

第十一章 反射及国际化问题

第一部分 Java中的 反射 reflect

- 反射: `java.lang.reflect`
- 如何动态的创建一个类?
- 如何动态的调用一个方法?
- 如何知道一个类所有的方法?

动态的构造一个类1 不传递任何构造参数

ch11.DynamicCreate01

```
public static void main(String[] args) {  
    try {  
        //要动态创建的类的全名  
        String className = "ch11.Student";  
        Class clazz = Class.forName(className);  
        //实例化, 本例子要求"ch11.Student"必须有无参数的构造方法。  
        Object obj = clazz.newInstance();  
        Student student = (Student) obj;  
        System.out.println("obj.class=" + obj.getClass().toString());  
        System.out.println("student.class=" + student.getClass().toString());  
    } catch (Exception e) {  
        System.out.println(e.getMessage());  
        e.printStackTrace();  
    }  
}
```

动态的构造一个类2 不传递任何构造参数

ch11.DynamicCreate02

```
public static void main(String[] args) {  
    try {  
        String className = "ch11.Student";  
        Class clazz = Class.forName(className);  
        //动态构造的另外一种方法，这种方法和DynamicCreate01类似  
        Object obj = clazz.getConstructor(new Class[0]).newInstance();  
        Student student = (Student) obj;  
        System.out.println("obj.class=" + obj.getClass().toString());  
        System.out.println("student.class=" + student.getClass().toString());  
    } catch (Exception e) {  
        System.out.println(e.getMessage());  
        e.printStackTrace();  
    }  
}
```

动态的构造一个类3 调用带有参数的构造函数

函数 ch11.DynamicCreate03

```
public static void main(String[] args) {  
    try {  
        String className = "ch11.Student";  
        Class clazz = Class.forName(className);  
        Class[] classes = new Class[2];  
        classes[0] = String.class;  
        classes[1] = Integer.class;  
        // clazz.getConstructor(classes) 寻找Student的带有参数的构造函数  
        // 且构造函数的第一个是参数String,第二个是Integer。  
        // .newInstance("tom", 123): 用"tom", 123实例化  
        Object obj = clazz.getConstructor(classes).newInstance("tom", 123);  
        Student student = (Student) obj;  
        System.out.println("obj.class=" + obj.getClass().toString());  
        System.out.println("student.class=" + student.getClass().toString());  
    } catch (Exception e) {  
        System.out.println(e.getMessage());  
        e.printStackTrace();  
    }  
}
```

java中的不定长参数

```
public class VariantParams {  
    public static void main(String[] args) {  
        List<String> list = asList("a", "b", "c");  
        for(String str:list) {  
            System.out.println(str);  
        }  
    }  
    public static <T> List<T> asList(T... a) {  
        //不定长参数 a是一个数组  
        java.util.ArrayList<T> list = new ArrayList<T>();  
        for (int i = 0; i < a.length; i++) {  
            list.add(a[i]);  
        }  
        return list;  
    }  
}
```

动态的构造一个类4 调用带有参数的构造函数

ch11.DynamicCreate04

```
public static void main(String[] args) {  
    try {  
        String className = "ch11.Student";  
        Object obj = createObject4(className, "aaa", 456);  
    } catch (Exception e) {  
        System.out.println(e.getMessage());  
        e.printStackTrace();  
    }  
}  
  
public static Object createObject4(String className, Object... values)  
    throws ClassNotFoundException, InstantiationException, Illegal  
        InvocationTargetException, NoSuchMethodException, SecurityExce  
Class clazz = Class.forName(className);  
Class[] classes = new Class[values.length];  
for (int i = 0; i < classes.length; i++) {  
    classes[i] = values[i].getClass();  
}  
Object obj = clazz.getConstructor(classes).newInstance(values);  
return obj;  
}
```

动态调用方法1 调用无参数方法

ch11. DynamicInvoke01

```
public static void main(String[] args) {  
    Student student = new Student("jack", 123);  
    try {  
        System.out.println(dynamicInvokeMethod(student, "getName"));  
        Object result = dynamicInvokeMethod(student, "getName");  
        System.out.println(result);  
    } catch (Exception e) {  
        System.out.println(e.getMessage());  
        e.printStackTrace();  
    }  
}
```

}

```
public static Object dynamicInvokeMethod(Object obj, String methodName,  
    InvocationTargetException, NoSuchMethodException, SecurityExcep  
    Method method = obj.getClass().getMethod(methodName);  
    return method.invoke(obj);  
}
```


动态调用方法2 调用有参数方法

ch11. DynamicInvoke02

```
public static void main(String[] args) {  
    Student student = new Student("jack", 123);  
    try {  
        dynamicInvokeMethod (student, "setName", "tom");  
        System.out.println(student.getName());  
    } catch (Exception e) {  
        System.out.println(e.getMessage());  
        e.printStackTrace();  
    }  
}  
  
public static Object dynamicInvokeMethod (Object obj, String methodName, Object... values)  
    throws IllegalAccessException, IllegalArgumentException, InvocationTargetException  
    SecurityException, ClassNotFoundException {  
    Class[] classes = new Class[values.length];  
    for (int i = 0; i < values.length; i++) {  
        classes[i] = values[i].getClass();  
    }  
    Method method = obj.getClass().getMethod(methodName, classes);  
    return method.invoke(obj, values);  
}
```

包含原始数据类型的方法动态调用

ch11.DynamicInvoke03/StudentPrime

StudentPrime包含如下方法

```
public void setAge(int age) {  
    this.age = age;  
}
```

int对应的类 int.class或者Integer.Type

动态调用属性

例子: ch11.DynamicField.java

```
public static void main(String[] args) {
    Object obj = new StudentPublic("jack", 123);
    try {
        dynamicField(obj, "name", "tom");
        System.out.println(((StudentPublic) obj).getName());
    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

public static void dynamicField(Object obj, String fieldName, Object value)
    throws IllegalArgumentException, InvocationTargetException, NoSuchMethodEx
{
    //取得所有public类型的属性
    Field field = obj.getClass().getField(fieldName);
    //直接设置属性值
    field.set(obj, value);
}
```

第二部分 Java中的乱码问题

μMLF 6"

◆ ◆ ◆ ◆

Home [D](#)

6.03 公斤/桶

什么是字符集

- 字符的集合

常见的几个字符集

- ASCII
- iso8859-1
- GB2312/GBK
- unicode

ASCII美国信息交换标准码

- 总共128个(0-127)

iso8859-1

- 西欧语言，兼容ASCII
- 包含阿尔巴尼亚语、巴斯克语、布列塔尼语、加泰罗尼亚语、丹麦语、荷兰语、法罗语、弗里西语、加利西亚语、德语、格陵兰语、冰岛语、爱尔兰盖尔语、意大利语、拉丁语、卢森堡语、挪威语、葡萄牙语、里托罗曼斯语、苏格兰盖尔语、西班牙语及瑞典语...

iso8859-1

| ISO/IEC 8859-1 | | | | | | | | | | | | | | | | |
|----------------|------|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|
| | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | | | | | | | | | | | | | | | | |
| 1x | | | | | | | | | | | | | | | | |
| 2x | SP | ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / |
| 3x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4x | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5x | P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| 6x | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7x | p | q | r | s | t | u | v | w | x | y | z | { | | } | ~ | |
| 8x | | | | | | | | | | | | | | | | |
| 9x | | | | | | | | | | | | | | | | |
| Ax | NBSP | ı | ¢ | £ | ¤ | ¥ | ¦ | § | ¨ | © | ª | « | ¬ | SHY | ® | ¯ |
| Bx | ° | ± | ² | ³ | ´ | µ | ¶ | · | ¸ | ¹ | º | » | ¼ | ½ | ¾ | ¿ |
| Cx | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï |
| Dx | Ð | Ñ | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û | Ü | Ý | Þ | ß |
| Ex | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï |
| Fx | ð | ñ | ò | ó | ô | õ | ö | ÷ | ø | ù | ú | û | ü | ý | þ | ÿ |

gb2312字符集

- 《信息交换用汉字编码字符集—基本集》
- 由中国国家标准总局发布,于1981年5月实施
- GBK字符集兼容gb2312
- [gb2312编码表](#)

unicode字符集

- 统一码、万国码、单一码
- Unicode用数字0-0x10FFFF来映射这些字符，最多可以容纳1114112个字符，或者说有1114112个码位
- 目前的Unicode分为17个平面（Plane），而每平面拥有65536个码点，最初的65536个字符在0号平面.

字符编码方案

- 字符编码方案是从一个或多个编码字符集到一个或多个固定宽度代码单元序列的映射

字符集编码方式 gb2312/gbk

- 将区位码直接保存在2个字节中
- 例如：

啊—B0A0

unicode编码

- UTF-32
- UTF-16
- UTF-8

UTF-8 1-4个字节变长编码

0x0000 - 0x007F

0xxxxxxx

0x0080 - 0x07FF

110xxxxx 10xxxxxx

0x0800 - 0xFFFF

1110xxxx 10xxxxxx 10xxxxxx

0x010000 - 0x10FFFF

1110xxxx 10xxxxxx 10xxxxxx 10xxxxxx

UTF-16 编码

- $U < 0x10000$

XXXXXXXX XXXXXXXX

- $U \geq 0x10000$

110110yyyyyyyyyy 110111xxxxxxxxxxx

注意：Unicode的 0xD800-0xDFFF为保留区

即：1101100000000000—

1101100011111111



“汽车” 在不同编码下的结果

| | |
|-----------------|---|
| gbk/gb2312 | <u>C6</u> <u>FB</u> <u>B3</u> <u>B5</u> |
| unicode | <i>FF FE</i> <u>7D</u> <u>6C</u> <u>66</u> <u>8F</u> |
| unicode big end | <i>FE FF</i> <u>6C</u> <u>7D</u> <u>8F</u> <u>66</u> |
| utf-8 | <i>EF BB BF</i> <u>E6</u> <u>B1</u> <u>BD</u> <u>E8</u> <u>BD</u> <u>A6</u> |
| | |

java乱码的根源

- 同样的byte[] 在不同的编码方式下表示不同的字符
- char->byte[]同样存在上述问题

二进制[C6 FB]究竟表示什么？

- iso8859-1 :  
- gbk: 汽
- utf-8: ?
- utf-16: ?

java乱码的根源

- java内部采用unicode

页面form —> java程序

java程序 —> 页面显示

数据库 —> java程序

java程序 —> 数据库

byte —> char

char —> byte

byte —> char

char —> byte

java乱码的根源

文件 —> java程序

byte —> char

java程序 —> 文件

char —> byte

流 —> java程序

byte —> char

java程序 —> 流
















char —> byte

解决乱码方法 charsetName一致

- String.getBytes(String charsetName)
- new String(byte[] data, String charsetName)
- InputStreamReader inReader=new
InputStreamReader(InputStream in, String
charsetName)
- 例子 ch11. CharSetDemo

java中的国际化问题

- 国际化 问题: i18n(internationalization)
- 资源文件类: ResourceBundle
- 地区类: Locale

| | | | |
|---|-----------------|---------------|------|
|  angular-locale_ee-gh.js | 2018/5/30 17:08 | JavaScript 文件 | 3 KB |
|  angular-locale_ee-tg.js | 2018/5/30 17:06 | JavaScript 文件 | 3 KB |
|  angular-locale_el.js | 2018/5/30 17:06 | JavaScript 文件 | 5 KB |
|  angular-locale_el-cy.js | 2018/5/30 17:08 | JavaScript 文件 | 5 KB |
|  angular-locale_el-gr.js | 2018/5/30 17:07 | JavaScript 文件 | 5 KB |
|  angular-locale_en.js | 2018/5/30 17:08 | JavaScript 文件 | 3 KB |
|  angular-locale_en-001.js | 2018/5/30 17:08 | JavaScript 文件 | 3 KB |
|  angular-locale_en-150.js | 2018/5/30 17:09 | JavaScript 文件 | 3 KB |
|  angular-locale_en-ag.js | 2018/5/30 17:10 | JavaScript 文件 | 3 KB |
|  angular-locale_en-ai.js | 2018/5/30 17:06 | JavaScript 文件 | 3 KB |
|  angular-locale_en-as.js | 2018/5/30 17:06 | JavaScript 文件 | 3 KB |
|  angular-locale_en-au.js | 2018/5/30 17:06 | JavaScript 文件 | 3 KB |
|  angular-locale_en-bb.js | 2018/5/30 17:06 | JavaScript 文件 | 3 KB |
|  angular-locale_en-be.js | 2018/5/30 17:07 | JavaScript 文件 | 3 KB |
|  angular-locale_en-bm.js | 2018/5/30 17:06 | JavaScript 文件 | 3 KB |