Groovy 分享

Sunweikun

Contents

如何设置Groovy的环境变量？

- 1.和Java一样先安装JDK.
- 2.设置环境变量java_home、path、classpath
- 3.下载GDK(http://dist.codehaus.org/groovy/distributions/groovy-1.0.zip)
- 4.设置环境变量GROOVY_HOME
- (我的变量值为D:\groovy-binary-1.6.2\groovy-1.6.2)
- 5.将GROOVY_HOME目录下的bin追加到环境变量path中
- 6. println 'Hello, world!' // 打印Hello, world!

Java与Groovy的判断比较

- **Java中的equals方法对应Groovy中的==，而Java中的==（判断是否引用同一对象）对应Groovy中的is方法**

```
String name1="Java"; String name2=new String("Java");
// Groovy中写为 name1 == name2
    if (name1.equals(name2)) {
      System.out.println("equal");
    } else {
      System.out.println("not equal");
    }
    // Groovy中写为 name1.is(name2)
    if (name1 == name2) {
      System.out.println("identical");
    } else {
      System.out.println("not identical");
    }
```

- **Java与Groovy的数组、循环比较**
- **Java中的数组定义int[] a = {1, 2, 3}; 在Groovy写成int[] a = [1, 2, 3]**
- **Java：**

```
for (int i =0; i < len; i++) {
  // do something
}
```

- **Groovy:**

```
for (int i =0; i < len; i++) {
  // do something
}

// 或者
for (i in 0..len-1) {
  // do something
}

// 或者
for (i in 0..<len) {
  // do something
}
```

- **Java中的inner class即内部类，在Groovy中用Closure实现（Closure是Java7正在考虑的一个特性，比inner class在语义方面更完善）**
- **Groovy中的注释比Java多了首行注释#!，其他与Java相同比如单行注释：//　多行注释：/* */　或者是　支持javadoc的/** */**
- **Java:**

```
/*
 * 多行注释
 */

/**
 * javadoc 注释
 */

// 单行注释
```

- **Groovy：**

```
#! 首行注释，使Unix shell能够定位Groovy启动程序以运行Groovy代码，例如
#!/usr/bin/groovy

/*
 * 多行注释
 */

/**
 * javadoc 注释
 */

// 单行注释
```

- **Java5中的for-each：for (Type t : iteratable) {...} 在Groovy中，for (t in iteratable) {...}**
- **Java:**
  - for (Type t : iterable) {
    // do something
    }
- **Groovy:**
  - for (t in iterable) {
    // do something
    }

- **Groovy**中**switch**语句与**Java**中相同，不过支持更多类型了，比如**String**
- **Groovy**的**while**语句跟**Java**相同，但废弃了**do-while**（考虑到语义方面的问题，而且**do-while**可以用其他形式的循环语句代替，使用频率低）
- **Java**中的**String**常量表示为**"Hello,** 满座**"**，在**Groovy**中可如下表示

```
// 双引号
"Hello, 满座"

// 单引号也可以
'Hello, 满座'

//多行字符串
"""Hello,
满座"""

//或者
'''Hello,
满座'''

// 替代字符串
def name = "满座"
"Hello, ${name}"
//或者
"Hello, $name"
```

- 对象创建在**Java**写为**Thought t = new Thought();** 在**Groovy**也可以这样写，不过还多了种写法：**def t = new Thought();**
- 静态方法调用在**Java**和**Groovy**中相同，即**ClassName.staticMethodName();**
- 实现接口和继承父类方面**Groovy**也与**Java**完全相同，即实现接口**class ClassName implements InterfaceName {...}** 继承父类：**class ClassName extends SuperClass {...}**
- 定义接口方面**Groovy**与**Java**完全相同，即**interface InterfaceNam{...} //** 在**Groovy**中默认为**public**的
- 正则表达式常量在**Java**中没有，在**Groovy**中表示为 **/pattern/**
- **Hash**常量(类型为**java.util.HashMap**)在**Java**没有，在**Groovy**中表示为 **def frequence = ["the": 5, "hello": 2, "world": 2]**
- 类变量即**static**变量，**Groovy**与**Java**相同，**static String name = "满座"**，在**Groovy**也可写为**static name = "满座"**

- 在**varargs**方法方面，**Groovy**比**Java**多一种表达方式，如下所示：
- **Java:**
  - public void varargsMethod(Type args) {
    //do something
    }
- **Groovy:**
  - // 与Java中的写法相同
    def varargsMethod(Type... args) {
    //do something
    }

    // Groovy还可以用[]代替...，反应出varargs的本质
    def varargsMethod(Type[] arg...) {
    //do something
    }

- 引用当前对象，**Groovy**和**Java**相同，在**Java**中用**this**表示，在**Groovy**中也用**this**表示，而且在**Groovy**中，**this**可以出现在**static**范围中，指向所在类的类对象，本例中，**this**等同于**ThisInStaticScope.class**（**Java**写法）或**ThisInStaticScope**（**Groovy**写法）

```
class ThisInStaticScope {
    static {
        println this
    }
    // 请不要诡异，参数类型可以省略。如果方法声明中有修饰关键字比如public，
synchronized，static等，则返回值类型可以省略。
    static main(args) {
        println this
    }
}
```

- 子类中调用父类方法，**Groovy**和**Java**也相同，在**Java**中 **super.methodName()** ，在**Groovy**中 **super.methodName()**
- 命名空间的定义，**Groovy**和**Java**相同，在**Java**中 **package edu.ecust.bluesun;** 在**Groovy**中 **package edu.ecust.bluesun**（分号可省略）
- 在导入类方面，**Groovy**和**Java**相同，在**Java**中 **import edu.ecust.bluesun.GroovyTest;** 在**Groovy**中 **import edu.ecust.bluesun.GroovyTest**
- **List**常量(类型为**java.util.ArrayList**)在**Java**中没有， 在**Groovy**中表示为 **def list = [3, 11, "Hello", "满座", "!"]**
- 在异常处理方面，**Groovy**与**Java**相同，除了不强制程序员捕获检查异常 **(checked exception)**外 **(**这跟**C#**很像，如果我没记错的话 **:)**
  并且在方法声明时，也可以不写**throws**语句。

- 方法的默认参数，**Java**中没有，**Groovy**中表示如下：

```
class Hello {
  //如果没有参数传入，默认打印出 Hello, 满座
  def greet(name="满座") {
    println("Hello, $name") //也可省略括号()
  }
}
```

- 在**Groovy**中，语句如果单独占一行的话，句尾的分号**(;)**可以省略，而在**Java**中每条语句后面必须跟有分号**(;)**
- 在**Groovy**中，如果不是**Boolean**或**boolean**类型，非**null**或非空(空字符串，**[]**，**[:]**)为**true**，**null**为**false**，而**Java**中对象不可以表示**true**或**false**；如果是**Boolean**或**boolean**类型，与**Java**中的一样。
- 在**Groovy**中，万事万物都是对象！而**Java**中不是这样，基本类型**(primitive type)**就不是对象。
- 在**Java**中，**Class**对象表示为**ClassName.class**，而在**Groovy**中，可以直接用**ClassName**表示**Class**对象
- **Groovy** 会自动导入**java.lang.*, java.util.*, java.net.*, java.io.*, java.math.BigInteger, java.math.BigDecimal, groovy.lang.*, groovy.util.***，而**Java**则只自动导入**java.lang.***

- **Groovy不仅有? :三元操作符，还有?:两元操作符，但Java只有? :三元操作符。**
  Groovy:
  - def a = null;
    // 如果a为“空”（null，空串“”，[]，[:]），那么结果为?:之后的那个值; 如果不为“空”，那么结果就是a
    def result = a ?: “default result”
    println result

    a = “满座"
    result = a ?: "default result"
    println result
- **Groovy能进行多重赋值，但Java不能**
- **Groovy:**
  - def a, b

    (a, b) = [1, 2] // 给a和b赋值

    println([a, b])

    (a, b) = [b, a] // 交换a和b的值

    println([a, b])

    def (c, d) = [1, 2] // 声明的同时进行初始化

    println([c, d])

- **Builder是Groovy相当有用的一个特性，样例常常用生成XML来展现Builder所带来的便利性，例如以下代码:**

```
def createXML(){
    def xml = new MarkupBuilder();
    xml.cars(number:2){
     description  '汽车大卖场'
     city '北京'
     img 'http://www.manzuo.com/beijing.jpg'
     car1(name:'卡尔·奔茨',ISBN:'1-2010-11-02')
     car2(name:'兰博基尼',ISBN:'1-2012-11-02')

    }
    xml.println();
 }
```

结果：

```
<cars number='2'>
  <description>汽车大卖场</description>
  <city>北京</city>
  <img>http://www.manzuo.com/beijing.jpg</img>
  <car1 name='卡尔·奔茨' ISBN='1-2010-11-02' />
  <car2 name='兰博基尼' ISBN='1-2012-11-02' />
</cars>
```

从例子可以看到,在Groovy中创建xml非常便利

- **Groovy Builder 创建HTML，代码示例：**

```
def createHTML(){
    def html =  new MarkupBuilder();
    html.html(){
      head{
        title 'Hello Word'
      }
      body{
        table{
          tr{td '姓名'  td 'Sunweikun'}
          tr{td '性别' td '男'}
          tr{td '爱好' td '打桌球、篮球、足球、KTV'}
        }
        DIV(id:'myDIV',class:2){h1 '太好玩啦'}
      }
    }
    html.println()
  }
```

结果：下一页

- 结果：

```
<html>
  <head>
    <title>Hello Word</title>
  </head>
  <body>
    <table>
      <tr>
        <td>姓名</td>
        <td>Sunweikun</td>
      </tr>
      <tr>
        <td>性别</td>
        <td>男</td>
      </tr>
      <tr>
        <td>爱好</td>
        <td>打桌球、篮球、足球、KTV</td>
      </tr>
    </table>
    <DIV id='myDIV' class='2'>
      <h1>太好玩啦</h1>
    </DIV>
  </body>
</html>
```

从以上代码看来，创建html的形式与创建xml非常相似。以节点方式创建，提高了同样功能java代码的效率

- **Groovy操作数据，效率特别高，代码简洁。同样我们与java的JDBC相比如下**

Java代码:

```
/**
 * 获得连接
 *
 * @return 连接对象
 * @throws ClassNotFoundException
 *             Class异常
 * @throws SQLException
 *             SQL异常
 */
public Connection getConn() throws ClassNotFoundException, SQLException{
    Env env = new Env();
    Class.forName(env.getProperty("DRIVER")); // 注册驱动
    // 获得连接并返回
    return DriverManager.getConnection(env.getProperty("URL"),env.getProperty("DBNAME"), env.getProperty("DBPASSWORD"));
}
```

Groovy代码:

```
/**
  * 数据库连接
  */
def static connectionOpen(){
    Sql.newInstance("jdbc:mysql://localhost/manzuo?useUnicode=true&amp;characterEncoding=UTF-8","root","root","org.gjt.mm.mysql.Driver");
}
```

以上比较。Java需要声明相关对象与返回值、异常处理,而Groovy不需要声明对象也不需要异常处理及返回类型。
相比之下Groovy是最好的选择

Groovy的CRUD等操作

```
/**
 * 查询
 */
def findAll(){
  this.connectionOpen().eachRow("select * from on_comptuan"){
    println "${it.id} -->"+"${it.activityName}-->"+"${it.region}"
  }
}
/**
 * 增加
 */
def add(){
  this.connectionOpen().execute("insert into on_comptuan(tname) values('hello Groovy')")
}
/**
 * 删除
 */
def delete(){
  this.connectionOpen().execute("delete from on_comptuan where id= ?",[66])
}
/**
 * 更新
 */
def update(){
  this.connectionOpen().execute("update on_comptuan set tname=? where id=?",["hahahhah",64])
}
```

- 看过以上之后我们会知道**Groovy**几乎完全兼容**Java**语法，还有人说**Groovy**是**Java**的孩子，**Groovy**不仅继承了**java**大部分的特性，而且还继承了很多种动态语言，比如**Python**，**Ruby**可能还会有其他语言，使**Groovy**变为高效率的编程语言。


LOGO
http://groovy.codehaus.org/
Thank You !