与拆箱

     以int和Integer的转换为例：

     1. int -> Integer (称为装箱: boxing )

          int nn = 123;

          Integer aa = new Integer(nn);

     2. Integer -> int ( 称为拆箱: unboxing )

          int vv = aa.intValue();

自动装箱/拆箱

     int nn = 123;

     Integer aa = nn;

     编译器自动创建了一个new Integer(nn)对象，称为自动装箱

     auto-boxing

     Integer aa = new Integer(100);

     int vv = aa;

     编译器自动取出aa的值，称为自动拆箱auto-unboxing。

又如：

void test(int a)

{

}

void what()

{

     Integer a = new Integer(123);

     test(a);

}

函数参数类型要求的是int，但我们传入一个Integer也是可以 的。。。虽然类型不匹配，但编译器会帮我们自动拆箱。

反之，如果参数要求Integer类型，我们传入一个int也 行。

void test(Integer a)

{

}

void what()

{

     int a = 123;

     test(a);

}

注意

1. Java里没有“重载操作符"的语法 ，我们自己写的类是不能这么自动转换的。

2. 只有原生类型primitive types和它们对应的包装类 Wrap classes之间支持。

3. 这是编译器特别加的东西，是Java语法的特例。

小结

介绍了Java里的几个包装类

介绍了装箱和拆箱这两个术语的含义

它们是为了弥补Java的语法缺陷而产生的，是编译

1. Java API

Java API: 指的是随JRE自带的类（在rt.jar里)

在基础篇里涉及的包:

     java.lang.\* (language) 语法支持包

     java.util.\* (utility) 工具包

     java.io.\* 输入/输出(文件操作)

     java.time.\* 时间/日期处理

     java.nio.\* (Network I/O) 网络字节编码

     java.text.\* 文本的格式化

在实际项目中，还需要使用第三方的API，如apache和 google的API。

java.lang.\*

     注意: java.lang不需要import，自动支持

     例如，Integer就属于java.lang

Java API 文档

帮助文档的结构：

     1. 继承层次树

     2. 功能描述

          Since：从哪一版开始引入

          See Also: 相关参考

     3. Field Summary: 成员变量概述

     4. Constructor Summary: 构造函数

     5. Method Summary: 成员函数概述

     6. Field Detail: 成员变量详解

     7. Method Detail: 成员函数详解

Java API 文档 ： Since

     注意Since的含义：

          Java至今已经有8个主版本，从1.0到1.8

          所谓的Java8就是Java1.8的简称

          如果你使用一个1.8才引入的类或函数，而JRE的版本是 1.7，则此程序无法运行。（应该要求用户安装JRE1.8）

          比如，

                java.time.Clock 就是1.8才有的

               java.lang.Integer是1.0就有的，但它的 toUnsignedString()函数却是1.8才引入的

Java API 文档 ： Deprecated

     Java默认是向前兼容的，也就是说，如果 Since标注的1.2，则表明自1.2以后的版本都 能用它。 但是，如果某个特性被标注为deprecated， 则表明将来的版本可能不再支持这个特性， 因而不推荐使用它。

     最典型的例子 java.util.Date类

//以Integer为例，讲解JavaAPI文档的使用。

与int互转

从int->Integer:

int nn = 123;

Integer aa1 = new Integer(nn);

Integer aa2 = Integer.valueOf(nn);

从Integer-> int

int vv = aa1.intValue();

与String互转

(1)Integer -> String:

int nn = 123;

String str1 = Integer.toBinaryString(nn); // 2进制

String str2 = Integer.toHexString(nn); // 16进制

String str3 = Integer.toString(nn, 10); // 10进制

(2) String -> Integer:

int vv = Integer.valueOf("A02B", 16);

注：valueOf()作转换的时候，有可能出异常 （如果字符串中出现非十六进制的字符，或是转其它进制时出现非该进制字符）

比较大小

Integer实现了Comparable 接口，所以用compareTo可以比较 两个Integer对象的大小 （文档中介绍了该类完成了那些接口）

Integer a1 = new Integer(12);

Integer a2 = new Integer(22);

int rc = a1.compareTo(a2);//大，返回1；小，返回-1；等，返回0

if(rc > 0)

     System.out.println("bigger");

else if(rc < 0)

     System.out.println("smaller");

else

     System.out.println("equal");

小结

以Integer为例，展示了Java API文档的阅读方法

1. Java API（ArrayList 链表）

ArrayList ：java.util.ArrayList ，一个封装了链表功能的类

构造一个链表

ArrayList<Student> ss  = new ArrayList<Student>();

ss.add( … );

ss.add( … );

add 添加对象

将对象添加到链表的末尾

add ( obj );

将对象插入到指定位置

add ( index, obj ); // index表示位置

get / set / remove / clear

获取

Student obj = ss.get(index);

替换

ss.set(index, obj);

删除

ss.remove(index);

清空

ss.clear();

遍例

对ArrayList对象可以按如下方式遍例

for(Student e : ss)

{

}

也可以按基本方法遍例

for(int i=0; i<ss.size(); i++)

{

Student e = ss.get(i);

}

合并

将ss2中的所有对象都添加到ss1的末尾 ：ss1.addAll(ss2);

将ss2中的所有对象都添加到ss1，插入到指 定位置 ：ss1.addAll(2, ss2);

子列 ：取得子列

List some =  ss.subList(fromIndex, toIndex);

查找

查找对象:

int index = ss.indexOf(obj);

ArrayList怎么比较元素的值？

: ArrayList会遍例所有节点对象e，当 obj.equals ( e ) 成立时，返回该节点的索引 ，所以，要重写Student类的equals方法，自定义 比较的规则。

查找

按ID比较:

public boolean equals(Object obj)

{

     Student other = (Student) obj;

     return other.id == this.id;

}

按name比较：

public boolean equals(Object obj)

{

     Student other = (Student) obj;

     return other.name.equals(other.name);

}

排序

调用sort函数，传入一个Comparator的对象

ss.sort(new Comparator<Student>(){

     @Override

     public int compare(Student o1, Student o2)

     {

          if(o1.id > o2.id) return 1;

          if(o1.id < o2.id) return -1;

          return 0;

     }

});

从低到高排序

从高到底排序

按名字排序。。。

小结

介绍了ArrayList类的使用方法

该类在项目中经常会用到。

以上源码参见示例源码。

1. Java API（HashMap 哈希表）

哈希表HashMap，就是《数据结构与算法》这 一门课程中的哈希表

哈希表HashMap中可以存储N个对象，每个对象由一个 唯一的Key值关联。

key1 -> value1

key2 -> value2

key3 -> value3

…

keyN -> valueN

(1)每个对象的key互不相同

(2)通过key值可以定位到相应的对象value

创建HashMap对象

HashMap<Integer, Student> ss  = new HashMap<Integer, Student>();

添加对象，每个对象与一个Key关联

ss.put(1001, new Student(1001, "Wang", "18012980989") );

ss.put(1032, new Student(1032, "Li", "13812542212") );

ss.put(1029, new Student(1029, "Shao", "18092927239") );

ss.put(1087, new Student(1087, "Chen", "12783903839") );

此时，ss中有4个对象，它们的Key分别为1001,1032, 1029, 1087

HashMap 哈希表

通过key可以获取相应的对象

Student value = ss.get(1029);

if(value != null)

{

}

也可以用remove来删除一个对象

ss.remove(1029);

注意

1. 在put一个对象时，如果这个key已经存在，则会 替换原有的对象（相当于 replace)

2. key的类型一般使用整型或字符串 ，如 Integer，Long, String 等 ，不能用int,long，这样语法上是不支持的

在这个例子里，如果每个同学不会重名，则也可以用

名字作为key

HashMap<String, Student> ss  = new HashMap<String, Student>();

也可以遍例HashMap里的所有对象

for(Student e : ss.values())

{

System.out.println("got: " + e );

}

HashMap的用途

     哈希表主要用于快速查找

     那么，怎么样查找才算是“快速”？

     假设有1000个Student对象，

     (1) 放在ArrayList里

          用indexOf查找，是按顺序从前往后找一遍，最多要比较1000次

     (2) 放在HashMap里

          用get获取，不是逐个比较，而是直接由Key值取得相应的 对象。。。速度快

注：哈希查找的原理见“数据结构与算法”这门课

HashMap的用途

     在Java里，HashMap还经常被用于当作一个“结构体”来 使用，把若干信息组合起来。。。( 这是很常见的写法）

     例如，

HashMap<String, Object> row  = new HashMap<String, Object>();

row.put("id", 1001);

row.put("name", "shaofa");

row.put("phone", "18012980989");

int id = (Integer) row.get("id");

String name = (String) row.get("name");

String phone = (String) row.get("phone");

小结

介绍了HashMap的方法

HashMap也是经常使用的一个类

1. 字符集与中文编码1

字符集 Charset

字符集，就是字符的集合 。比如，把英文字母、数字、标点等符号合在一起，构成一 个字符集。

ASCII: 最简单的字符集，一共收集了几十个 字符。( 该字符集的编号：ISO/IEC 646 )

该字符集中共有几十个符号，为它们一一编号，每个字符 用与一个整数对应。

例如，

'A'  65

'0'  48

称'A'的编码为65 。

ISO/IEC 646: 即ISO发布的第646号标准

其他字符集： ISO 8859-1 ：ISO发布的第8859号标准的第1 部分，涵盖了西欧的常用字符（德文、法文） 。

GB2312 中文简体国标码（汉字数：6763）

GBK 扩展中文GB编码(兼容GB2312)

GB18030 用1-4个字节编码，容量超大

BIG5 中文繁体编码

CJK 中日韩大字符集编码

Unicode 统一码，用0-0x10FFFF来映射全球各国 的语言文字

注：同一个汉字，在不同的字符集中的编码可能不同！

对于ASCII字符集，其数值范围在0-127之间，用1个 字节就可以表示。

对于GBK字符集，共约几千个字符，显然无法用一个 字节表示。。。

使用2个字节可以表示GBK字符集

例如，

'中' -> D6 90

'国' -> B9 FA

把D6 90称为 '中'的GBK编码

Unicode字符集，是把全球各国各地方，每一种 文字都合在了一起，英文，中日韩，阿拉伯文， 德法，东欧，南美等各地区的文字。

编号范围：0-0x10FFFF

Unicode用数字0-0x10FFFF来映射这些字符，最 多可以容纳1114112个字符

问题: 用几个字节来表示一个Unicode字符呢？

Unicode字符集

第一种方案：UTF-32编码 ：每个字符用一个int来表示

特点：简单，但太浪费空间

"abcd" -> 16个字节

第二种方案：UTF-16编码 ：特点: 用1~2个short来表示一个字符

第三种方案：UTF-8编码 ：用1~4个字节来表示一个字符的编码 (节省空间)

字符集与字符编码

区分 字符集 与 字符编码

字符集： Charset ，为每个字符规定一个数字编号

字符编码：Character Encoding，用几个字节来表示 这个数字编码。

非常典型的例子，Unicode字符集有3种编码方式：

UTF-32, UTF-16, UTF-8

对于字符 "ab中国"

GBK: 61 62 D6 D0 B9 FA

UTF-16: 61 00 62 00 2D 4E FD 56

UTF-8 : 61 62 E4 B8 AD E5 9B FD

特点：

GBK编码：英文占1字节，中文占2字节

UTF-16: 英文2字节，汉字一般2字节

UTF-8 : 英文1字节，汉字一般占3字节

字符集与字符编码

给定一串字节数据，如果不说明它的字符集及编码方式， 我们就无法知道它代表了哪些字符。

比如，

byte[] data = {

(byte)0x61, (byte)0x62, (byte)0xD6,

(byte)0xD0, (byte)0xB9, (byte)0xFA

};

如果知道它是GBK编码，则我们知道它是"ab中国"

如果不知道它的编码，则无法确定它到底代表什么。。。

小结

字符集的概念：为每个字符编码

介绍了几个常见的字符集：646(ASCII), 8859, UNICODE, GBK

同一个汉字，在不同的字符集中的编码可能不同。

1. 字符集与中文编码2

char 与 String ：

Java里用char表示一个字符

     char ch1 = 'a';

     char ch2 = '中';

可以把char理解为一个short型

     String str = "ab中国";

则str.length()为4,无论是英文字母还是汉 字都占一个字符。

byte[] -> String

给定一串字节数据，转化为String

例：

byte[] data = …

String str = new String(data, "GBK");

String -> byte

例：

String str = "中国";

byte[] data = str.getBytes("UTF-8");//此时，data中存的就是符合utf-8标准的编号，编号对应着“中国”二字

字符集

charset参数可以设置为： （具体编码是那个编码，可以右击.java文件，properties|resource|text file encoding）

US-ASCII

ISO-8859-1

UTF-8 （常用）

GBK (常用)

当指定UTF-16编码里需指定大小端

UTF-16BE， UTF-16LE， UTF-16

比如，"中国"

UTF-16LE: 2D 4E FD 56

UTF-16BE: 4E 2D 56 FD

当指定为UTF-16时，则添加2个前缀字节，用于 表示大小端。

FE FF 4E 2D 56 FD (FE FF标识为大端)

FF FE 2D 4E FD 56 (FF FE标识为小端)

所以，当拿到一串UTF-16编码的字节时，需要知

道它是LE还是BE (little-endian, big-endian)

默认字符集

如果不指明charset参数，则使用平台默认的字符集

( platform default charset)

// byte[] -> String

String str = new String(data);

// String -> byte[]

byte[] data = str.getBytes();

检查默认字符集：

Charset cs = Charset.defaultCharset();

System.out.println("默认字符集: " + cs.toString());

注意

源文件\*.java本身也有一个编码问题 ,但\*.java在编译之后，会统一转成Unicode

注意源代码中文乱码的问题。

查看\*.java的编码：右击.java文件，properties|resource|text file encoding

小结

介绍了如何将String和byte[]数组互相转换

1. 字符集与中文编码3

\* 提示

本节不是特别重要。有一个印象就可以了。 上一节的最后，已讲了这个问题。

乱码问题

展示：拿到一个源文件Hello.java，用记事 本打开没有发现问题，但编译运行后显示乱 码。。。

若一个\*.java源文件是以UTF-8编码存储的，而平台默认是GBK，

命令行运行时，添加如下参数 (注意写成UTF8, 不是UTF-8)，告知编译器编码格式方可。

javac -encoding UTF8 my\Hello.java

java my.Hello

也就是说，javac在编译的时候，需要知道源文件的 编码格式。默认按平台的default charset进行编译。

javac会把双引号中的字节转成Unicode字节，存在 \*.class文件里。

在Eclipse中设置编码

在Eclipse中打开一个源文件，如果出现中文乱码， 则需要重新设置该文件的编码属性。

右键 properties | Text file encoding

小结

介绍java源文件的编码，可由此可能导致的问题。

1. 时间处理1

相关API

在Java API用以下class/package来处理时间：

     java.util.Date 大部分函数已经deprecated

     java.util.Calendar; 日期操作

     java.text.SimpleDateFormat 格式化字符串

     java.time.\* 在Java8新加入的包

当前时间

long now = System.currentTimeMillis();

取得当前时间

返回自 1970-01-01 00:00:00.000 以来的毫 秒数。(此时间点称为元点 Epoch )

Date d = new Date();

创建一个Date对象，初始值为当前时间

与long型互转

long t = d.getTime(); // Date -> 毫秒值

d.setTime(t); // 毫秒值 -> Date

由于Date类的基本全部函数都是deprecated，可以认为该 类基本没用了。。。还不如直接用一个long型 ，但有些第三方API接口仍然使用Date类。。。所以没有 obsolete（废弃），只是deprecated（不赞成） 。。

Date <-> String

使用java.text.SimpleDateFormat

// 格式 2016-02-17 11:23:56

SimpleDateFormat fmt

= new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

Date now = new Date();

String strNow = fmt.format(now);

// String -> Date

Date when = fmt.parse("2012-03-12 11:00:24");

Calendar类主要用于毫秒值与年月日时分秒之间 的转换。

Date <-> 年月日时分秒

比如，用long型整数记录了一个时间，但这个时 间到底是哪一天。。。需要转成一个用户可读的 “年月日时分秒” ；

再比如，已经年月日时分秒，如何转成一个毫秒 值。。

// 创建Calendar对象, 初始值为当前时间

Calendar cal = Calendar.getInstance();

//Calendar的构造函数是protected，所以不能new，只能调用其提供的函数新建一个对象。

// Date -> Calendar

cal.setTime ( d );

// Calendar -> Date，毫秒值转化成时间格式

Date d = cal.getTime();

Calendar

取得年月日时分秒

int year = cal.get(Calendar.YEAR);

int month = cal.get(Calendar.MONTH);

int day = cal.get(Calendar.DAY\_OF\_MONTH);

int hour = cal.get(Calendar.HOUR\_OF\_DAY);

int minute = cal.get(Calendar.MINUTE);

int second = cal.get(Calendar.SECOND);

注意，月份是从0开始的,当前是2016-2-17,则month的 值为1 （ Calendar.FEBRUARY )

Calendar

设置年月日时分秒

cal.set(Calendar.YEAR, 2016);

cal.set(Calendar.MONTH, Calendar.FEBRUARY);

cal.set(Calendar.DAY\_OF\_MONTH, 17);

cal.set(Calendar.HOUR\_OF\_DAY, 11);

cal.set(Calendar.MINUTE, 45);

cal.set(Calendar.SECOND, 0);

注意，月份是从0开始的,当前是2016-2-17,则month的 值为1 （ Calendar.FEBRUARY )

日期算术

// 22天之后

cal.add(Calendar.DAY\_OF\_MONTH, 22);

// 6个月之前

cal.add(Calendar.MONTH, -6);

Calendar类内部会计算闰年、月份这些信息

1. 时间处理2

这节课简要介绍 java.time.\*包的内容，使 用这个包也可以实现时期和时间的处理。

但这个包用起来较为复杂。

几个术语

Epoch 时间元点 : 1970-01-01 00:00:00.000

ZoneOffset时区：从-12到+14，比如中国处于+8区

Instant时刻 ：自Epoch以来的时间点，以毫秒计量

LocalDateTime本地日期时间：不带Zone信息

LocalDate: 本地日期

LocalTime: 本地时间

ZoneOffset ：

构造时区对象

ZoneOffset zone = ZoneOffset.ofHours(8);

LocalDateTime

当前系统时间

LocalDateTime dt1 = LocalDateTime.now();

指定年月日时分秒 2016-2-17 17:50:00

LocalDateTime dt2

= LocalDateTime.of(2016,2,17,17,50,0);

转成秒值

ZoneOffset zone = ZoneOffset.ofHours(8);

long tms = dt2.toEpochSecond(zone);

转成instant

Instant ins = dt2.toInstant(zone);

LocalDateTime

时间算术

// 120天以后

LocalDateTime dt3 = dt2.plusDays(120);

// 50分钟以前

LocalDateTime dt4 = dt2.minusMinutes(50);

小结

这个包设计的有点复杂。。。

由于大多数API使用的还是Date类型，所以还是以上节课为主。

1. 文件和目录操作

File

java.io.File 用于表示一个文件或目录的路径名(pathname)

例如，

File f = new File("c:/example/123.txt");

先判断它是否存在

if(f.exists())

然后再看它是一个目录，还是一个文件

if(f.isDirectory()) …

else if(f.isFile()) …

注意

File只是包装一个路径名，可以认为它内部只有一个 字符串。

File f = new File("c:/abc.txt");

这一句并不对文件系统里的文件进行操作 ，即使创建1000个File对象，对文件系统也没有任何影 响。

f.exists() 这个函数会检查文件系统 ，也就是说，在调用File的函数时才会调用文件系统 文件属性 。

如果f.exists() && f.isFile() ，则可以查看文件的属性

f.canExecute() 是否可执行

f.canRead() 是否可读

f.canWrite() 是否可写

f.isHidden() 是否为隐藏文件

f.length() 文件的大小 (单位: byte)

f.lastModified() 最后修改时间 (返回毫秒值)

f.getName() 文件名 ( 例如, "abc.txt"，不含路径)

文件操作

如果f.exists() && f.isFile() ，则可以执行以下操作

f.delete() 删除文件 （但不一定成功删除）

f.renameTo() 重命名 (相当于move）

f.setExecutable() 设置可执行属性

f.setReadable() 设置可读属性

f.setWritable() 设置可写属性

f.setLastModified() 更新最后修改时间

得到文件所在的目录

File f = new File("c:/example/123.txt");

File location = f.getParentFile();

注：getParentFile()只是按路径推算上级目录，并不 要求f这个文件是存在的

f.getName() 文件名 ( 例如, "abc.txt"，不含路径)

也不与文件系统打交道 ，只是按照路径名推算了一下。

目录操作

如果d.exists() && d.isDirectory(), 则可以对此 目录执行以下操作

d.renameTo() 文件夹改名

d.delete() 删除文件夹 (但不一定能成功删除)

其他的可读、可写、可执行属性，要看文件系统本

身是否支持。。。

如果目录不存在， 使用下面的函数自动创建：

d.mkdirs(); // 创建分级路径

注：

File d = new File("c:/example/doc/en/1");

d.mkdirs();

自动将父级目录一并创建，不用担心父级目录不 存在。

目录操作

列出子项，将子文件、子文件夹一并列出

File[] children = d.listFiles();

for(File child : children)

{

     System.out.println("got: " + child);

     if(child.isDirectory())

     {

     }

     else if(child.isFile())

     {

     }

}

设置过滤器，例如，只列出 \*.mp4

FileFilter filter = new FileFilter(){

     @Override

     public boolean accept(File pathname)

     {

          if(pathname.getName().endsWith(".mp4"))

               return true;

          return false;

     }

};

File[] children = d.listFiles(filter);

相对路径

File d1 = new File("c:/example/aaa");

File d2 = new File(d1, "123");

则d2是一个相对路径，表示d1下的123.txt

File d3 = new File(d1, "../bbb")

则d3表示 c:/example/aaa/../bbb

File d1 = new File("c:/example/aaa");

File d2 = new File(d1, "../bbb");

File a1 = d2.getAbsoluteFile();

try

{

     File a2 = d2.getCanonicalFile();

} catch (IOException e){}

其中，

absolute的结果为c:\example\aaa\..\bbb (非标准形式)

canonical的结果为C:\example\bbb (标准形式)

小结

介绍了File的用法

注意：

1.本身只表示一个路径字符串(pathname)

2.的相关函数会与文件系统打交道

3. File既可以用于文件操作，也可用于目录操作

1. 文件读写1

写文件

使用FileOutputStream可以把数据写入文件

什么叫数据? 数据就是一串字节

例如，

byte[] data = {1,2,3,4,5,6};

打开文件，写入数据。。。用UltraEdit查看文件内容。。。

FileOutputStream fout  = new FileOutputStream(filename);

fout.write(data);

fout.close();

追加模式写入

在构造时第2个参数置为true，表示append （附加）

new FileOutputStream(filename, true);

写入数组的一部分

fout.write(data, 2, 3);

//则写入 buf[2], buf[3], buf[4]

读文件

使用FileInputStream可以从文件中读取数据

// 设置一个大的缓冲区

byte[] buf = new byte[1024];

// 打开文件，把数据读到缓冲区里

FileInputStream fin = new FileInputStream(filename);

count = fin.read(buf, 0, 1024);

fin.close();

如果文件很大，则需要分段读取

while(true)

{

int n = fin.read(buf,0,1024); //每次最多读1024

if(n <= 0) break;

… 处理数据 …

}

数据的编码

如果我们的数据类型是Integer, String，则需要将数据转成byte[]，然后才能写入到文件

例如，对于String

写：先把String->byte[]，然后写入文件

读：从文件读出到byte[]，然后把byte[]->String

那么，其他类型怎么转成 byte[] 呢？（见下源码有应用）

byte [] data = str .getBytes( "GBK");

String str2 = new String(buf , 0, n , "GBK" );

注：FileInput/OutputStream只负责读写byte[]，

至于怎么将byte[]转换成Java程序中的各种对象， 是另外的事情。

小结

介绍了如何向文件中读写byte[]型数据

要将自己的类型转成byte[]，再写到文件里去。反之，

public static void write( byte[] data)

       {

               // 文件名

              File filename = new File("c:/example/haha2" );

               // 创建文件, 写入数据

               try {

                     FileOutputStream  fout = new FileOutputStream(filename );

                      fout.write( data);

                      fout.close();

              }

               catch (IOException e )

              {

              }

       }

        public static int read( byte[] buf)

       {

               // 缓冲区

               int count = 0;

               // 文件名

              File filename = new File("c:/example/haha2" );

               try

              {

                     FileInputStream fin = new FileInputStream(filename );

                      count = fin.read( buf);

                      fin.close();

              } catch (IOException e )

              {

              }

               return count ;

       }

        public static void main(String[] args)

       {

              String str = "你好，中国" ;

               try

              {

                      byte [] data = str .getBytes( "GBK");

                      write( data);

              } catch (UnsupportedEncodingException e )

              {

                      e.printStackTrace();

              }

               byte [] buf = new byte[1024];

               int n = read( buf);

               try

              {

                     String str2 = new String(buf , 0, n , "GBK" );

                     System. out .println("got:" + str );

              } catch (UnsupportedEncodingException e )

              {

                      // TODO Auto-generated catch block

                      e.printStackTrace();

              }

              System. out .println("exit" );

       }

1. 文件读写2

int <-> byte[]

本节重点研究各种数据与byte[]之间的转换

现有

byte[] buf = new byte[4];

int a = 0x12345678;

怎么把a的值放到buf里去呢？

使用java.nio.ByteBuffer类可以完成int到 byte[]的转换

byte[] buf = new byte[4];

int a = 0x12345678;

ByteBuffer enc = ByteBuffer.wrap(buf);     //wrap:包；掩护

enc.putInt(a);

可以发现，buf中的4个字节为: 12 34 56 78

int <-> byte[]

反过程：把byte[]->int

ByteBuffer dec = ByteBuffer.wrap(buf);

int value = dec.getInt();

从而实现了把一个int写入到文件，再读取出 来的过程。。。

文件读写示例

写过程：

byte[] buf = new byte[4];

int a = 0x12345678;

ByteBuffer enc = ByteBuffer.wrap(buf);

enc.putInt(a);

int size = enc.position(); // 字节数

write(buf, 0, size);

文件读写示例

读过程：

byte[] buf = new byte[1024];

int count = read(buf);

ByteBuffer srcbuf

= ByteBuffer.wrap(buf, 0, count);

int n = srcbuf.getInt();     //将读到的数据转换为int型

int与String混编

例如，有3个数据，

int id = 201601;

String name = "邵发";

String phone = "18600012345";

byte[] buf = new byte[1024];

试着将它们编入一个buf中。。。

ByteBuffer dstbuf = …

dstbuf.putInt(id);

dstbuf.put???

可以先把String转成byte[]

byte[] s1 = name.getBytes("GBK");

byte[] s2 = phone.getBytes("GBK");

dstbuf.put(s1);

dstbuf.put(s2);

似乎没问题，然而，在读取的时候如何读出name和 phone这两个字段？

解决方案：对于字符串采用Length + Content编码

byte[] s1 = name.getBytes("GBK");

enc.putShort(s1.length); //头2字节表示长度

enc.put(s1); // 然后再把内容编进去

读的时候：

short length = dec.getShort(); // 头2字节表示长度

dec.get(output, 0, length);

示例源码节选：

public static void main(String[] args)

       {

               int id = 201601;

              String name = "邵发" ;

              String phone = "18600012345" ;

               byte [] buf = new byte[1024];

               // 写入

              ByteBuffer dstbuf = ByteBuffer. wrap( buf);

               dstbuf.putInt( id);

              AfByteBuffer. putString( dstbuf, name);//AfByteBuffer:工程中的另一个类，其函数putString的作用是把第二个（字符串型）参数转化为字节型，存入第一个参数中。具体见示例源码。

              AfByteBuffer. putString( dstbuf, phone);

               int total = dstbuf .position();

               write( buf, 0, total);

               // 读出

               int n = read( buf);

              ByteBuffer srcbuf = ByteBuffer. wrap( buf, 0, n);

               int a1 = srcbuf .getInt();

              String a2 = AfByteBuffer.getString( srcbuf);

              String a3 = AfByteBuffer.getString( srcbuf);

              System. out .println("exit" );

       }

ByteBuffer.wrap函数详解：

public static [ByteBuffer](file:///\\java\nio\ByteBuffer.html) wrap(byte[] array,     //被放在缓冲区中的数组，另外有一个wrap函数，只有这一个参数；

                              int offset,       //缓冲区从数组的这个地方开始；

                              int length)       //从数组中取出的子数组长度，和offset共同决定着数组的长度。

buffer : 缓冲区；

wrap：包；掩护。

函数作用：Wraps a byte array into a buffer.

在ByteBuffer中每读取一次，相应的指针就后移一次，所以很容易实现分段读取。

例如，getShort()函数，只读取两个字节，并返回读取到的内容。

小结

介绍了int型和String型的字节编码，以及混编的方

法，其他类型与此类似。

1. 文件读写3

ArrayList<Student>

上节课介绍了如何读写一个学生的信息数据

putInt(id)

putString(name)

putString(phone) …

现有一个链表ArrayList<Student>，如何将 多个Student对象都写入到文件里呢？

仍然使用 Length + Content编码

ArrayList<Student> ss = …

将ss[0] 编码，得到数据D0,长度N0

将ss[1] 编码，得到数据D1,长度N1

…

依次写入文件:

N0 D0 N1 D1 N2 D2 …

其中，

Ni, 2字节，表示后续的数据长度

Di, Student对象的字节编码，长度为Ni

演示

在Eclipse中讲解

小结

再次运用 Length + Content 编码的方法，将链表里

的多个对象存储到文件中。

实际项目中，可以根据实际情况自行设计，只要能

满足“能写能读”的要求就行。

本节所介绍的“字节编码”技术，在其它场合也会用

1. 文本文件的按行读取

文本文件

什么叫文本文件？

没有严格的定义。一般认为如果文件的都是 可见字符，那就是文本文件。

注：文本文件与二进制文件并没有明确的区别

注：并不是说以.txt为后缀，就叫文本文件。

按行读写

什么叫“行”？

一般以 \n 或 \r 或\r\n作为一行的分隔符。

按行读取：就是按顺序读取，当读取到分隔符时， 将一行的内容返回。

注意：一行内容不能太长。。。否则没法按行读 取。。比如，完全没有分隔符，一行到底。。

BufferedReader      //带缓冲区的阅读器

java.io.BufferedReader用于按行读取

创建一个BufferedReader

BufferedReader reader  = new BufferedReader(…, size);

读取一行

while(true)

{

String nextline = reader.readLine();

if(nextline == null) break;

… 处理这一行字符串 …

}

BufferedReader的输入源

要点：构造一个Reader的对象作为构造函数 的参数，表示数据源。(Reader是一个基类，他有很多常用的子类，常用以下两种作为BufferedReader的数据源)

第1种办法：已知File

File -> FileReader -> BufferedReader

File source = …

FileReader m = new FileReader(source);

BufferedReader reader = new BufferedReader(fr);

// 以File作为输入源  File -> FileReader

        public static void test1(File source) throws Exception

       {

              FileReader m = new FileReader(source);

              BufferedReader reader = new BufferedReader(m);

               while (true )

              {

                     String nextline = reader.readLine();

                      if (nextline == null) break;

                     System.out.println( "got: " + nextline);

              }

              reader.close();

       }

第2种办法：已知InputStream

InputStream ->InputStreamReader-> BufferedReader

InputStream source = …

InputStreamReader m

= new InputStreamReader(source,"GBK");

BufferedReader reader

= new BufferedReader(fr);

// 以InputStream作为输入源  InputStream -> InputStreamReader

        public static void test2(InputStream source) throws Exception

       {

              InputStreamReader m = new InputStreamReader(source, "GBK" );

              BufferedReader reader = new BufferedReader(m);

               while (true )

              {

                     String nextline = reader.readLine();

                      if (nextline == null) break;

                     System.out.println( "got: " + nextline);

              }

              reader.close();

       }

使用System.in

System.in就是控制台输入，它本身也是InputStream类型，因此适用 于第2种方法。

InputStreamReader m = new InputStreamReader(System.in);

BufferedReader reader = new BufferedReader(m);

while(true)

{

// 用户在控制台输入，输入回车完成输入

String nextline = reader.readLine();

if(nextline == null) break;

}

控制台输入都是下面这个套路，用到的时候照搬就行了。

public static void test3() throws Exception

       {

              InputStreamReader m = new InputStreamReader(System.in);

              BufferedReader reader = new BufferedReader(m);

               while (true )

              {

                      // 输入提示

                     System.out.print( "> " );

                      // 用户输入

                     String nextline = reader.readLine();

                      if (nextline == null) break;

                      // 终止条件

                      if ("exit" .equals(nextline))

                     {

                           System.out.println( "good bye" );

                            break ;

                     }

                      // 处理用户输入

                     System.out.println( "handle command: " + nextline);

              }

              reader.close();

       }

这是以上三个函数的调用：

public static void main(String[] args)

       {

               try

              {

                      // 以File作为输入源

                      // test1(new File("c:/example45.txt"));

                      // 以InputStream作为输入源

                      //test2(new FileInputStream("c:/example45.txt"));

                      // 以控制台为输入

                     test3();

              } catch (Exception e)

              {

                      // TODO Auto-generated catch block

                     e.printStackTrace();

              }

       }

}

注：Eclipse自带控制台的输入支持得不好，请用Windows控制台 测试。java –cp bin my.Hello

注意

如果一行太长，可以调整一下内部缓冲区的 长度。

如果一行的长度为0（空行），则返回String 的长度为0，相当于 str = "";

小结

按行读取只针对文本文件，此技术在工程用到的不多。

本节课重点是展示java.io.\*下面的几个类的转换。

java.io.\*下面的几个重要的基类：

File

InputStream/OutputStream

1. \* 配置文件

配置文件：用于存放程序的运行参数

在Java里，通常使用两种格式的配置文件：

(1)\*.properties

(2)\*.xml

本节讲解properties的用法 ：

例子

有一个应用程序，功能是打印温度转换表。

“摄氏温度” – “华氏温度”

允许用户定义以下参数：

起点：range.from , 整数

终点: range.to , 整数

标题: title，字符串

Properties

使用java.util.Properties类可以操作 \*.properties格式的配置文件

一、创建 properties文件

放在src\下

二、加载 properties文件

Properties props = new Properties();

props.load(…); // 加载

String title = props.getProperty(..,..)

例子

config.properties

内容如下：

title=Celsius-Fahrenheit

range.from=1

range.to=20

配置文件放在哪里？

1. 指定绝对路径

props.load( new FileReader("c:/config.properties"))

2. 相对路径

以java执行的当前路径为起点，指定相对路径

props.load( new FileReader("bin/config.properties"))

3. 作为资源文件 （主要使用此方法，前两个方式有缺陷）

放在class的根目录下 （bin文件夹）

InputStream is = Hello.class.getResourceAsStream("/config.properties");

package my;

import java.io.IOException;

import java.io.InputStream;

import java.util.Properties;

public class Hello

{

    public static void main(String[] args)

    {

        Properties props = new Properties();

        try

        {

            // File name = new File("D:/JavaProjects/HelloWorld/bin/config.properties"); //绝对路径

//            File name = new File("bin/config.properties"); //相对路径

//            InputStream is = new FileInputStream(name); //无论是绝对路径，还是相对路径，都需要这一步。

            InputStream is

                = Hello.class.getResourceAsStream("/config.properties");

            props.load(is);

            is.close(); // 用完关闭这个流

        } catch (IOException e)

        {

            e.printStackTrace();

        }

        // 取得配置参数

        String title = props.getProperty("title", "----");

        String rangeFrom = props.getProperty("range.from", "1");

        String rangeTo = props.getProperty("range.to", "50");

        int from = Integer.valueOf(rangeFrom);

        int to = Integer.valueOf(rangeTo);

        // 显示

        AfTemp t = new AfTemp(title, from, to);//另外一个类，用于摄氏温度和华氏温度的换算，参见源码

        t.display();

    }

}

注意

1. 放在src的文件，会被Eclipse自动拷贝到bin目 录下

2. \*.properties中一般不设置中文 按Java官方的规定，中文需要转义。（直接输入中文，Eclipse会自动完成转义）

3. 更多地以\*.xml作为配置文件

资源文件

随程序发布的配置文件、图片文件等，一般

放在class根目录下。

资源文件的加载：

InputStream res = Hello.class.getResourceAsStream(…)

小结

介绍了配置文件的种类， Properties配置文件的使 用方法。

介绍了资源文件的加载方法。

位置：放在class根目录下

加载方法：getResourceAsStream()

1. 反射机制(Reflection) (1)

反射 Reflectoin

反射机制是Java的一个非常重要的机制。一 些著名的应用框架都使用了此机制。

例如，

struts, spring, hibernate  android app的界面设计器 …

在设计应用框架时一般都要使用反射，它是 Java的一个非常优越的特性。

从Class说起

java.lang.Class

它是Java语法的一个基础类，用于描述一个 class对象。 在文件系统中,class以文件的形式存在 Student.class 。在运行时的JVM中，该\*.class文件被加载到内存中成为一个 对象，对象的类型就是java.lang.Class

例如，

Class cls = Student.class;

System.out.println("Name: " + cls.getName());

其中，cls这个对象就是这个Class的描述。

再例如，

Student obj = new Student();

Class cls = obj.getClass();

其中，obj是一个对象，obj.getClass()则是获取它的Class 描述。

Class有什么用

用于判断一个运行时对象的类型

public static void test1(Object obj)

{

     Class cls = Student.class;

     if(cls.isInstance(obj))

     {

     }

}

其中，cls.Instance(obj)意思是判断obj是否为my.Student 的一个实例。

另一种写法，也可以判断一个运行时对象的类型

public static void test2(Object obj)

{

     String clsName = obj.getClass().getName();

     if(clsName.equals("my.Student"))

     {

     }

}

其中,obj.getClass()可以得到对象obj的具体类型

应用展示

让一个对Student对象可以与String和Integer进行比较。。。

public boolean equals(Object obj)

{

     // 与一个String对象比较

     if(String.class.isInstance(obj))

     {

          String other = (String)obj;

          return other.equals(this.name);

     }

     // 与一个Integer对象比较

     if(Integer.class.isInstance(obj))

     {

          Integer other = (Integer)obj;

          return this.id == other;

     }

     return false;

}

以上是Student类中equals函数的重写。

小结

初步介绍了Class的概念，并介绍了它的简单使用。

Class是一个类型的描述，一个\*.class文件加载到JVM

1. 反射机制(Reflection) (2)

Reflection

给定一个\*.class文件中，我们可以得到以下信息：

类名 (含package路径)

函数 (名称，参数类型，返回值）

域 （名称，类型）

实现的接口 (interfaces)

… …

使用java.lang.reflect.\*下的类来实现。。。

注意：不需要源文件，只需要\*.class文件

动态加载一个\*.class文件到内存:

// 加载my/Student.class

Class cls = Class.forName("my.Student");

// 获取函数列表

Method[] methods = cls.getMethods();

// 获取成员变量列表

Field[] fields = cls.getFields();

也就是说，虽然我们不知道Student类的代码，但是这个 class文件本身可以反映(reflect)出这些信息。。。

结论：通过Reflection机制，我们可以直接 从class文件反推出它有哪个成员变量、有哪 些函数。

例1 遍历Method

已经函数名，找到对象的 Method

Class cls = …

String methodName = "setId";

// 获取所有Method列表，顺序比对

Method[] methods = cls.getMethods();

for(Method m : methods)

{

     if(m.getName().equals(methodName))

     {

          break;

     }

}

例2 查找Method

已经函数名，参数列表，寻找Method

Class cls = …

String methodName = "setId";

Class[] parameterTypes = { int.class };

Method m = cls.getMethod(methodName,

parameterTypes);

例3 Reflection的运用

// 创建一个实例, 要求有一个不带参数的构造函数

Object obj = cls.newInstance();

// 找到method

Class[] parameterTypes = { int.class };

Method m1 = cls.getMethod("setId", parameterTypes);

// 调用method

Object[] paramters = { 123 };

m1.invoke(obj, paramters);

更多细节 Class Loader

Class.forName(…) 可以加载一个class文件。。

（1）由谁负责加载？

由Class Loader负责加载，具体地讲，JVM提供了一 个内置的 bootstrap class loader。

（2）从哪里加载？

从classpath下寻找class，以及该class里面import 的class

小结

1. Reflection: 从\*.class文件中，可以reflect得到它 的描述信息：类名，函数名等

2. 通过Class，可以创建一个实例instance

3. 通过Class，可以找到它的某个Method，进而就 可以调用这个函数。

4. 需要知道Class Loader的存在及其作用

49. 反射机制(Reflection) (3)

Reflection 应用框架

一般地，当设计一个应用框架时才有可能用 到Reflection技术。

例如，著名的Struts Spring Hibernate框架 (角色：设计师）

AfShell: Command Ui Framework

AfShell: 一个自定义的Framework，使用这个

Framework可以很方便的写出基于命令行界面的应用 程序。

添加AfShell框架支持

（1）加入afshell.jar到BuildPath

（2）添加afshell.properties

（3）在main中调用 AfShell.start()

完成！

演示。。。

AfShell: Command Ui Framework

添加命令处理

（1）自定义Action类

此类必须有一个函数public int execute()

（2）配置afshell.properties

login=my.LoginAction

演示。。。

添加命令行参数的支持

例如，用户输入

login username=shaofa password=123456

（1）添加成员变量

username

password

（2）添加setter

setUsername()

setPassword()

则AfShell框架会自动的把参数值传进来，然后再调用 execute()函数。。。

设计方法

那么，这样的一个Framework该如何实现？

例如，用户输入了

login username=shaofa password=123456

（1）得到命名名 "login"

（2）通过配置文件，找到my.LoginAction

（3）加载my.LoginAction，创建一个实例

（4）根据username=shaofa,找到setter函数setUsername，

调用此函数

（5）根据password=123456,找到setter函数并调用

（6）调用excute()函数，得到返回值

完成！

小结

介绍一个应用框架AfShell的使用方法

如果要使用这个框架，并不需要知道Reflection技术

如果自己设计一个框架给别人使用，则需要精确掌 握和运用Reflection技术。

请试着使用Reflection技术，自己来实现这个框架。

1. 反射机制(Reflection) (4)

AfShell: Command Ui Framework

那么，这样的一个Framework该如何实现？

例如，用户输入了

login username=shaofa password=123456

（1）得到命名名 "login"

（2）通过配置文件，找到my.LoginAction

（3）加载my.LoginAction，创建一个实例

（4）根据username=shaofa,找到setter函数 setUsername，调用此函数

（5）根据password=123456,找到setter函数并调用

（6）调用excute()函数，得到返回值

AfShell.java

package af;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.lang.reflect.Method;

import java.util.ArrayList;

import java.util.Properties;

public class AfShell

{

    private Properties props = new Properties();

    private AfShell()

    {

        // 加载 afshell.properties, 第46课

        props = new Properties();

        try

        {

            InputStream is

                = AfShell.class.getResourceAsStream("/afshell.properties");

            props.load(is);

            is.close(); // 加载完后关闭此文件

        } catch (Exception e)

        {

            System.out.println("请提供afshell.properties文件,否则无法运行AfShell!");

            e.printStackTrace();

        }

    }

    // 启动 AfShell 命令行界面框架

    public static void start()

    {

        AfShell shell = new AfShell();

        System.out.println("------- AfShell Command Ui Framework ------");

        System.out.println("作者: 阿发你好");

        System.out.println("官网: http://afanihao.cn");

        System.out.println("此演示程序是 Java学习指南(基础篇) 的官网资料，最新版本请到官网下载!");

        System.out.println("\n");

        try

        {

            shell.process();

        } catch (Exception e)

        {

            e.printStackTrace();

        }

    }

    // 输入界面

    private void process() throws Exception

    {

        InputStreamReader m = new InputStreamReader(System.in);

        BufferedReader reader = new BufferedReader(m);

        // 读取控制台的输入, 第45课

        while(true)

        {

            // 输入提示

            System.out.print("# ");

            // 用户输入

            String nextline = reader.readLine();

            if(nextline == null) break;

            // 去除左右的空白

            nextline = nextline.trim();

            // 终止条件

            if("exit".equals(nextline))

            {

                System.out.println("good bye");

                break;

            }

            if(nextline.length() == 0)

            {

                continue;

            }

            // 处理用户输入

            try{

                handleCommand(nextline);

            }catch(Exception e)

            {

                e.printStackTrace();

            }

        }

        reader.close();

    }

    // 处理命令

    private void handleCommand(String cmdline) throws Exception

    {

        Command cmd = new Command();

        cmd.parse(cmdline);

//        System.out.println("名字: " + cmd.name);

//        for(Pair p : cmd.arguments)

//        {

//            System.out.println("参数: " + p.k + "=" + p.v);

//        }

        /\* 从afshell.properties读取该命令的配置 \*/

        String clsName = props.getProperty(cmd.name, "").trim();

        if(clsName.length() == 0)

        {

            throw new Exception("在afshell.properties中找不到有关 "+cmd.name + " 的配置!");

        }

        /\* Load Class \*/

        Class cls = Class.forName(clsName);

        Object obj = cls.newInstance();

        Method[] methods = cls.getMethods();

        /\* 使用setters来设置其属性 \*/

        for(Pair p : cmd.arguments)

        {

            String key = p.k;

            String value = p.v;

            // 标准化setter的名字，例如 username -> setUsername

            char firstChar = Character.toUpperCase(key.charAt(0));

            StringBuffer strbuf = new StringBuffer("set" + key);

            strbuf.setCharAt(3, firstChar);

            // 找到对应的Method

            String methodName = strbuf.toString();

            Method method = null;

            for(Method m: methods)

            {

                if(m.getName().equals(methodName))

                {

                    method = m;

                    break;

                }

            }

            if(method == null)

            {

                throw new Exception("找到对应的setter: " + methodName);

            }

            // 查看参数类型

            Class[] pt = method.getParameterTypes();

            // 执行setter

            Object p0 = createParameterObject (pt[0], value);

            Object args[] = { p0 };

            method.invoke(obj, args);

        }

        /\* 执行execute 函数 \*/

        Method execute = cls.getMethod("execute", null);

        Integer returnValue = (Integer)execute.invoke(obj, null);

        System.out.println("------> return value = " + returnValue + " <--------");

    }

    private Object createParameterObject(Class theType, String value)

    {

        String t = theType.getName();

        if(t.equals("int") || t.equals("java.lang.Integer"))

        {

            return Integer.valueOf(value);

        }

        if(t.equals("byte") || t.equals("java.lang.Byte"))

        {

            return Byte.valueOf(value);

        }

        if(t.equals("short") || t.equals("java.lang.Short"))

        {

            return Short.valueOf(value);

        }

        if(t.equals("long") || t.equals("java.lang.Long"))

        {

            return Long.valueOf(value);

        }

        if(t.equals("boolean") || t.equals("java.lang.Boolean"))

        {

            return Boolean.valueOf(value);

        }

        if(t.equals("char") || t.equals("java.lang.Character"))

        {

            return Character.valueOf(value.charAt(0));

        }

        if(t.equals("java.lang.String"))

        {

            return value;

        }

        return value;

    }

    // 单元测试

//    public static void main(String[] args)

//    {

//        AfShell.start();

//    }

    ////////////// 解析器 ////////////

    class Pair

    {

        String k;  // key

        String v;  // value

        public Pair()

        {}

        public Pair(String k, String v)

        {

            this.k = k;

            this.v = v;

        }

    }

    class Command

    {

        String name="";

        ArrayList<Pair> arguments = new ArrayList<Pair>();

        public void parse(String cmdline) throws Exception

        {

            arguments.clear();

            String[] parts = cmdline.split(" ");

            this.name = parts[0].trim();

            for(int i = 1; i<parts.length; i++)

            {

                String p = parts[i];

                p=p.trim();

                if(p.length() == 0) continue;

                String[] kv = p.split("=");

                if(kv.length != 2)

                {

                    throw new Exception("参数错误, " + p);

                }

                String k = kv[0];

                String v = kv[1];

                k = k.trim();

                v = v.trim();

                this.arguments.add(new Pair(k,v));

            }

        }

    }

}

AfAction.java

package af;

public abstract class AfAction

{

        public abstract int execute() throws Exception;

}

1. \* 注解(annotation)

注解 Annotation

注解 Annotation：以@开头的句法

public class Child extends Base

{

     @Override

     public void test()

     {

          super.test();

     }

}

注解 Annotation：以@开头的句法

注释 Comment : // 或 /\* \*/

annoation和comment对代码的运行都没有影响。

区别：

comment: 给程序员看的，不会被编译到class文件中

annoation: 给编译器和IDE看的，有些会被编译到class文 件中，形成metadata。

@override的作用

告诉编译器，此函数是override父类的函数，因此 函数原型必须和父类的相同。 如果不同，则编译器/IDE会给出warning。。。

注意：override这一行可以去掉，也不关系。

相关文档与教程

官网提供的Java文档及教程

http://docs.oracle.com/javase/8/docs/index.ht

ml

->Tutorials and Training( The Java Tutorials )

->Trails Covering the Basics（Learning the Java

Language ）

->Annotations

1. 线程(1)

引例

有一个儒生Confucian，他的工作就是每天念经500遍。

public class Confucian

{

     public void doJob()

     {

          for(int i=1; i<=500; i++)

          {

          System.out.println("人之初，性本善 ..." + i);

          }

     }

}

还有一个佛教徒Buddhist,其工作也是念经。。

public class Buddhist

{

     public void doJob()

     {

          for(int i=1; i<=500; i++)

          {

               System.out.println("ma mi ma mi hong ...");

          }

     }

}

问题：你手下管理着两名员工，一个是Confucian，一 个是Buddhist，如何让他们同时工作？

Confucian e1 = new Confucian();

e1.doJob();

Buddhist e2 = new Buddhist();

e2.doJob();

我们发现，只有e1先工作完成了，才能开始e2的工作， 两者无法“同时”进行。。。

线程 Thread

引入线程机制，可以并行地做多件事情。。。

继承于Thread类，重写run()，填上它要做的工作。。

public class Buddhist extends Thread

{

     @Override

     public void run()

     {

          for(int i=1; i<=500; i++)

          {

               System.out.println("ma mi ma mi hong ...");

           }

     }

}

启动线程

Confucian e1 = new Confucian();

e1.start();

Buddhist e2 = new Buddhist();

e2.start();

启动一个线程 start()

线程主函数 run() : 描述了这个线程的工作任务

线程的调度

CPU只有一个，而系统中同时运行的进程有几十 个。。。 每个进程又同时运行n个线程。。。忙得过来？

时间片划分：将1秒钟划分为N个小的时间片，大家 轮流运行。。。

比如，以5ms为一个时间片，每个线程运行5ms之后 就轮到下一个线程运行。。。 这样，所有的线程都有机会被运行，由于切换的速 度很快，给用户的感觉是：它们好像同时在运行！

Sleep暂歇

线程可以执行Sleep操作，用于告诉操作系统: 我要 休息一会，接下来的一段时间内不要管我。。。

例如，

Thread.sleep(10000);

表示接下来将休息10秒钟，在这10秒钟内该线程主 动放弃CPU的使用权。。。

排队机制

假设线程①休息了10秒之后，那么醒来之后就能立 即获得CPU的使用权吗？不能。醒来之后要排队！ 系统中有一个队列，所有需要CPU的线程在这里排队。

操作系统根据特定的算法，来决定下一个是谁。 （比始说，有的线程优先级较高，而有的较低）

使用sleep

当创建线程时，sleep是一个经常要使用的函数

原则：尽可能少的占用CPU，让别的线程也有机制运行。

试想，如果一个线程里是这样的：

while(true)

{

System.out.println("我就赖着CPU。。。");

}

那其他的线程会高兴吗？

使用sleep

当创建线程时，sleep是一个经常要使用的函数

原则：尽可能少的占用CPU，让别的线程也有机制运行。

试想，如果一个线程里是这样的：

while(true)

{

System.out.println("我就赖着CPU。。。");

}

那其他的线程会高兴吗？

演示：

在Confucian和Buddhist中添加sleep…

虽然我们看到在任务管理器下很多线程，但在同一 时刻，大多数线程都在sleep。。。

小结

引入线程的概念 （多条线并行发展）

介绍了线程调度的机制，由操作系统负责调度

介绍了sleep的作用，及其重要意义

1. 线程(2)

线程的创建

第一种办法：继承 Thread

class Confusican extends Thread

{

     public void run()

     {

     }

}

new Confusican().start();

第二种办法：实现 Runnable 接口

class Confusican implements Runnable

{

     public void run()

     {

     }

}

Confusican obj = new Confusican();

Thread t = new Thread(obj);

t.start();

这是因为java不允许多重继承，如果你的类已经继承于别 的类，又要作为线程来运行，则可以使用这种方式。。

有时候使用匿名类更方便一些

Thread t = new Thread(){

public void run()

{

}

};

t.start();

小结

介绍了创建线程对象的两种方法

1. 线程(3)

线程的终止

当一个线程的start()调用后，线程为Alive状态。

当一个线程的主函数run()退出后，线程死亡(Dead)。

public void run()

{

     for(int i=1; i<=10; i++)

     {

          System.out.println("..." + i);

     }

}

该线程打印10句后终止。

（可以类比一下主线程，当main()中退出时，主程序终 止。。。）

让一个线程终止，就是要想办法让它从run()中退出。

例如，设置一个标识变量，

public boolean quitflag = false;

public void run()

{

     for(; ; )

     {

          if(quitflag) break; // 退出循环

          … 正常业务 …

     }

}

展示。。。

sleep与interrupt

背景：Buddhist中每5秒输出一次，sleep的时间很 长，如何让它立即退出？？

方法：

调用 t2.interrupt()来中断目标线程。则sleep()函数将抛出异常，结束sleep状态

等待线程的退出

Buddhist t3 = new Buddhist();

t3.start();

// 等待t3退出

t3.join();

join() 函数会阻塞住当前线程，等待目标线程退出后， 再继续执行。。。

注：在Java里, join()只起等待作用（线程死亡后不需 要手工回收，由JVM自动回收）

注意事项

本质上，线程只有一种退出方式：从run()函数自然 退出。 我们需要通过程序设计，控制它从run()中退出。

不能 stop() ,不能destroy()… deprecated 。很多初学者会有一个想法：有没有简单暴力的手段， 直接杀死一个线程？比如kill() ? 千万不要这么想！

（正在使用ATM存钱的时候，断电了可以吗？)

小结

线程只有一种终止方式：从run()中退出。

介绍如何控制一个线程的退出。

1. 线程的同步机制(synchronized) (1)

引例

密钥Key ：一个16字节的数组

密钥更新线程KeyUpdater：定期更新密钥

密钥获取线程KeyPrinter：打印显示密钥

密钥的完整性：密钥的16字节必须相同。

例如，

下面是有效密钥

1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A

下面是无效密钥

1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1B 1B 1B

密钥更新时应保证密钥的完整性。

什么样的情况算是出错？KeyPrinter获取密钥，发现密钥 不完整（16个字节不全相同），则认为此密钥无效!

分析代码，推断有没有可能出错。。。

运行代码，看看没有出错。。。

构造出错条件：假设 key.update()需要10ms 才能更新完成。。。

两个线程同时访问一个对象时，可能发生数据不同步的现象。

数据不同步：

比如，线程KeyUpdater正在更新密钥。。。

0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A xx xx xx xx

当更新过程完成了75%的时候。。。

另一个线程KeyPrinter要求获取当前密钥。。。它拿走了密 钥，发现它是一个不完整的密钥。。。

出错！

为了实现多线程对同一对象的同步访问，引入互斥锁的 概念。

synchronized (lock) // 申请锁

{  // 上锁 locked

     … 关键代码 …

}  // 解锁 unlocked

（1）若别的线程正持有该锁(locked)，则本线程阻塞等待

（2）若此锁空闲（unlocked)，则本线程持有锁(locked)， 进入大括号执行，完毕之后释放锁(unlocked)。

形象理解：

好比一间房子，如果是空的，就能进入使用里面的资源， 并在里面把门反锁，此时别人无法进入。（保证了资源的独 占使用）。此时有人想用房子，就得门口等着。。。

synchronized ( lock ) // 等待进入

{ // 进入，从里面反锁

… 使用资源res …

} // 出来

synchronized ( lock )

在Java里， synchronized (lock) 括号中的锁可以是任意

对象，甚至是数组对象。

比如，

Object obj = new Object();

synchronized (obj)

{

}

byte[] data = new byte[];

synchronized (data)

{

}

为了增强可读性，可以直接把要访问的那个对象当成锁， 类似下面这样：

synchronized(key)

{

     key.update(seed);

}

则可以一目了然地表明：大括号中要锁定的资源是key 对象。

小结

介绍了线程同步的概念：多个线程访问同一对象时， 一个读，一个写，则容易发生不同步的现象。

线程同步：就是让两个线程步调一致，有序地访问 的访问同一对象。

synchronized代码块用于同步访问。

1. 线程的同步机制(synchronized) (2)

synchronized 的更多写法

public void update(byte seed)

{

     synchronized (data)

     {

     }

}

public byte[] get()

{

     synchronized (data)

     {

     }

}

此时，外部就可以放心地调用update和get，不必担心不同步的问题。

由于要锁定的资源对象就是自己，所以也可以使用

synchronized (this)

{

 }

进一步简化的写法

public synchronized void update()

{

}

相当于

public void update()

{

     synchronized (this)

     {

     }

}

如果函数内的所有语句都被synchronized括起来，则可以这么简 写。

死锁 deadlock

死锁：因为不恰当地使用锁，导致线程锁死。

第一种情况:

synchronized (lock)

{

     synchronized (lock)

     {

     }

}

自己就在房子里，另一个自己在房子外面敲门？

第二种情况：两个线程互相等待。。。

线程①

synchronized (lock1)

{

     synchronized (lock2)

     {

     }

}

线程②

synchronized (lock2)

{

     synchronized (lock1)

     {

     }

}

注意事项

大括内的代码尽可能快地完成，以提高多线程的运行效 率。（让别人少等一会）

synchronized (lock)

{

}

建议：把不影响共享资源访问的耗时操作都移出去

小结

介绍了synchronized的更多写法

介绍了死锁的成因

1. 线程的通知机制(wait/notify)

引例

鸡蛋Egg

篮子ArrayList<Egg>

母鸡Hen : 每隔一段时间产出一个鸡蛋，放入篮子

男孩Boy : 负责将篮子里的鸡蛋取出来吃掉

要求：确保篮子里的鸡蛋被及时取走。 （如果取走不及时，鸡蛋就会堆积在篮子里。。。）

生产者 – 消费者 模型

轮询机制

由于Boy不确定母鸡什么时候会产出鸡蛋，只好频繁的 去检查篮子里有没有鸡蛋。有则取出，没有则空跑一趟。 假设母鸡每5-10秒会随机产出一个鸡蛋 ，则小明为了确保能及时取走鸡蛋，可以每隔1秒去检查 一次。。。这样，虽然会空跑很多次，但能够完成目标。

这种实现机制称为：轮询机制。（反复查询）

特点：效率低，但实现起来简单。

通知机制

为了提升小明的效率，不让小明空跑，则可以使用通知机制。 小明等待通知 basket.wait … ，母鸡在产出鸡蛋后，立即通知小明来取 basket.notify … ，这样，小明就能够在第一时间取走鸡蛋。

这种实现机制称为：通知机制

特点：效率高，但实现起来复杂。

wait / notify

wait / notify 是Object类的方法

发出通知

synchronized (basket)

{

     basket.add(egg);

     basket.notify();

}

等待通知

synchronized (basket)

{

     try{ basket.wait(); } catch (InterruptedException e){}

     if(basket.size() > 0) egg = basket.remove(0);

}

wait / notify

wait/notify 使用注意事项：

(1)必须结合synchronized使用

(2)synchronized对象和wait/notify的对象必须是同一 个

这是Java特定的机制，必须这么使用（官方说法是，在 wait/notify之前，必须先成为该对象的监视者Monitor) 。当然，也可以使用别的方法来实现通知机制。但这套机 制是Java自带的。

notify / notifyAll

notify : 通知一个线程

如果有很多线程都在wait，则由系统选择一个

notifyAll ： 通知所有线程

小结

介绍了轮询机制和通知机制的区别

介绍了如何使用wait/notify来实现通知机制