day03

Object类

Object是所有类的顶级超类,其中有两个经常被子类重写的方法:

toString()与equals().

```
package object;

/**
 * Object是所有类的顶级超类.里面有几个经常被子类重写的方法,其中包括toString和equals
 */
public class Demo {
    public static void main(String[] args)
{
        Point p = new Point(1,2);
        /*
            Object已经实现了toString方法.直接继承下来时返回的字符串内容为当前对象的地址信息.格式为:类名@地址.
            toString方法实际开发中很少直接写代码去调用它,都是在不经意间被自动执行的.
            例如在使用System.out.println输出时.与字符串连接操作时.
         */
```

```
        //System.out.println(Object obj)输出给定对象toString返回的字符串到控制台
        System.out.println(p);

        //字符串连接操作时,会将非String类型的对象调用toString转换为字符串后拼接.
        String line = "这是一个Point:" + p;
        System.out.println(line);

        Point p2 = new Point(1,2);
        System.out.println("p2:"+p2);
        /*
            对于引用类型而言,变量保存的值是对象的地址.
            ==比较是比较两个变量的值是否相等,因此对于引用类型而言就是比较地址是否相等,那么意思就是比较是否为同一个对象.

            equals是Object定义的另一个方法,意图是比较两个对象的内容是否相同.但是如果子类不重写这个方法,则没有实际意义,因为Object实现时内部仍然是用==比较的!
         */
        System.out.println(p == p2);//false

 System.out.println(p.equals(p2));//true
    }
}
```

包装类

java定义了8个包装类,目的是为了解决基本类型不能直接参与面向对象开发的问题,使得基本类型可以通过包装类的实例以对象的形式存在.

- 其中数字类型的包装类都继承自java.lang.Number,而char和boolean的包装类直接继承自Object
- Number是一个抽象类,定义了一些方法,目的是让包装类可以将其表示的基本类型转换为其他数字类型.

```
package integer;

public class IntegerDemo1 {
    public static void main(String[] args)
{
        //基本类型转换为包装类
        int i = 123;
        //java推荐我们使用包装类的静态方法
valueOf将基本类型转换为包装类,而不是直接new
        Integer i1 =
Integer.valueOf(i);//Integer会重用-128-127之
间的整数对象
        Integer i2 = Integer.valueOf(i);
        System.out.println(i1==i2);//true

 System.out.println(i1.equals(i2));//true

        double dou = 123.123;
```

```
        Double dou1 =
Double.valueOf(dou);//Double则是直接new
        Double dou2 = Double.valueOf(dou);

 System.out.println(dou1==dou2);//false

 System.out.println(dou1.equals(dou2));//tr
ue

        //包装类转换为基本类型
        int in = i1.intValue();//获取包装类对
象中表示的基本类型值
        double doub = i1.doubleValue();
        System.out.println(in);//123
        System.out.println(doub);//123.0

        in = dou1.intValue();//大类型转小类型
可能存在丢精度!
        doub = dou1.doubleValue();
        System.out.println(in);//123
        System.out.println(doub);//123.123
    }
}
```

包装类常用功能

```
package integer;

public class IntegerDemo2 {
```

```
    public static void main(String[] args)
{
        //1可以通过包装类获取其表示的基本类型的取
值范围
        //获取int的最大值和最小值?
        int imax = Integer.MAX_VALUE;
        System.out.println(imax);
        int imin = Integer.MIN_VALUE;
        System.out.println(imin);

        long lmax = Long.MAX_VALUE;
        System.out.println(lmax);
        long lmin = Long.MIN_VALUE;
        System.out.println(lmin);

        /*
            2字符串转换为基本类型的前提是该字符串
正确描述了基本类型可以保存的值,否则
            会抛出异常:NumberFormatException
         */
        String str = "123";
//          String str = "123.123";//这个字符串
不能解析为int值!
        int d = Integer.parseInt(str);
        System.out.println(d);//123
        double dou =
Double.parseDouble(str);
        System.out.println(dou);//123.123
    }
}
```

自动拆装箱特性

JDK5之后推出了一个新的特性:自动拆装箱

该特性是编译器认可的.当编译器编译源代码时发现有基本类型和引用类型相互赋值使用时会自动补充代码来完成他们的转换工作,这个过程称为自动拆装箱.

```
package integer;

public class IntegerDemo3 {
    public static void main(String[] args)
{
        /*
            触发自动拆箱特性,编译器会补充代码将包装类转换为基本类型,下面的代码会变为:
            int i = new
Integer(123).intValue();
         */
        int i = new Integer(123);
        /*
            触发编译器自动装箱特性,代码会被编译器改为:
            Integer in =
Integer.valueOf(123);
         */
        Integer in = 123;
    }
}
```

File类

File类的每一个实例可以表示硬盘(文件系统)中的一个文件或目录(实际上表示的是一个抽象路径)

使用File可以做到:

- 1:访问其表示的文件或目录的属性信息,例如:名字,大小,修改时间等等
- 2:创建和删除文件或目录
- 3:访问一个目录中的子项

但是File不能访问文件数据.

```
public class FileDemo {
    public static void main(String[] args)
{
        //使用File访问当前项目目录下的demo.txt文
件
        /*
            创建File时要指定路径，而路径通常使用
相对路径。
            相对路径的好处在于有良好的跨平台性。
            "./"是相对路径中使用最多的，表示"当前
目录"，而当前目录是哪里
            取决于程序运行环境而定，在idea中运行
java程序时，这里指定的
            当前目录就是当前程序所在的项目目录。
         */
//         File file = new
File("c:/xxx/xxx/xx/xxx.txt");
```

```
        File file = new File("./demo.txt");
        //获取名字
        String name = file.getName();
        System.out.println(name);
        //获取文件大小(单位是字节)
        long len = file.length();
        System.out.println(len+"字节");
        //是否可读可写
        boolean cr = file.canRead();
        boolean cw = file.canWrite();
        System.out.println("是否可读:"+cr);
        System.out.println("是否可写:"+cw);
        //是否隐藏
        boolean ih = file.isHidden();
        System.out.println("是否隐藏:"+ih);

    }

}
```

创建一个新文件

createNewFile()方法，可以创建一个新文件

```
package file;

import java.io.File;
import java.io.IOException;

/**
 * 使用File创建一个新文件
```

```
 */
public class CreateNewFileDemo {
    public static void main(String[] args)
throws IOException {
        //在当前目录下新建一个文件:test.txt
        File file = new File("./test.txt");
        //boolean exists()判断当前File表示的位
置是否已经实际存在该文件或目录
        if(file.exists()){
            System.out.println("该文件已存
在!");
        }else{
            file.createNewFile();//将File表
示的文件创建出来
            System.out.println("文件已创
建!");
        }

    }
}
```

删除一个文件

delete()方法可以将File表示的文件删除

```
package file;

import java.io.File;

/**
 * 使用File删除一个文件
```

```
 */
public class DeleteFileDemo {
    public static void main(String[] args)
{
        //将当前目录下的test.txt文件删除
        /*
            相对路径中"./"可以忽略不写，默认就是
从当前目录开始的。
         */
        File file = new File("test.txt");
        if(file.exists()){
            file.delete();
            System.out.println("文件已删
除!");
        }else{
            System.out.println("文件不存
在!");
        }
    }
}
```

创建目录

mkDir():创建当前File表示的目录

mkDirs():创建当前File表示的目录，同时将所有不存在的父目录一同创建

```
package file;

import java.io.File;
```

```
/**
 * 使用File创建目录
 */
public class MkDirDemo {
    public static void main(String[] args)
{
        //在当前目录下新建一个目录:demo
//          File dir = new File("demo");
        File dir = new
File("./a/b/c/d/e/f");

        if(dir.exists()){
            System.out.println("该目录已存
在!");
        }else{
//              dir.mkdir();//创建目录时要求所在
的目录必须存在
            dir.mkdirs();//创建目录时会将路径上
所有不存在的目录一同创建
            System.out.println("目录已创
建!");
        }
    }
}
```

删除目录

delete()方法可以删除一个目录，但是只能删除空目录。

```
package file;
```

```
import java.io.File;

/**
 * 删除一个目录
 */
public class DeleteDirDemo {
    public static void main(String[] args)
{
        //将当前目录下的demo目录删除
        File dir = new File("demo");
//          File dir = new File("a");
        if(dir.exists()){
            dir.delete();//delete方法删除目录
时只能删除空目录
            System.out.println("目录已删
除!");
        }else{
            System.out.println("目录不存
在!");
        }
    }
}
```