黑马程序员
itheima.com

非贷款，0元入学，不1万就业不给1分钱学费，我们已干四年了！

笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

Chapter 10 反射+正则

反射

JAVA反射机制是在运行状态中，对于任意一个类（class文件），都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象方法的功能称为java语言的反射机制。

动态获取类中信息，就是java反射。可以理解为对类的解剖。

如果想要对指定名称的字节码文件进行加载并获取其中的内容并调用，这时就使用到了反射技术。

P.S.

所谓的框架就是对外提供一些接口，也就是功能扩展的标准，由实现类按照这个接口标准去实现。框架内部如果需要操纵这些实现类的对象完成某些操作，那么只需要把这些实现类的全名（包名+类名）写在某个配置文件中，框架代码只需要读取这个配置文件，就可以获取这个实现类的字节码文件，然后利用反射技术创建这个实现类的对象并且调用相应的方法完成一些操作。

用于描述字节码的类就是Class类，创建对象，可以提取字节码文件中的内容，如字段、构造函数、一般函数。该类就可以获取字节码文件中的所有内容，那么反射就是依靠该类完成的。想要对一个类文件进行解剖，只要获取到该类的字节码文件对象即可。

示例：获取字节码文件对象的3种方式

Person.java

```
01.  package cn.itcast.bean;
02.
03.  public class Person
04.  {
05.          private int age;
06.          private String name;
07.
08.      public Person(int age,String name){
09.                  super();
10.                  this.age = age;
11.                  this.name = name;
12.
13.                  System.out.println("Person param run..." + this.name + ":" +
     this.age);
14.          }
15.
16.          public Person(){
17.                  super();
18.
19.                  System.out.println("person run");
20.          }
21.
22.          public void show(){
23.                  System.out.println(name + "...show run..." + age);
24.          }
25.
26.          private void privateMethod(){
27.                  System.out.println("method run");
28.          }
29.
30.          public void paramMethod(String str,int num){
31.                  System.out.println("paramMethod run..." + str + ":" + num);
32.          }
33.
34.          public static void staticMethod(){
35.                  System.out.println("static method run...");
36.          }
37.  }
```

复制代码

ReflectDemo.java

```
import cn.itcast.bean.Person;

//要想要对字节码文件进行解剖，必须要有字节码文件对象。
public class ReflectDemo
{
        public static void main(String[] args) throws ClassNotFoundException {
                getClassObject_1();
                System.out.println("--------------------");
                getClassObject_2();
                System.out.println("--------------------");
                getClassObject_3();
        }

    /*
        * 获取字节码对象的方式：
        * 方式一：Object类中的getClass()方法的。
        * 想要用这种方式，必须要明确具体的类，并创建对象。
        * 麻烦。
        */
        public static void getClassObject_1(){

                Person p = new Person();
                Class clazz = p.getClass();

                Person p1 = new Person();
                Class clazz1 = p1.getClass();

                System.out.println(clazz == clazz1);
        }

        /*
        * 方式二：任何数据类型都具备一个静态的属性.class来获取其对应的Class对象。
        * 相对简单，但是还是要明确用到类中的静态成员。
        * 还是不够扩展。
        */
        public static void getClassObject_2(){

                Class clazz = Person.class;
                Class clazz1 = Person.class;

                System.out.println(clazz == clazz1);
        }

        /*
        * 方式三：只要通过给定的类的字符串名称就可以获取该类，更为扩展。
        * 可以用Class类中的方法完成。
        * 该方法就是forName。
        * 这种方法只要有名称即可，更为方便，扩展性更强。
        */
        public static void getClassObject_3() throws ClassNotFoundException {

                //可以把类的字符串名称写到配置文件中，然后读取出来。
```

```
53.                 String className = "cn.itcast.bean.Person";
54.                 Class clazz = Class.forName(className);
55.
56.                 System.out.println(clazz);
57.         }
58. }
```

复制代码

运行结果：

```
D:\code\day27>javac -d . Person.java

D:\code\day27>javac ReflectDemo.java

D:\code\day27>java ReflectDemo
person run
person run
true
--------------------
true
--------------------
class cn.itcast.bean.Person
```

示例：获取Class中的构造函数

```
01. import cn.itcast.bean.Person;
02. import java.lang.reflect.Constructor;
03. import java.lang.reflect.InvocationTargetException;
04.
05. public class ReflectDemo
06. {
07.         public static void main(String[] args) throws Exception {
08.                 createNewObject_1();
09.                 System.out.println("--------------------");
10.                 createNewObject_2();
11.         }
12.
13.         public static void createNewObject_1() throws
    ClassNotFoundException,InstantiationException,IllegalAccessException {
14.                 //早期：new时候，先根据被new的类的名称找寻该类的字节码文件，并加载进内
    存，
15.                 //并创建该字节码文件对象，并接着创建该字节文件的对应的Person对象。
16.                 //Person p = new Person();
17.
18.                 //现在：
19.                 String name = "cn.itcast.bean.Person";
20.                 //找寻该文件类文件，并加载进内存，并产生Class对象。
21.                 Class clazz = Class.forName(name);
22.                 //如何产生该类的对象呢？
23.                 Object obj = clazz.newInstance();//调用Person的空参构造函数
24.         }
25.
26.         public static void createNewObject_2() throws
    ClassNotFoundException,InstantiationException,NoSuchMethodException,IllegalAc
    cessException,InvocationTargetException {
27.
```

```
28.                 //Person p = new Person("小强",39);
29.
30.                 /*
31.                 * 当获取指定名称对应类中的所体现的对象时。
32.                 * 而该对象初始化不使用空参数构造函数该怎么办呢？
33.                 * 既然是通过指定的构造函数进行对象的初始化。
34.                 * 所以应该先获取到该构造函数，通过字节码文件对象即可完成。
35.                 * 该方法是：getConstructor(parameterTypes);
36.                 */
37.
38.                 String name = "cn.itcast.bean.Person";
39.                 //找寻该名称类文件，并加载进内存，并产生Class对象。
40.                 Class clazz = Class.forName(name);
41.                 //获取到了指定的构造函数对象
42.                 Constructor constructor =
     clazz.getConstructor(int.class,String.class);
43.                 //通过该构造器对象的newInstance方法进行对象的初始化。
44.                 Object obj = constructor.newInstance(38,"小明");
45.         }
46. }
```

复制代码

运行结果：

```
D:\code\day27>javac ReflectDemo.java
注意: ReflectDemo.java 使用了未经检查或不安全的操作。
注意: 要了解详细信息，请使用 -Xlint:unchecked 重新编译。

D:\code\day27>java ReflectDemo
person run
--------------------
Person param run...小明:38
```

示例：获取Class的字段

```
01. import cn.itcast.bean.Person;
02. import java.lang.reflect.Field;
03.
04. public class ReflectDemo
05. {
06.         public static void main(String[] args) throws Exception {
07.                 getFieldDemo();
08.         }
09.
10.         /*
11.         * 获取字节码文件中的字段。
12.         */
13.         public static void getFieldDemo() throws Exception {
14.
15.                 Class clazz = Class.forName("cn.itcast.bean.Person");
16.
17.                 //getField只能获取所有可访问公共字段，private获取不到。
18.                 //Field field = claszz.getField("age");
19.
```

```
20.                 //getDeclaredField可以获取到公共字段，也可以获取到私有字段。
21.                 Field field = clazz.getDeclaredField("age");
22.
23.                 //对私有字段的访问取消权限检查，暴力访问。
24.                 field.setAccessible(true);
25.
26.                 Object obj = clazz.newInstance();
27.
28.                 //为对象的属性赋值
29.                 field.set(obj,89);
30.
31.                 //获取某对象的某属性值
32.                 Object o = field.get(obj);
33.
34.                 System.out.println(field);
35.         }
36. }
```

复制代码

运行结果：

```
D:\code\day27>javac ReflectDemo.java

D:\code\day27>java ReflectDemo
person run
private int cn.itcast.bean.Person.age
```

示例：获取Class中的方法

```
01. import cn.itcast.bean.Person;
02. import java.lang.reflect.Method;
03. import java.lang.reflect.Constructor;
04.
05. public class ReflectDemo
06. {
07.         public static void main(String[] args) throws Exception {
08.                 getMethodDemo_1();
09.                 System.out.println("---------------------------");
10.                 getMethodDemo_2();
11.                 System.out.println("---------------------------");
12.                 getMethodDemo_3();
13.         }
14.
15.         /*
16.         * 获取指定Class中的公共函数。
17.         */
18.         public static void getMethodDemo_1() throws Exception {
19.
20.                 Class clazz = Class.forName("cn.itcast.bean.Person");
21.
22.                 Method[] methods = clazz.getMethods();//获取的都是公有的方法
23.
```

```
24.                 methods = clazz.getDeclaredMethods();//只获取本类中所有方法，包括私有。
25.
26.                 for(Method method : methods){
27.                         System.out.println(method);
28.                 }
29.         }
30.
31.         public static void getMethodDemo_2() throws Exception {
32.
33.                 Class clazz = Class.forName("cn.itcast.bean.Person");
34.
35.                 Method method = clazz.getMethod("show",null);//获取空参数一般方法
36.
37.                 Object obj = clazz.newInstance();
38.
39.                 Constructor constructor = clazz.getConstructor(int.class,String.class);
40.                 obj = constructor.newInstance(37,"小明");
41.
42.                 method.invoke(obj,null);
43.         }
44.
45.         public static void getMethodDemo_3() throws Exception {
46.
47.                 Class clazz = Class.forName("cn.itcast.bean.Person");
48.
49.                 Method method = clazz.getMethod("paramMethod",String.class,int.class);//获取空参数一般方法
50.
51.                 Object obj = clazz.newInstance();
52.
53.                 Constructor constructor = clazz.getConstructor();
54.                 obj = constructor.newInstance();
55.
56.                 method.invoke(obj,"小强",89);
57.         }
58. }
```

复制代码

运行结果：

```
D:\code\day27>java ReflectDemo
public void cn.itcast.bean.Person.paramMethod(java.lang.String,int)
public static void cn.itcast.bean.Person.staticMethod()
public void cn.itcast.bean.Person.show()
private void cn.itcast.bean.Person.method()
---------------------------
person run
Person param run...小明:37
小明...show run...37
---------------------------
person run
person run
paramMethod run...小强:89
```

练习：反射练习

代码：

PCI.java

```
package cn.itcast.reflect.test;

public interface PCI
{
        public void open();
        public void close();
}
```

复制代码

SoundCard.java

```
package cn.itcast.reflect.test;

public class SoundCard implements PCI
{
        public void open(){
                System.out.println("sound open");
        }

        public void close(){
                System.out.println("sound close");
        }
}
```

复制代码

NetCard.java

```
package cn.itcast.reflect.test;

public class NetCard implements PCI
{
        public void open(){
                System.out.println("net open");
        }

        public void close(){
                System.out.println("net close");
        }
}
```

复制代码

Mainboard.java

```
01.	package cn.itcast.reflect.test;
02.
03.	public class Mainboard
04.	{
05.		public void run(){
06.			System.out.println("main board run...");
07.		}
08.
09.		public void usePCI(PCI p){
10.			if(p != null){
11.				p.open();
12.				p.close();
13.			}
14.		}
15.	}
```

复制代码

pci.Properties

```
01.	pci1=cn.itcast.reflect.test.SoundCard
02.	pci2=cn.itcast.reflect.test.NetCard
```

复制代码

ReflectTest.java

```
01.	package cn.itcast.reflect.test;
02.
03.	import java.io.File;
04.	import java.io.FileInputStream;
05.
06.	import java.util.Properties;
07.
08.	/*
09.	* 电脑运行
10.	*/
11.	public class ReflectTest
12.	{
13.		public static void main(String[] args) throws Exception {
14.
15.			Mainboard mb = new Mainboard();
16.
17.			mb.run();
18.			//每次添加一个设备都需要修改代码传递一个新创建的对象
19.			//mb.usePCI(new SoundCard());
20.			//能不能不修改代码就可以完成这个动作
```

```
                //不用new完成，而是只获取其class文件，在内部实现创建对象的动作。

                File configFile = new File("pci.properties");

                Properties prop = new Properties();
                FileInputStream fis = new FileInputStream(configFile);

                prop.load(fis);

                for(int x = 0; x < prop.size(); x++){

                        String pciName = prop.getProperty("pci" + (x + 1));

                        Class clazz = Class.forName(pciName);//用Class去加载这个pci子类

                        PCI p = (PCI)clazz.newInstance();

                        mb.usePCI(p);
                }

                fis.close();
        }
}
```

复制代码

运行结果：

```
D:\code\day27>javac -d . Mainboard.java

D:\code\day27>javac -d . PCI.java

D:\code\day27>javac -d . Mainboard.java

D:\code\day27>javac -d . SoundCard.java

D:\code\day27>javac -d . NetCard.java

D:\code\day27>javac -d . ReflectTest.java

D:\code\day27>java cn.itcast.reflect.test.ReflectTest
main board run...
sound open
sound close
net open
net close
```

正则

示例：

```java
/*
* 正则表达式。
*
* 正则表达式用于操作字符串数据。
* 通过一些特定的符号来体现的。
* 所以我们为了掌握正则表达式，必须要学习一些符号。
*
* 虽然简化了，但是阅读性差。
*/
public class RegexDemo
{
        public static void main(String[] args){
                String qq = "123456701";
                //checkQQ(qq);

                //正则表达式。
                String regex = "[1-9][0-9]{4,14}";
                boolean b = qq.matches(regex);
                System.out.println(qq + ":" + b);
        }
        /*
         * 需求：定义一个功能对QQ号进行校验。
         * 需求：长度5~15，只能是数字，0不能开头。
         */
         public static void checkQQ(String qq){

                int len = qq.length();

                if(len >= 5 && len <= 15){
                        if(!qq.startsWith("0")){
                                try{
                                        long l = Long.parseLong(qq);

                                        System.out.println(l + ":正确");
                                }catch(NumberFormatException e){
                                        System.out.println(qq + ":含有非法字符");
                                }
                        }else{
                                System.out.println(qq + ":不能0开头");
                        }
                }else{
                        System.out.println(qq + ":长度错误");
                }
         }
}
```

复制代码

运行结果：

```
D:\code\day27>javac RegexDemo.java

D:\code\day27>java RegexDemo
123456701:true
```

正则表达式常用构造摘要

字符类

[abc] a、b 或 c（简单类）

[^abc] 任何字符，除了 a、b 或 c（否定）

[a-zA-Z] a 到 z 或 A 到 Z，两头的字母包括在内（范围）

预定义字符类

. 任何字符（与行结束符可能匹配也可能不匹配）

\d 数字：[0-9]

\D 非数字： [^0-9]

\s 空白字符：[\t\n\x0B\f\r]

\S 非空白字符：[^\s]

\w 单词字符：[a-zA-Z_0-9]

\W 非单词字符：[^\w]

Greedy 数量词

X?　　X，一次或一次也没有

X*　　X，零次或多次

X+　　X，一次或多次

X{n}　　X，恰好 n 次

X{n,}　X，至少 n 次

X{n,m} X，至少 n 次，但是不超过 m 次

边界匹配器

^ 行的开头

$ 行的结尾

\b 单词边界

\B 非单词边界

\A 输入的开头

\G 上一个匹配的结尾

\Z 输入的结尾，仅用于最后的结束符（如果有的话）

\z 输入的结尾

示例：

```
public class RegexDemo
{
        public static void main(String[] args){

                String str = "aoob";
                String reg = "ao?b";

                boolean b = str.matches(reg);
                System.out.println(str + ":" + b);

                reg = "ao+b";
                b = str.matches(reg);
                System.out.println(str + ":" + b);

                str = "ab";
                reg = "ao+b";
                b = str.matches(reg);
                System.out.println(str + ":" + b);

                str = "ab";
                reg = "ao*b";
                b = str.matches(reg);
                System.out.println(str + ":" + b);

                str = "aooooooob";
                reg = "ao{4,}b";
                b = str.matches(reg);
                System.out.println(str + ":" + b);

                reg = "ao{4,6}b";
                b = str.matches(reg);
                System.out.println(str + ":" + b);
        }
}
```

复制代码

运行结果：

```
D:\code\day27>javac RegexDemo.java

D:\code\day27>java RegexDemo
aoob:false
aoob:true
ab:false
ab:true
aooooooob:true
aooooooob:false
```

正则表达式常见功能：匹配、切割、替换、获取

示例：

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexDemo
{
        public static void main(String[] args){
                functionDemo_1();
                System.out.println("------------------------");
                functionDemo_2();
                System.out.println("------------------------");
                functionDemo_3();
                System.out.println("------------------------");
                functionDemo_4();
                System.out.println("------------------------");
                functionDemo_5();
                System.out.println("------------------------");
                functionDemo_6();
                System.out.println("------------------------");
                functionDemo_7();
        }

        /*
         * 正则表达式对字符串的常见操作
         * 1. 匹配
         *              其实使用的就是String类中的matches方法。
         *
         * 2. 切割
         *              其实使用的就是String类中的split方法。
         *
         * 3. 替换
         *
         * 4. 获取
         */

         /*
          * 演示匹配
          */
        public static void functionDemo_1(){
                //匹配手机号码是否正确
                String tel = "15800001111";

                String regex = "1[358]\\d{9}";

                boolean b = tel.matches(regex);

                System.out.println(tel + ":" + b);
        }

         /*
          * 演示切割
          */
        public static void functionDemo_2(){
                String str = "zhangsan      xiaoqiang            zhaoliu";

                String[] names = str.split(" +");

                for(String name : names){
                        System.out.println(name);
                }
        }

        public static void functionDemo_3(){
                String str = "zhangsan.xiaoqiang.zhaoliu";

                String[] names = str.split("\\.");

                for(String name : names){
                        System.out.println(name);
                }
        }

        /*
         * 在表达式 ((A)(B(C))) 中，存在四个这样的组：
         * 1 ((A)(B(C)))
         * 2 \A
         * 3 (B(C))
         * 4 (C)
         * 组零始终代表整个表达式。
         */
        public static void functionDemo_4(){
                String str = "zhangsanttttxiaoqiangmmmmmzhaoliu";

                //(.)表示一组，\\1+表示与第1组相同的出现1次以上
                String[] names = str.split("(.)\\1+");

                for(String name : names){
                        System.out.println(name);
                }
        }

         /*
          * 演示替换
          */
        public static void functionDemo_5(){
                String str = "zhangsanttttxiaoqiangmmmmmzhaoliu";

                //$表示前一个参数的第一组
                str = str.replaceAll("(.)\\1+","$1");

                System.out.println(str);
        }

        public static void functionDemo_6(){
                String str = "15800001111";

                str = str.replaceAll("(\\d{3})(\\d{4})(\\d{4})","$1****$2");

                System.out.println(str);
        }

        /*
          * 演示获取
          * 将正则规则进行对象的封装。
          * Pattern p = Pattern.compile("a*b");
          * 通过正则对象的matcher方法字符串相关联。获取要对字符串操作的匹配器对象Matcher。
          * boolean b = m.matches();
          */
        public static void functionDemo_7(){
                String str = "da jia hao,ming tian bu fang jia";

                // \\b表示单词边界
                String regex = "\\b[a-z]{3}\\b";

                //1. 将正则封装成对象
                Pattern p = Pattern.compile(regex);

                //2. 通过正则对象获取匹配器对象
                Matcher m = p.matcher(str);

                //使用Matcher对象的方法对字符串进行操作。
                //既然要获取三个字母组成的单词。
                //查找：find();
                while(m.find()){
                        System.out.println(m.group());//获取匹配的子序列
                        System.out.println(m.start() + ":" + m.end());
                }
        }
}
```

复制代码

运行结果：

```
D:\code\day27>javac RegexDemo.java

D:\code\day27>java RegexDemo
15800001111:true
------------------------
zhangsan
xiaoqiang
zhaoliu
------------------------
zhangsan
xiaoqiang
zhaoliu
------------------------
zhangsan
xiaoqiang
zhaoliu
------------------------
zhangsantxiaoqiangmzhaoliu
------------------------
158****0000
------------------------
jia
3:6
hao
7:10
jia
29:32
```

P.S.

1、Pattern类为正则表达式的编译表示形式。指定为字符串的正则表达式必须首先被编译为此类的实例。然后，可将得到的模式用于创建Matcher对象，依照正则表达式，该对象可以与任意字符序列匹配。执行匹配所涉及的所有状态都驻留在匹配器中，所以多个匹配器可以共享同一模式。

练习1：

```
public class RegexTest
{
        public static void main(String[] args){
                test();
        }

        /*
         * 1. 治疗口吃：我我...我我...我我我我...要要要要...要要要要...学学学学学...学学编编...编编编编..编..编...程程...程程程
         * 2. 对ip地址排序
         * 3. 对邮件地址校验
         */

         /*
          * 1. 治口吃
          */
         public static void test(){
                String str = "我我...我我...我我我我...要要要要...要要要要...学学学学学...学学编编...编编编编..编..编...程程...程程程";

                //1. 将字符串中.去掉，用替换。
                str = str.replaceAll("\\.+","");

                //2. 替换叠词
                str = str.replaceAll("(.)\\1+","$1");
                System.out.println(str);
         }
}
```

复制代码

运行结果：

```
D:\code\day27>javac RegexTest.java

D:\code\day27>java RegexTest
我要学编程
```

练习2：

```
import java.util.TreeSet;
import java.io.PrintStream;

public class RegexTest
{
        public static void main(String[] args){
                test();
        }

        /*
         * ip地址排序。
         * 192.168.10.34 127.0.0.1 3.3.3.3 105.70.11.55
         */

        public static void test(){
                String ip_str = "192.168.10.34 127.0.0.1 3.3.3.3 105.70.11.55";

                //1. 为了让ip可以按照字符串顺序比较，只要让ip的每一段的位数相同。
                //所以，补零，按照每一位所需最多0进行补充，每一段都加两个0。

                ip_str = ip_str.replaceAll("(\\d+)","00$1");
                System.out.println(ip_str);

                //然后每一段保留数字3位。
                ip_str = ip_str.replaceAll("0*(\\d{3})","$1");
                System.out.println(ip_str);

                //1. 将ip地址切出。
                String[] ips = ip_str.split(" +");

                TreeSet<String> ts = new TreeSet<String>();

                for(String ip : ips){
                        ts.add(ip);
                }

                for(String ip : ts){
                        System.out.println(ip.replaceAll("0*(\\d+)","$1"));
                }
        }
}
```

复制代码

运行结果：

```
D:\code\day27>javac RegexTest.java

D:\code\day27>java RegexTest
00192.00168.0010.0034 00127.000.000.001 003.003.003.003 00105.0070.0011.0055
192.168.010.034 127.000.000.001 003.003.003.003 105.070.011.055
3.3.3.3
105.70.11.55
127.0.0.1
192.168.10.34
```

练习3：

```
import java.util.TreeSet;
import java.io.PrintStream;

public class RegexTest
{
        public static void main(String[] args){
                test();
        }

        //对邮件地址校验。
        public static void test(){

                String mail = "abc1@sina.com";

                String regex = "[a-zA-Z0-9_]+@[a-zA-Z0-9]+(\\.[a-zA-Z]{1,3})+";

                boolean b = mail.matches(regex);

                System.out.println(mail + ":" + b);
        }
}
```

复制代码

运行结果：

```
D:\code\day27>javac RegexTest.java

D:\code\day27>java RegexTest
abc1@sina.com:true
```

练习4：爬虫

```
import java.util.ArrayList;
import java.util.List;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.io.IOException;

/*
* 网页爬虫：其实就是一个程序用于在互联网中获取符合指定规则的数据。
* 爬取邮箱地址。
*/
public class RegexTest
{
        public static void main(String[] args) throws IOException {
                List<String> list = getMails();

                for(String mail : list){
                        System.out.println(mail);
                }
        }

        public static List<String> getMails() throws IOException {
                //1. 读取源文件。
                BufferedReader bufr = new BufferedReader(new FileReader("c:\\mail.html"));

                //2. 对读取的数据进行规则的匹配。从中获取符合规则的数据。
                String mail_regex = "\\w+@\\w+(\\.\\w+)+";

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

                Pattern p = Pattern.compile(mail_regex);

                String line = null;

                while((line = bufr.readLine()) != null){
                        Matcher m = p.matcher(line);

                        while(m.find()){
                                list.add(m.group());
                        }
                }

                //3. 将符合规则的数据存储到集合中。
                return list;
        }
}
```

复制代码

运行结果：

```
D:\code\day27>javac RegexTest.java

D:\code\day27>java RegexTest
4546456@qq.com
sedi@sina.com
whoami@gmail.com
hehe@yahoo.com
```

从互联网上爬邮箱

```
import java.util.ArrayList;
import java.util.List;
import java.io.BufferedReader;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;

public class RegexTest
{
        public static void main(String[] args) throws IOException {
                List<String> list = getMailsByWeb();

                for(String mail : list){
                        System.out.println(mail);
```

```
17.             }
18.         }
19.
20.         public static List<String> getMailsByWeb() throws IOException {
21.
22.             URL url = new URL("http://www.itheima.com/aboutt/1376.html");
23.
24.             BufferedReader bufr = new BufferedReader(new
    InputStreamReader(url.openStream()));
25.
26.             String mail_regex = "\\w+@\\w+(\\.\\w+)+";
27.
28.             List<String> list = new ArrayList<String>();
29.
30.             Pattern p = Pattern.compile(mail_regex);
31.
32.             String line = null;
33.
34.             while((line = bufr.readLine()) != null){
35.                 Matcher m = p.matcher(line);
36.
37.                 while(m.find()){
38.                     list.add(m.group());
39.                 }
40.             }
41.
42.             return list;
43.         }
44.     }
```

复制代码

运行结果：

```
D:\code\day27>javac RegexTest.java

D:\code\day27>java RegexTest
eima@itcast.cn
heima@csdn.net
```

~END~

~爱上海，爱黑马~