知识点列表

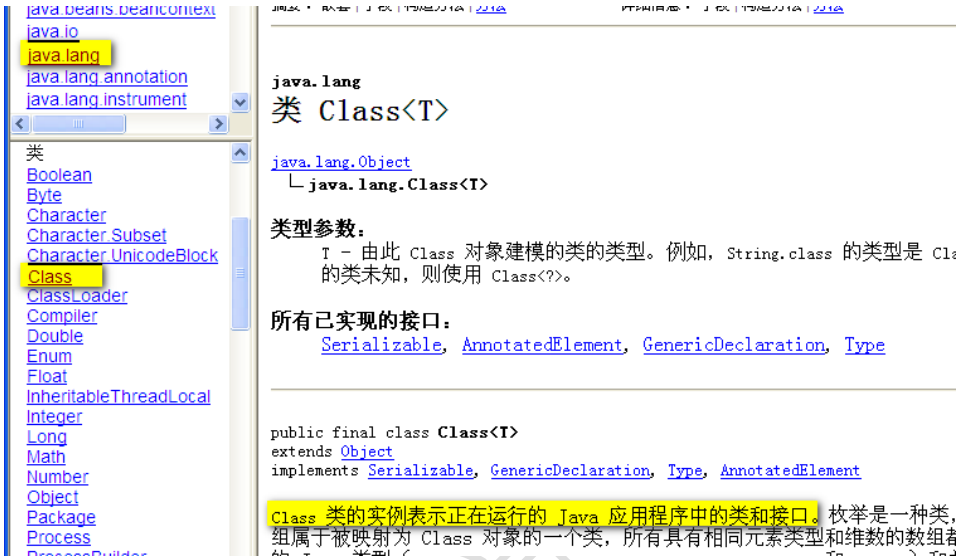
编号	名称	描述	级别
1	通过反射获得对象的类/属性/方法/构造器	了解如何通过反射获得对象的类/属性/方法/构造器 ,理解反射这种广泛存在于Java中的用法	*
2	通过反射创建对象实例	掌握利用反射创建对象实例	**
3	通过反射访问某对象的某属性	理解通过反射访问某对象的某属性	*
4	通过反射访问某对象的某方法	理解通过反射访问某对象的某方法	*

注： "*"理解级别 "**"掌握级别 "***"应用级别

目录

1. Java 反射

在学习反射之前，让我们先了解“类（Class）”。“方法”、“属性”、“类”都是名词，那么相应的在 Java 中会有这样一些特殊的类：“方法类（Method 类）”、“属性类（Field 类）”、“构造器类（Constructor 类）”、“类类（Class 类）”。



如上所示，任何 Java 的类或接口都是 Class 类的一个实例。

反射就是 Java 自我管理这些（类、对象）的机制。

1) 反射的作用（**重点理解**）
   - 可以通过反射机制**发现**对象的类型，发现类型的方法/属性/构造器
   - 可以**创建**对象并**访问**任意对象方法和属性等
2) Class 加载

   类加载到内存：Java 将磁盘类文件加载到内存中，为一个对象（实例），这个对象是 Class 的实例
3) Class 实例代表 Java 中类型
   - 获得基本类型实例
     - ✓ int.class
     - ✓ long.class
     - ✓ ...
   - 获得类类型（Class）实例:

     Class cls = String.class;

     Class cls = Class.forName("java.lang.String");

     Class cls = "abc".getClass();

     以上方法获得的 cls 是同一个对象，就是 String 类内存加载的结果

1.1. 通过反射获得对象的类/属性/方法/构造器 *

【案例】反射演示_“发现”对象的类/属性/方法/构造器

- **版本 01**

ReflectDemo.java

```
1  package corejava.day14.ch02;
2  import java.lang.reflect.Constructor;
6
7  /** 反射演示 */
8  public class ReflectDemo {
9    public static void main(String[] args) {
10       reflect("s"); //java.lang.String
11       reflect(1);   //java.lang.Integer
12   }
13
14   /**
15    * 反射方法
16    * 用于发现    obj 的类型是什么
17    *             obj 有哪些属性
18    *             obj 有哪些方法
19    *             obj 有哪些构造器
20    *
21    * @param obj 表示被"反射"的对象，被用于"发现"的对象
22    */
23   public static void reflect(Object obj){
24     //1. getClass()
25     //   返回对象的类型，是Object类的方法
26     Class cls = obj.getClass();
27     System.out.println("类:"+cls.getName());
28
29     //2. getDeclaredFields()
30     //   返回在类上获得声明的所有属性(字段)
31     Field[] fields = cls.getDeclaredFields();
32     System.out.println("属性:");
33     for(Field field : fields){
34       System.out.println(
35               field.getType()+" :"+ //属性类型
36               field.getName());     //属性名称
37     }
```

```
38
39      //3. getDeclaredMethods()
40      //    返回在类上获得声明的所有方法
41      Method[] methods = cls.getDeclaredMethods();
42      System.out.println("方法:");
43      for (Method method : methods) {
44        System.out.print(method.getReturnType()+" ");
45        System.out.print(method.getName() + " ");
46        System.out.println(
47          Arrays.toString(method.getParameterTypes()));
48      }
49
50      //4. getDeclaredConstructors()
51      //    返回返回在类上获得声明的所有构造器
52      Constructor[] constructors =
53                        cls.getDeclaredConstructors();
54      System.out.println("构造器:");
55      for (Constructor c : constructors) {
56        System.out.print(c.getName() + " ");
57        System.out.println(
58            Arrays.toString(c.getParameterTypes()));
59      }
60    }
61  }
```

- **版本 02**

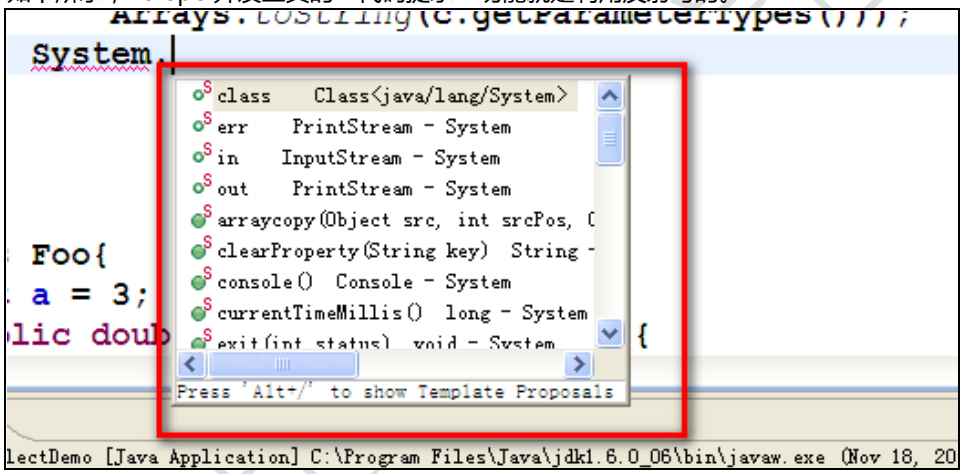
ReflectDemo.java

```
1  package corejava.day14.ch02;
2  import java.lang.reflect.Constructor;
6
7  /** 反射演示 */
8  public class ReflectDemo {
9    public static void main(String[] args) {
10       reflect(new Foo());   //corejava.day14.ch02.Foo
11   }
12
14     * 反射方法
22   public static void reflect(Object obj){
60 }
61
62 class Foo{
63   int a = 3;
64   public double add(int b, Double d){
65     return a+b+d;
66   }
67 }
```

```
Console
<terminated> ReflectDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 18, 2011 3:
类:corejava.day14.ch02.Foo
属性:
int :a
方法:
double add [int, class java.lang.Double]
构造器:
corejava.day14.ch02.Foo []
```

Java 是如何知道程序员开发程序的类、属性、方法的？通过反射技术（反射技术是 Java 底层 JVM 运行程序的机制）

如下所示，Eclispe 开发工具的”代码提示“功能就是利用反射写的。



1.2. 通过反射创建对象实例 **

【案例】反射演示_根据类名创建对象实例

ReflectDemo.java

```
1  package corejava.day14.ch03;
2  import java.lang.reflect.Constructor;
6
7  /** 反射演示 */
8  public class ReflectDemo {
9    public static void main(String[] args) {
10       //! 执行create()方法前提是参数类有无参构造器
11       Object obj = create("corejava.day14.ch03.Foo");
12       reflect(obj);
13
14       reflect(create("java.util.Date"));
15   }
16
17   /**  任务1 : 根据"类名"创建对象实例  */
18   public static Object create(String classname){
19     try {
20
21       //1.  加载类
22       //1.1 在CLASSPATH中查找对应的类
23       //1.2 采用"懒加载"方式装载到内存
24       Class cls = Class.forName(classname);
25       //2.  创建类实例
26       Object obj = cls.newInstance();
27
28       return obj;
29     } catch (Exception e) {
30       e.printStackTrace();
31       throw new RuntimeException("没搞定!",e);
32     }
33   }
34
35   /**
36    * 反射方法
44   public static void reflect(Object obj){
82 }
83
84
85 class Foo{
86   int a = 3;
87   public double add(int b, Double d){
88     return a+b+d;
89   }
90 }
```

注：

- ✓ Class.forName() 静态方法，可以利用类名在 **CLASSPATH** 中查找对应的类，并且装载到内存，返回这个”class “
- ✓ Class.forName()加载类的过程采用” **懒惰方式**”

“**懒惰方式**”，即检查发现如果已经加载了（内存中存在）就不再加载，直接返回已经加载的类，相当于“手工”去检查内存中是否已经加载了某个类

- ✓ .newInstance()方法，会利用默认（无参数）构造器创建类实例（实例对象）
- ✓ 第 31 行　比较简便的方法，告诉用户“对象没有创建成功”
  e.printStackTrace()；表示出错的原因

1.3. 通过反射访问某对象的某属性 *

【案例】反射演示_访问某对象的某属性

ReflectDemo2.java

```
1  package corejava.day14.ch04;
2  import java.lang.reflect.Constructor;
6
7  /** 反射演示 */
8  public class ReflectDemo2 {
9    public static void main(String[] args) {
10       //! 执行create()方法前提是参数类有无参构造器
11       Object obj = create("corejava.day14.ch04.Foo");
12       System.out.println(
13               getFieldValue(obj, "a")); //3
14   }
15
16   /**  任务1 : 根据"类名"创建对象实例  */
17   public static Object create(String classname){
33
34   /**  任务2 : 访问某对象的某属性*/
35   public static Object getFieldValue(
36           Object obj, String fieldname){
37     try{
38       //1. 反射出类型
39       Class cls = obj.getClass();
40       //2. 反射出类型字段
41       Field field = cls.getDeclaredField(fieldname);
42       //3. 在对象obj上读取field属性的值
43       Object val = field.get(obj);
44       return val;
45     }catch(Exception e){
46       e.printStackTrace();
47       throw new RuntimeException("没搞定!", e);
48     }
49   }
50
```

```
52⊕    * 反射方法
60⊕   public static void reflect(Object obj){
98  }
99
100 class Foo{
101   int a = 3;
102⊖   public double add(int b, Double d){
103     return a+b+d;
104   }
105 }
```

注：

✓ **第 43 行** field.get(obj)可以获得对象属性值
field.set(Object obj, Object value)可以设置对象属性值

1.4. 通过反射访问某对象的某方法 *

【案例】反射演示_访问某对象的某方法

ReflectDemo2.java

```
2⊕import java.lang.reflect.Constructor;
6
7  /** 反射演示 */
8  public class ReflectDemo2 {
9⊖   public static void main(String[] args) {
10      //! 执行create()方法前提是参数类有无参构造器
11      Object obj = create("corejava.day14.ch04.Foo");
12      Object mObj = call(
13            obj,                                          //obj
14            "add",                                        //方法名
15            new Class[]{int.class , Double.class},        //参数类型
16            new Object[]{2 , 3.5}                         //参数
17      );
18
19      System.out.println(mObj);    //8.5
20    }
21
22    /**  任务1 : 根据"类名"创建对象实例  */
23⊕   public static Object create(String classname){
39
40    /**  任务2 : 访问某对象的某属性*/
41⊕   public static Object getFieldValue(
```

```
56
57
58  /**  任务3 : 访问某对象的某方法
59   *
60   * @param obj              被调用对象
61   * @param method           方法名
62   * @param paramTypes       方法参数类型列表
63   * @param params           方法调用参数列表
64   *
65   * @return 在对象obj上调用
66   *         方法签名是(method,paramTypes)的方法,
67   *         params是传递的参数,
68   *         返回的是方法的结果,
69   *         如果无返回值, 返回null
70   */
71  @SuppressWarnings("unchecked")
72  public static Object call(
73          Object    obj,
74          String    method,
75          Class[]   paramTypes,
76          Object[]  params){
77
78    try{
79      //1. 发现类型
80      Class cls = obj.getClass();
81      //2. 发现方法
82      Method m = cls.getDeclaredMethod(
83                              method, paramTypes);
84      //3. 在对象obj调用方法m, 传递参数类别params
85      Object val = m.invoke(obj, params);
86      return val;
87    }catch(Exception e){
88      throw new RuntimeException("错误了!", e);
89    }
90  }
91
93    * 反射方法
101 public static void reflect(Object obj){
139 }
140
141 class Foo{
142   int a = 3;
143   public double add(int b, Double d){
144     return a+b+d;
145   }
146 }
```

注：

- ✓ **方法签名** 由方法名称与参数列表组成（没有返回值和访问控制符），被称为方法签名
- ✓ call 方法表示，调用任意对象的任意方法
  - ■ 调用无参方法时这样写：

```
12        Object mObj = call(
13              obj,                //obj
14              "add",              //方法名
15              new Class[]{},      //参数类型
16              new Object[]{}      //参数
17        );
```