目录

Day18. Java
1 反射 Reflect
1.1 获得“类对象”
1.2 获得包名、类名
1.3 成员变量的定义信息
1.4 构造方法的定义信息
1.5 方法的定义信息
1.6 反射创建对象
1.7 反射调用成员变量
1.8 反射调用成员方法
1.9 反射的作用：动态编程
2 注解
3 JUnit
4 网络爬虫

Day18. Java

1 反射 Reflect

- 从方法区，获得类对象
- 对这个类照透视，获得这个类的定义信息
  - 包名、类名
  - 成员变量
  - 构造方法
  - 方法
- 反射创建对象
- 反射调用成员

1.1 获得“类对象”

- A.class
- Class.forName("day1801.A")
- a1.getClass() 从Object 继承的方法

1.2 获得包名、类名

- c.getPackage().getName() 包名
- c.getName() 完整的类名
- c.getSimpleName() 获得类名，不含包名

反射
项目：day1801_反射

类：day1801.Test1

```
package day1801;

import java.util.Scanner;

public class Test1 {
   public static void main(String[] args) throws Exception {
      System.out.println("输入完整的类名:");
      String s = new Scanner(System.in).nextLine();
      Class<?> c = Class.forName(s);

      System.out.println(c.getPackage().getName());
      System.out.println(c.getName());
      System.out.println(c.getSimpleName());

      /*
       * java.lang.String
       * java.util.ArrayList
       * java.io.File
       */

   }
}
```

1.3 成员变量的定义信息

- getFields()
  获得可见的成员变量，包含从父类继承的变量

- getDeclaredFields()
  获得本类中定义的所有成员变量，
  包括私有变量

- getField(变量名)
- getDeclaredField(变量名)
  获得一个成员变量

1.4 构造方法的定义信息

- getConstructors()
  获得可见的构造方法

- getDeclaredConstructors()
  获得所有构造方法，包括私有

- getConstructor(参数类型列表)
- getDeclaredConstructor(参数类型列表)
  获得一个构造方法
  参数类型列表：String.class, int.class

1.5 方法的定义信息

- getMethods()
  获得可见的方法，包括从父类继承的方法

- getDeclaredMethods()

获得本类定义的方法
包括私有

- getMethod(方法名，参数类型列表)
- getDeclaredMethod(方法名，参数类型列表)
  获得一个方法

1.6 反射创建对象

- 执行无参构造

```
obj = c.newInstance()
```

- 执行有参构造

```
//从类，获得构造方法
Constructor t = c.getConstructor(int.class, String.class)

//用构造方法的操作，来新建实例，并执行这个构造方法
obj = t.newInstance(9527, "唐伯虎")
```

反射创建对象

Test2

```
package day1801;

import java.lang.reflect.Constructor;
import java.util.Scanner;

public class Test2 {
   public static void main(String[] args) throws Exception {
      System.out.println("输入类名: ");
      String s = new Scanner(System.in).nextLine();
      Class<?> c = Class.forName(s);

      Object o1 = null;
      Object o2 = null;
      try {
         o1 = c.newInstance();
         System.out.println(o1);
      } catch (Exception e) {
         System.out.println("不能执行无参构造");
      }
      System.out.println("\n\n----------------------");
      try {
         Constructor<?> t = c.getConstructor(int.class);
         o2 = t.newInstance(100);
         System.out.println(o2);
      } catch (Exception e) {
         System.out.println(
          "不能执行 int 参数的构造方法");
      }
   }
}
```

1.7 反射调用成员变量

- 获取一个变量

  ```
  Field f = c.getDeclaredField(变量名);
  ```

- 使私有变量，也可以被访问

  ```
  f.setAccessible(true)
  ```

- 为变量赋值

  ```
  f.set(对象，值)
  为指定对象的变量赋值
  如果是静态变量，第一个参数给 null 值
  ```

- 访问变量的值

  ```
  f.get(对象)
  获得指定对象中，该变量的值
  如果是静态变量，参数给 null 值
  ```

1.8 反射调用成员方法

- 获得指定的方法

  ```
  m = c.getDeclaredMethod(方法名,参数类型列表);
  ```

- 使私有方法，可以被调用

  ```
  m.setAccessible(true);
  ```

- 调用方法

  ```
  Object r = m.invoke(对象，参数数据)
  ```

  - 如果是静态方法，第一个参数给 null
  - 如果方法不返回值 void，得到null值

反射调用成员变量

Test3

```
package day1801;

import java.lang.reflect.Field;

public class Test3 {
   public static void main(String[] args) throws Exception {
      Class c = Student.class;
      Student s = new Student();

      //获得id变量
      Field id = c.getDeclaredField("id");
      //使私有变量可以被访问
      id.setAccessible(true);
      //给变量id赋值，指定对象 s
```

```
        id.set(s, 9527);
        System.out.println(s.getId());
        System.out.println("\n\n----------------");
        //访问指定的对象s中的id变量值
        int a = (Integer) id.get(s);
        System.out.println(a);




    }
}
```

Test4

```
package day1801;

import java.lang.reflect.Method;

public class Test4 {
    public static void main(String[] args) throws Exception {
        Class c = Student.class;
        Student s = new Student();

        //获得getName()和setName()方法
        Method getname =
            c.getMethod("getName");
        Method setname =
             c.getMethod("setName", String.class);

        setname.invoke(s, "唐伯虎");

        Object r = getname.invoke(s);
        System.out.println(r);

    }
}
```

Student

```
package day1801;

public class Student {
    private int id;
    private String name;
    private String gender;
    private int age;

    public Student() {
        super();
    }
    public Student(int id, String name, String gender, int age) {
        super();
        this.id = id;
        this.name = name;
        this.gender = gender;
```

```
        this.age = age;
    }
    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;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

}
```

1.9 反射的作用：动态编程

- 有的程序，有固定的执行流程
- 每一步执行的代码，可以任意配置
- 只需要修改配置文件，而不用修改代码，程序执行的运算，就可以完全改变

- 框架（SSM）底层代码，使用反射做动态编程

动态编程

```
配置文件:  src\config.txt
day1801.B;b
day1801.C;c
day1801.A;a

配置的类：公开，有无参构造
配置的方法：公开，无参，无返回值

Runner
自己编写一个运行工具，
根据配置文件的配置来执行

src/config.txt
A
B
C
Runner

package day1801;
```

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.net.URLDecoder;
import java.util.ArrayList;

/*
 * 执行器，根据config.txt的配置来执行
 */
public class Runner {
   static ArrayList<String> list =
         new ArrayList<>();
   static {
      try {
         //用相对路径得到文件完整路径
         String path =
            Runner.class
            .getResource("/config.txt")
            .getPath();
         path = URLDecoder.decode(path, "UTF-8");

         // BR--ISR--FIS--path
         BufferedReader in =
          new BufferedReader(
          new InputStreamReader(
          new FileInputStream(path),"GBK"));
         String line;
         while((line = in.readLine()) != null) {
            //"       sdfsdf ; sdfs "
            //"sdfsdf;sdfs"
            line = line.replaceAll("\\s+", "");
            if(line.length() == 0) {
                continue;
            }
            list.add(line);
         }
         in.close();
      } catch (Exception e) {
         System.out.println("无法加载配置文件");
         e.printStackTrace();
      }
   }

   public static void main(String[] args) {
      System.out.println(list);
   }
}
```

2 注解

- 为编译器、其他开发工具、Java程序，来提供代码的额外信息

  ```
  @Override
  public String toString() {
     ...
  }
  ```

  @Override注解，由编译器处理，会检查方法的定义，是否是正确的方法重写语法

- 自己定义的注解，需要自己来编写处理代码

注解

```
*)  自定义一个 @Test 注解
*)  在测试类的一组方法上，添加 @Test 注解
      class A {
         @Test
         public void f1() {}

         public void f2() {}

         @Test
         public void f3() {}
      }

*）定义一个运行器工具类 Runner，在 A 类中，自动查找带有 @Test 注解的方法，自动执行

项目：day1802_反射
类：day1802.Test
            A
            Runner
```

Test

```
package day1802;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

//对自定义注解进行注解（元注解）
//@Target({ElementType.METHOD,ElementType.FIELD,ElementType.CONSTRUCTOR,ElementType.PARAMETER})
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Test {
   int id() default 0;
   /*
    * *) 属性名可以任意命名
    * *) 名字叫 value 的属性名，有特殊待遇
    *    *) 单独赋值时，可以不写属性名
    *
    *    @Test(id=9527, value="....")
    *    @Test("....")
    */
   String value() default "";
}
```

A

```
package day1802;

public class A {
   @Test(id=9527)
   public void a() {
      System.out.println("a方法");
   }
   @Test(id=9528, value="测试方法b")
   public void b() {
```

```
        System.out.println("b方法");
    }
    public void c() {
        System.out.println("c方法");
    }
    @Test("测试方法d")
    public void d() {
        System.out.println("d方法");
    }
}
```

Runner

```
package day1802;

import java.lang.reflect.Method;

public class Runner {
    public static void launch(Class c) {
        try {
            //创建实例
            Object obj = c.newInstance();
            //获取所有方法
            Method[] a = c.getMethods();
            //遍历方法
            for (Method m : a) {
                //获得方法上的@Test注解信息
                //没有@Test注解，得到null值
                Test test =
                    m.getAnnotation(Test.class);
                //有@Test注解
                if(test != null) {
                    //获得注解属性数据
                    int id = test.id();
                    String msg = test.value();
                    System.out.println("\n\n-----------------");
                    System.out.println(id);
                    System.out.println(msg+"\n");
                    m.invoke(obj);//反射调用此方法
                }
            }
        } catch (Exception e) {
            System.out.println("执行失败");
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Runner.launch(A.class);
    }
}
```

3 JUnit

- Java 开源的单元测试框架
- eclipse 直接提供了插件支持

JUnit

项目：day1803_单元测试
类：day1803.Test1

添加 JUnit 开发包
项目-右键-Build Path-Add Libraries...

```
package day1803;

import org.junit.Test;

public class Test1 {
   @Test
   public void f1() {
      System.out.println(1);
   }

   @Test
   public void f2() {
      System.out.println(2);
   }

   @Test
   public void f3() {
      System.out.println(3/0);
   }
}
```

4 网络爬虫

- 自动获取web页面数据，抽取需要的数据

爬虫

项目：day1804_爬虫
类：day1804.Test1

添加 JUnit
项目-右键-Build Path-Add Libraries...

lib.zip 中的 lib 文件夹，拖拽到eclipse项目根目录

选中这10个jar文件

Test1

https://p.3.cn/prices/mgets?skuIds=J_7437780

https://list.jd.com/list.html?cat=9987,653,655

Test1

```
package day1804;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Map;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import org.junit.Test;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Test1 {
   @Test
   public void test1() throws Exception {
      String s =
         Jsoup
         .connect("http://www.baidu.com")
         .execute()
         .body();

      System.out.println(s);
   }

   @Test
   public void test2() throws Exception {
      String s =
       Jsoup
       .connect("https://item.jd.com/7437780.html")
       .execute()
       .body();

      System.out.println(s);
   }

   @Test
   public void test3() throws Exception {
      Document doc =
         Jsoup
         .connect("https://item.jd.com/7437780.html")
         .get();

      Elements es = doc.select("div.sku-name");

      Element e = es.get(0);
      String title = e.text();
      System.out.println(title);
   }

   @Test
   public void test4() throws Exception {
      String s =
         Jsoup
         .connect("https://p.3.cn/prices/mgets?skuIds=J_7437780")
         .ignoreContentType(true)
         .execute()
         .body();

      System.out.println(s);

      //解析 JSON 格式数据
```

```
        ObjectMapper mapper = new ObjectMapper();
        ArrayList<Map<String, String>> list =
         mapper.readValue(s, ArrayList.class);

        String price = list.get(0).get("p");
        System.out.println(price);
    }
}
```

Test2

```
package day1804;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import org.junit.Test;

public class Test2 {
    @Test
    public void test1() throws Exception {
        Document doc =
            Jsoup
            .connect("https://list.jd.com/list.html?cat=9987,653,655")
            .get();

        Elements es =
         doc.select("#plist li.gl-item");

        for (Element e : es) {
            String title =
             e.select("div.p-name em").get(0).text();

            String url =
             e.select("div.p-name a")
              .get(0)
              .attr("href");

            System.out.println(title);
            System.out.println(url);
            System.out.println("\n");
        }
    }
}
```