Agile Java

阿左 [1] Nobody [2]

January 20, 2013

[1]感谢读者
[2]感谢国家

Contents

List of Figures

List of Tables

Part I

基本概念

Chapter 1

开发环境

1.1 JUnit4

基本的 JUnit4 单元测试例子：

```
1  package net.jade;
2
3  import static org.junit.Assert.assertTrue;
4  import org.junit.BeforeClass;
5  import org.junit.AfterClass;
6  import org.junit.Before;
7  import org.junit.After;
8  import org.junit.Test;
9  import org.junit.Ignore;
10 import junit.framework.JUnit4TestAdapter;
11
12 public class HelloTest {
13
14   /**
15    * class setup must be static
16    */
17   @BeforeClass
18   public static void runBeforeClass() {
19     System.out.println("class setUp... ");
20   }
21
22   /**
23    * class tearDown must be static
```

```
24    */
25   @AfterClass
26   public static void runAfterClass() {
27     System.out.println("class tearDown... ");
28   }
29
30   @Before
31   public void setUp() {
32     System.out.println("func setUp... ");
33   }
34
35   @After
36   public void tearDown() {
37     System.out.println("func tearDown... ");
38   }
39
40   @Test
41   public void func01() {
42     System.out.println("func01... ");
43     assertTrue("hello".equals("hello"));
44   }
45
46   /**
47    * now can except for exception
48    */
49   @Test(expected=ArithmeticException.class)
50   public void func02() {
51     System.out.println("func02... ");
52     System.out.println("result is: " + (2/0));
53   }
54
55   /**
56    * this function will not run
57    * we want ignore this function
58    */
59   @Ignore
60   public void func03() {
61     System.out.println("func03... ");
62   }
63
64   /**
65    * test time out
```

```
66     */
67    @Test(timeout=500)
68    public void func04() {
69
70      System.out.println("func04... ");
71      try {
72        Thread.sleep(300);
73      } catch (InterruptedException ex) {
74        // do nothing
75      }
76    }
77
78    /**
79     * make junit4 programe alse can be used in
80     * junit3 environment
81     */
82    public static junit.framework.Test suite() {
83      return new JUnit4TestAdapter(HelloTest.class);
84    }
85  }
```

如果用 jdk 自带的方式编译与运行很麻烦：

```
1  #!/bin/bash
2  rm -rf build/classes
3  mkdir  build
4  mkdir  build/classes
5  javac -cp build/classes:lib/junit-4.8.2.jar \
6    -sourcepath src -d build/classes \
7    src/net/jade/*.java
8  java  -cp build/classes:lib/junit-4.8.2.jar \
9    org.junit.runner.JUnitCore net.jade.HelloTest
10 rm -rf build
```

有了 ant 的帮助就方便很多了：

```
1  <?xml version="1.0" ?>
2  <project name="simple" default="all" basedir=".">
3
4    <property name="projectName" value="Simple Project"/>
5
6    <property name="src.dir" value="src"/>
7    <property name="lib.dir" value="lib"/>
```

```
8
9   <property name="build.dir"        value="build"/>
10  <property name="build.classes"    value="${build.dir}/classes
        "/>
11  <property name="build.lib"        value="${build.dir}/lib"/>
12  <property name="build.pkg"        value="${build.dir}/pkg"/>
13  <property name="junit.output.dir" value="${build.dir}/
        junitreport"/>
14
15  <path id="compile.libs">
16    <fileset dir="${lib.dir}">
17      <include name="**/*.jar"/>
18    </fileset>
19    <pathelement location="${build.classes}"/>
20  </path>
21
22  <target name="clean" description="Remove all generated files.
        ">
23    <delete dir="${build.dir}" />
24  </target>
25
26  <target name="prepare" depends="clean"
27    description="Create build folders.">
28    <mkdir dir="${build.dir}"/>
29    <mkdir dir="${build.classes}"/>
30    <mkdir dir="${build.lib}"/>
31  </target>
32
33  <!-- compile. -->
34  <target name="compile" depends="prepare"
35    description="compile java scources.">
36    <javac srcdir="${src.dir}" destdir="${build.classes}"
37      includeantruntime="off">
38      <classpath refid="compile.libs"/>
39    </javac>
40  </target>
41
42  <!-- Run JUnit test classes. -->
43  <target name="junit" depends="compile">
44    <mkdir dir="${junit.output.dir}"/>
45    <junit fork="yes" printsummary="withOutAndErr"
46      haltonerror="yes" haltonfailure="yes" >
```

```
47        <formatter type="xml"/>
48        <classpath refid="compile.libs"/>
49        <test todir="${junit.output.dir}" name="net.jade.
              HelloTest"/>
50      </junit>
51    </target>
52
53    <!-- Generate JUnit report. -->
54    <target name="report"  depends="junit">
55      <junitreport todir="${junit.output.dir}">
56        <fileset dir="${junit.output.dir}">
57          <include name="TEST-*.xml"/>
58        </fileset>
59        <report format="frames" todir="${junit.output.dir}"/>
60      </junitreport>
61    </target>
62
63    <!-- Generate HTML format report. -->
64    <target name="jar" depends="report" description="compress jar
          .">
65      <jar basedir="${build.classes}" excludes="**/Test.class"
66        jarfile="${build.lib}/${projectName}.jar" />
67    </target>
68
69    <target name="all" depends="jar" description="all.">
70    </target>
71
72  </project>
```

Part II

常用工具

Chapter 2

基本工具

2.1 日期时间处理

2.1.1 格里高利历

通过 GregorianCalendar 进行日期操作：

```
1  package example;
2
3  import java.util.Date;
4  import java.util.Calendar;
5  import java.util.GregorianCalendar;
6
7  public class CalendarExample{
8
9    public static Date createDate(int year, int month, int day){
10     Calendar cal = new GregorianCalendar();
11     cal.clear();
12     cal.set(Calendar.YEAR, year);
13     cal.set(Calendar.MONTH, month-1);
14     cal.set(Calendar.DAY_OF_MONTH, day);
15     return cal.getTime();
16   }
17
18   public static Date addDay(Date date, int dayNum) {
19     Calendar cal = new GregorianCalendar();
20     cal.setTime(date);
21     cal.add(Calendar.DAY_OF_YEAR, dayNum);
```

```
22      return cal.getTime();
23    }
24
25  }
```

2.2 文本

2.2.1 换行符

在不同操作系统下取得换行符：

```
1  package stringtools;
2
3  public class StringConstans {
4
5    public static final String NEW_LINE = System.getProperty("
         line.separator");
6
7  }
```

2.3 枚举类型

```
1  package stringtools;
2
3  public enum Gender {
4    female, male
5  }
```

```
1  package stringtools;
2
3  public enum Color {
4
5    RED(255, 0, 0), BLUE(0, 0, 255), GREEN(0, 255, 0), //
6    YELLOW(255, 255, 0), BLACK(0, 0, 0), WHITE(0, 255, 0);
7
8    private int redValue;
9    private int greenValue;
10   private int blueValue;
```

```
11
12    private Color(int rv, int gv, int bv) {
13      this.redValue = rv;
14      this.greenValue = gv;
15      this.blueValue = bv;
16
17    }
18
19    public String toString() {
20      return super.toString() + "(" + redValue + "," + greenValue
              + ","
21            + blueValue + ")";
22    }
23
24  }
```

```
1   package test;
2
3   import static org.junit.Assert.assertEquals;
4
5   import org.junit.After;
6   import org.junit.Before;
7   import org.junit.Test;
8
9   import stringtools.Gender;
10  import stringtools.Color;
11
12  public class EnumTest {
13
14    @Test
15    public void testGender() {
16      Gender g = Gender.male;
17      assertEquals("male", g.toString());
18      assertEquals(g, Gender.valueOf("male"));
19    }
20
21    @Test
22    public void testColor() {
23      Color c = Color.RED;
24      assertEquals("RED(255,0,0)", c.toString());
25      assertEquals(Color.RED, c.valueOf("RED"));
26    }
```

```
27
28 }
```

2.4 数学

2.4.1 NaN 与无穷大

NaN 表示非数字，定义在java.lang.Float 与java.lang.Double 中。这两个类中同样还定义了正负无穷大的常量POSITIVE_INFINITY 和NEGATIVE_INFINITY 。整数除以 0 会导致错误，但 double 和 float 会在数学上生产合理的无穷大。

```
1  package test;
2
3  import static org.junit.Assert.assertEquals;
4  import static org.junit.Assert.assertTrue;
5  import static org.junit.Assert.assertFalse;
6
7  import org.junit.After;
8  import org.junit.Before;
9  import org.junit.Test;
10
11 import java.lang.Double;
12
13 public class MathTest {
14
15   @Test
16   public void testNaN() {
17     // boolean express alway return false
18     assertFalse(Double.NaN > 0.0);
19     assertFalse(Double.NaN < 0.0);
20     assertFalse(Double.NaN == 0.0);
21   }
22
23   @Test
24   public void testInfinity() {
25     double x = 1.0;
26     double tolerance = 0.5;
27
28     assertEquals( Double.POSITIVE_INFINITY,
29       Double.POSITIVE_INFINITY * 100, tolerance );
```

```
30      assertEquals( Double.NEGATIVE_INFINITY,
31        Double.POSITIVE_INFINITY * -1, tolerance );
32
33      assertEquals( Double.POSITIVE_INFINITY, x / 0.0, tolerance
            );
34      assertEquals( Double.NEGATIVE_INFINITY, x /-0.0, tolerance
            );
35
36      assertEquals(  0.0, x / Double.POSITIVE_INFINITY, tolerance
             );
37      assertEquals( -0.0, x / Double.NEGATIVE_INFINITY, tolerance
             );
38      assertEquals(   x , x % Double.POSITIVE_INFINITY, tolerance
             );
39
40      assertTrue( Double.isNaN( x  % 0.0) );
41      assertTrue( Double.isNaN(0.0 / 0.0) );
42      assertTrue( Double.isNaN(0.0 % 0.0) );
43
44      assertEquals( Double.POSITIVE_INFINITY,
45        Double.POSITIVE_INFINITY / x, tolerance );
46      assertEquals( Double.NEGATIVE_INFINITY,
47        Double.NEGATIVE_INFINITY / x, tolerance );
48
49      assertTrue( Double.isNaN(Double.POSITIVE_INFINITY % x) );
50
51      assertTrue(
52          Double.isNaN(Double.POSITIVE_INFINITY /Double.
                POSITIVE_INFINITY) );
53      assertTrue(
54          Double.isNaN(Double.POSITIVE_INFINITY %Double.
                POSITIVE_INFINITY) );
55      assertTrue(
56          Double.isNaN(Double.POSITIVE_INFINITY /Double.
                NEGATIVE_INFINITY) );
57      assertTrue(
58          Double.isNaN(Double.POSITIVE_INFINITY %Double.
                NEGATIVE_INFINITY) );
59      assertTrue(
60          Double.isNaN(Double.POSITIVE_INFINITY /Double.
                POSITIVE_INFINITY) );
61      assertTrue(
```

```
62          Double.isNaN(Double.POSITIVE_INFINITY %Double.
                POSITIVE_INFINITY) );
63      assertTrue(
64          Double.isNaN(Double.POSITIVE_INFINITY /Double.
                NEGATIVE_INFINITY) );
65      assertTrue(
66          Double.isNaN(Double.POSITIVE_INFINITY %Double.
                NEGATIVE_INFINITY) );
67
68    }
69
70  }
```

2.4.2 通过位逻辑处理权限

记录权限的枚举类：

```
1  package example;
2
3  public enum UserAuth {
4    ADD(1),EDIT(2),DELETE(4),SEARCH(8);
5
6    private int mask;
7
8    UserAuth(int mask) {
9      this.mask = mask;
10   }
11
12   public int getMask() {
13     return this.mask;
14   }
15 }
```

系统管理员类：

```
1  package example;
2
3  public class SysAdmin {
4    private int authValue = 0x0;
5
6    public void setAuth(UserAuth ... flags) {
```

```
    for(UserAuth f: flags)
      this.authValue |= f.getMask();
  }

  public void unsetAuth(UserAuth ... flags) {
    for(UserAuth f: flags)
      this.authValue &= ~f.getMask();
  }

  public boolean hasAuth(UserAuth ... flags) {
    boolean r = true;
    for(UserAuth f: flags)
      if(f.getMask() != (this.authValue & f.getMask()))
      { r = false; break; }
    return r;
  }

  public int getAuthValue() {
    return this.authValue;
  }
}
```

权限判断：

```
package test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import org.junit.Test;
import junit.framework.JUnit4TestAdapter;

import java.util.Date;

import example.UserAuth;
import example.SysAdmin;

public class UserAuthTest {

  private static SysAdmin a = new SysAdmin();
  private static UserAuth add = UserAuth.ADD;
  private static UserAuth edit = UserAuth.EDIT;
```

```
19    private static UserAuth del = UserAuth.DELETE;
20    private static UserAuth find = UserAuth.SEARCH;
21
22    @Test
23    public void testAuth() {
24      a.setAuth(add, del);
25      assertEquals(Integer.valueOf("101",2).intValue(), a.
            getAuthValue());
26      assertTrue(  a.hasAuth(add, del)  );
27      assertFalse( a.hasAuth(edit, find) );
28      assertTrue(  a.hasAuth(add)  );
29      assertTrue(  a.hasAuth(del)  );
30      assertFalse( a.hasAuth(edit) );
31      assertFalse( a.hasAuth(find) );
32
33      a.setAuth(edit);
34      a.setAuth(find);
35      a.unsetAuth(add, del);
36      assertFalse(  a.hasAuth(add, del)  );
37      assertTrue( a.hasAuth(edit, find) );
38      assertFalse(  a.hasAuth(add)  );
39      assertFalse(  a.hasAuth(del)  );
40      assertTrue( a.hasAuth(edit) );
41      assertTrue( a.hasAuth(find) );
42
43      a.unsetAuth(edit);
44      assertFalse(  a.hasAuth(add, del, edit)  );
45      assertTrue( a.hasAuth(find) );
46      assertFalse(  a.hasAuth(add)  );
47      assertFalse(  a.hasAuth(del)  );
48      assertFalse( a.hasAuth(edit) );
49      assertTrue( a.hasAuth(find) );
50    }
51
52  }
```

2.4.3 异或操作实现奇偶检验

基本的思想就是数一下位的值为 1 的个数是奇数还是偶数：

```
1  package example;
```

```
2
3  public class ParityChecker {
4    public byte checksum(byte [] bytes) {
5      byte checksum = bytes[0];
6      for(int i=1; i<bytes.length; i++)
7        checksum ^= bytes[i];
8      return checksum;
9    }
10 }
```

更加严格的检验除了给整个字节流加一位检验以外，还给每一个字节加上一个检验位。

2.4.4 BitSet

java.util.BitSet 类封装了一个以二进制位为元素的向量，并且长度可变方便进行位操作。这个类的优点不多，但是它的范围超过 int 类的取值范围。

2.4.5 数字的不同进制显示

```
1  package test;
2
3  import static org.junit.Assert.assertEquals;
4  import org.junit.Test;
5  import junit.framework.JUnit4TestAdapter;
6
7  import java.util.Date;
8
9  import example.UserAuth;
10 import example.SysAdmin;
11
12 public class NumberStringTest {
13
14   @Test
15   public void testNumberString() {
16     assertEquals("101", Integer.toBinaryString(5));
17     assertEquals("21",  Integer.toOctalString(17));
18     assertEquals("32",  Integer.toHexString(50));
19
20     assertEquals("101", Integer.toString( 5, 2));
```

```
21      assertEquals("21",  Integer.toString(17, 8));
22      assertEquals("32",  Integer.toString(50,16));
23
24      assertEquals((int)253,  (int)Integer.decode("0xFD"));
25      assertEquals((int)253,  (int)Integer.decode("0XFD"));
26      assertEquals((int)253,  (int)Integer.decode( "#FD"));
27      assertEquals((int)15,   (int)Integer.decode( "017"));
28      assertEquals((int)10,   (int)Integer.decode(  "10"));
29
30      assertEquals((int)-253,  (int)Integer.decode("-0xFD"));
31      assertEquals((int)-253,  (int)Integer.decode("-0XFD"));
32      assertEquals((int)-253,  (int)Integer.decode( "-#FD"));
33      assertEquals((int)-15,   (int)Integer.decode( "-017"));
34      assertEquals((int)-10,   (int)Integer.decode(  "-10"));
35    }
36
37  }
```

2.4.6 随机数

Math 类提供的 random 方法返回一个从 0.0 到 1.0 之间的 double 类伪随机数。

java.util.Random 功能更全面，产生 boolean、byte、int、long、float、double、甚至高斯型结果的伪随机数。如：Random 类的nextBoolean 方法根据提供的种子（没有种子就用系统时间当种子）返回布尔型的随机数，相同的种子产相同的数字序列。

还有一个java.util.SecureRandom 类用来生成标准的、强加密的伪随机数。

Chapter 3

输入输出

3.1 字符流

Writer 接口提供了对字符流的输出。下面的例子接受一个 writer 用来输出文本，而不关心具体输出到哪里：

```
1  package example;
2
3  import java.io.*;
4
5  public class ReportExample {
6
7    private Writer writer;
8
9    public void writeReport(Writer writer) throws IOException {
10     this.writer = writer;
11     this.writeHeader();
12     this.writeBody();
13     this.writeFooter();
14   }
15
16   public void writeHeader() throws IOException {
17     this.writer.write("This is the Header.\n");
18   }
19
20   public void writeBody() throws IOException {
21     this.writer.write("This is the Body.\n");
22   }
```

```
23
24    public void writeFooter() throws IOException {
25      this.writer.write("This is the Footer.\n");
26    }
27
28  }
```

```
1   package test;
2
3   import static org.junit.Assert.assertEquals;
4   import org.junit.Test;
5   import junit.framework.JUnit4TestAdapter;
6
7   import java.io.Writer;
8   import java.io.IOException;
9   import java.io.StringWriter;
10
11  import example.ReportExample;
12
13  public class ReportTest {
14
15    @Test
16    public void testReport() throws IOException {
17      String exp ="This is the Header.\n" +
18        "This is the Body.\n" + "This is the Footer.\n";
19
20      Writer w = new StringWriter();
21      ReportExample re = new ReportExample();
22      re.writeReport(w);
23
24      assertEquals(exp, w.toString());
25    }
26
27  }
```

3.1.1 文本文件

把输出写入一个文本文件中：

```
1   package example;
2
```

```
3  import java.io.*;
4
5  public class FileExample extends ReportExample {
6
7    public void writeReport(String fileName) throws IOException {
8      Writer w = new BufferedWriter(new FileWriter(fileName));
9      try{
10       super.writeReport(w);
11     } catch (IOException e) {
12       //
13     } finally {
14       w.close();
15     }
16   }
17
18 }
```

测试检查文件有内容是否正确，当然在操作之前与之后要删除文件：

```
1  package test;
2
3  import static org.junit.Assert.assertEquals;
4  import org.junit.Test;
5  import junit.framework.JUnit4TestAdapter;
6
7  import java.io.Writer;
8  import java.io.IOException;
9  import java.io.StringWriter;
10
11 import example.ReportExample;
12
13 public class ReportTest {
14
15   @Test
16   public void testReport() throws IOException {
17     String exp ="This is the Header.\n" +
18       "This is the Body.\n" + "This is the Footer.\n";
19
20     Writer w = new StringWriter();
21     ReportExample re = new ReportExample();
22     re.writeReport(w);
23
```

```
24      assertEquals(exp, w.toString());
25    }
26
27  }
```

其他 File 类的常用方法：

创建临时文件：createTempFIle

创建空白文件：createNewFile

文件操作：delete、deleteOnExit、renameTo

查询文件或路径名：getAbsoluteFile、getAbsolutePath、getCanonicalFile、getCanonicalPath、getName、getPath、toURI、toURL

级别：isFile、isDirectory

属性操作：isHidden、lastModified、length、canRead、canWrite、setLastModified、setReadOnly

目录操作：exists、list、listFiles、listRoots、mkdir、mkdirs、getParent、getParentFile

3.1.2 字节流的转换

InputStreamReader 和 OutputStreamWriter 包装了 InputStream 和 OutputStream。将其转换为字符流。

3.2 数据流

3.2.1 基本数据类型

DataInputStream 和 DataInputStream 提供了基本类型的读写操作，如：readInt、writeint 等：

```
1  package example;
2
3  import java.io.*;
4  import java.util.*;
5
6  public class BaseTypeExample {
7    private List<LogRecord> recs =
```

```
8       new ArrayList<LogRecord>();
9
10    public void add(LogRecord rec) {
11      this.recs.add(rec);
12    }
13
14    public List<LogRecord> getRecs() {
15      return this.recs;
16    }
17
18    public void clearRecs() {
19      this.recs.clear();
20    }
21
22    public void store(String fileName) throws IOException {
23      if(null != this.recs && this.recs.size() > 0) {
24        DataOutputStream o = null;
25        try{
26          o = new DataOutputStream(
27              new FileOutputStream(fileName));
28          o.writeInt(this.recs.size());
29          for(LogRecord r: this.recs) {
30            o.writeLong(r.getId());
31            o.writeUTF(r.getName());
32            o.writeLong(r.getCreateTime().getTime());
33          }
34        } finally {
35          o.close();
36        }
37      }
38    }
39
40    public void load(String fileName) throws IOException {
41      this.recs.clear();
42      DataInputStream ipt = null;
43      try {
44        ipt = new DataInputStream(
45            new FileInputStream(fileName));
46        int recCount = ipt.readInt();
47        for(int i=0; i< recCount; i++) {
48          LogRecord r = new LogRecord();
49          r.setId(ipt.readLong());
```

```
50          r.setName(ipt.readUTF());
51          r.setCreateTime(new Date(ipt.readLong()));
52          this.recs.add(r);
53        }
54      } finally {
55        ipt.close();
56      }
57    }
58
59  }
```

测试检查文件有内容是否正确，当然在操作之前与之后要删除文件：

```
1   package test;
2
3   import static org.junit.Assert.*;
4   import org.junit.Test;
5   import junit.framework.JUnit4TestAdapter;
6
7   import java.io.*;
8   import java.util.*;
9
10  import example.*;
11
12  public class BaseTypeTest {
13
14    private final String fileName = "testBaseType.txt";
15
16    private void deleteFile(String fileName) {
17      File f = new File(fileName);
18      if(f.exists()) {
19        assertTrue("Unable to delete " + fileName,
20            f.delete());
21      }
22    }
23
24    @Test
25    public void testBinaryFile() throws IOException {
26      BaseTypeExample be = new BaseTypeExample();
27      LogRecord r = new LogRecord();
28      r.setId(1L);
29      r.setName("rec1");
```

```
30      r.setCreateTime(new Date());
31      be.add(r);
32      //
33      r = new LogRecord();
34      r.setId(2L);
35      r.setName("rec2");
36      r.setCreateTime(new Date());
37      be.add(r);
38      //
39      r = new LogRecord();
40      r.setId(3L);
41      r.setName("rec3");
42      r.setCreateTime(new Date());
43      be.add(r);
44      try{
45        this.deleteFile(fileName);
46        //
47        be.store(fileName);
48        //
49        be.load(fileName);
50        assertEquals(3, be.getRecs().size());
51        for(int i=0; i<be.getRecs().size(); i++) {
52          LogRecord l = be.getRecs().get(i);
53          assertEquals(new Long(i+1), l.getId());
54          assertEquals("rec"+(i+1), l.getName());
55        }
56      } catch (Exception e) {
57        //
58      } finally {
59        this.deleteFile(fileName);
60      }
61    }
62
63  }
```

3.2.2 序列化对象

实现了java.io.Serializable 接口的类（包括子类）能够被序列化。对于类中不需要被序列化的成员可以通过关键字“transient”来修饰。加上serialVersionUID 来记录类的版本变化（不同的版本中可能会改变类的成

员):

```
package example;

import java.util.*;
import java.io.*;

public class LogRecSe implements Serializable {
  public static final long serialVersionUID = 1L;

  private Long id;
  private String name;
  private Date createTime;
  private transient Date updateTime;

  public LogRecSe() {
  }

  public Long getId() {
    return this.id;
  }

  public void setId(Long id) {
    this.id = id;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return this.name;
  }

  public Date getCreateTime() {
    return this.createTime;
  }

  public void setCreateTime(Date createTime) {
    this.createTime = createTime;
  }
}
```

直接使用 ObjectInputStream 与 ObjectOutputStream 来读写对象：

```
1  package example;
2
3  import java.io.*;
4  import java.util.*;
5
6  public class ObjectExample {
7    private List<LogRecSe> recs =
8      new ArrayList<LogRecSe>();
9
10   public void add(LogRecSe rec) {
11     this.recs.add(rec);
12   }
13
14   public List<LogRecSe> getRecs() {
15     return this.recs;
16   }
17
18   public void clearRecs() {
19     this.recs.clear();
20   }
21
22   public void store(String fileName) throws IOException {
23     if(null != this.recs && this.recs.size() > 0) {
24       ObjectOutputStream o = null;
25       try{
26         o = new ObjectOutputStream(
27             new FileOutputStream(fileName));
28         o.writeObject(recs);
29       } finally {
30         o.close();
31       }
32     }
33   }
34
35   public void load(String fileName) throws IOException,
36       ClassNotFoundException
37   {
38     this.recs.clear();
39     ObjectInputStream ipt = null;
40     try {
```

```
41        ipt = new ObjectInputStream(
42            new FileInputStream(fileName));
43        int recCount = ipt.readInt();
44        this.recs = (List<LogRecSe>) ipt.readObject();
45      } finally {
46        ipt.close();
47      }
48    }
49
50  }
```

```
1   package test;
2
3   import static org.junit.Assert.*;
4   import org.junit.Test;
5   import junit.framework.JUnit4TestAdapter;
6
7   import java.io.*;
8   import java.util.*;
9
10  import example.*;
11
12  public class ObjectSaveTest {
13
14    private final String fileName = "testObjectType.txt";
15
16    private void deleteFile(String fileName) {
17      File f = new File(fileName);
18      if(f.exists()) {
19        assertTrue("Unable to delete " + fileName,
20            f.delete());
21      }
22    }
23
24    @Test
25    public void testSaveObject() throws IOException {
26      ObjectExample be = new ObjectExample();
27      LogRecSe r = new LogRecSe();
28      r.setId(1L);
29      r.setName("rec1");
30      r.setCreateTime(new Date());
31      be.add(r);
```

```
32      //
33      r = new LogRecSe();
34      r.setId(2L);
35      r.setName("rec2");
36      r.setCreateTime(new Date());
37      be.add(r);
38      //
39      r = new LogRecSe();
40      r.setId(3L);
41      r.setName("rec3");
42      r.setCreateTime(new Date());
43      be.add(r);
44      try{
45        this.deleteFile(fileName);
46        be.store(fileName);
47        be.load(fileName);
48        assertEquals(3, be.getRecs().size());
49        for(int i=0; i<be.getRecs().size(); i++) {
50          LogRecSe l = be.getRecs().get(i);
51          assertEquals(new Long(i+1), l.getId());
52          assertEquals("rec"+(i+1), l.getName());
53        }
54      } catch (Exception e) {
55        //
56      } finally {
57        this.deleteFile(fileName);
58      }
59    }
60
61  }
```

Chapter 4

类、接口与反射

4.1 内部类

内部类可以访问定义在外部类中的实例变量；静态内部类不可以访问定义在外部类中的实例变量。

如果静态内部类不是 private，就可以被外部代码使用。

静态内部类可以被序列化，内部类不可以。

匿名内部类使用接口名加实现接口中的方法直接 new 出一个对象。但因为匿名内部类没有类名，所以也没有办法定义一个构造函数出来，不过可以用初始化代码块来完成初始化工作（用花括号括起来）。

内部类和匿名内部类不能访问局部变量，它所在的方法的参数就是典型的局部变量。为了可以访问，我们常常把方法的参数设为 final。因为内部类存在可能超过声明它的方法之外。而局部变量在离开代码块后就不存在了的。所以要声明为 final 才能被内部类访问。

4.2 适配器（Adapter）

由于在写匿名内部类时，要实现接口所有抽象方法的实现。这样会把内部类代码写得很长，为了在写内部类时简短一点，可以先写一个类实现对应的接口，但这个类实现的方法内容为空：方法中什么事也不做，或仅仅为了符合语法而 return null。这样一个有等于没有的类被叫作适配器。以后根据它来写内部类，就只要实现你要用到的方法就可以了，其他的方法虽然没有内容，但是在语法上已经合法了。

应用场景：第三方系统提供的接口 Dct，只提供了我们接口，但还没有拿到实现，我们要先根据接口进行测试，保证我们的调用没有问题：

```
1  package example;
2
3  public interface Dct {
4
5    public String sizeImage(String imgUrl, int width, int height)
         ;
6
7    public String encodeUrl(String url);
8
9  }
```

```
1  package example;
2
3  public class DctAdapter implements Dct {
4
5    public String sizeImage(String imgUrl, int width, int height)
         {
6      return null;
7    }
8
9    public String encodeUrl(String url) {
10     return null;
11   }
12
13 }
```

```
1  package test;
2
3  import static org.junit.Assert.*;
4  import org.junit.Test;
5  import junit.framework.JUnit4TestAdapter;
6
7  import example.*;
8
9  public class DctTest {
10
11   @Test
12   public void testAdapter() {
13     Dct mockDct = new DctAdapter() {
```

```
14        public String sizeImage(String url,
15            int width, int height)
16        {
17          return "/"+ width + "x" + height +
18            "/" + url;
19        }
20      };
21
22      assertEquals("/30x30/aa.png", mockDct.sizeImage("aa.png",
            30, 30));
23
24    }
25
26  }
```

4.3 反射

通过 Object 的 getClass 方法可以在程序运行过程中判断一个对象的类型。一个类的成员方法 isAssignableFrom 可以判断这个类是不是参数类的子类。Class 类的静态方法 forName 可以根据类名在运行期间生成一个类。

```
1  package example;
2
3  public class RefectClassExp {
4
5    public static Class createClass(String name) {
6      Class clazz = null;
7      try {
8        clazz = Class.forName(name);
9      } catch (ClassNotFoundException e) {
10       //
11     }
12     return clazz;
13   }
14
15 }
```

```
1  package test;
2
3  import static org.junit.Assert.*;
```

```
4  import org.junit.Test;
5  import junit.framework.JUnit4TestAdapter;
6
7  import java.util.*;
8  import example.*;
9
10 public class ClassRefTest {
11
12   @Test
13   public void testClassInstance(){
14     assertTrue("aaa" instanceof String);
15     assertTrue(String.class.isInstance("aaa"));
16   }
17
18
19   @Test
20   public void testClassRef() {
21     Class clazz = RefectClassExp.createClass("java.sql.Date");
22     assertEquals("java.sql.Date", clazz.getName());
23     assertFalse(clazz.isAssignableFrom(Date.class));
24     assertTrue(Date.class.isAssignableFrom(clazz));
25   }
26
27 }
```