groovy高级语法

一、json操作

使用groovy自带的json工具进行json操作

groovy.json.JsonSlurper：将json原数据转成实体对象

groovy.json.JsonOutput：将实体对象转成json数据

def list = [

new Person(name: 'John', age: 25),

new Person(name: 'Major', age: 26)

]

// 对象转json

def json = JsonOutput.toJson(list)

println json // 一行输出json字符串

println JsonOutput.prettyPrint(json) // 以json格式输出json字符串

// json转对象

def jsonSlurper = new JsonSlurper()

jsonSlurper.parse(obj) // jsonSlurper.parseText(str)

groovy是完全兼容java的，所以java能使用的第三方json库（如：gson、fastjson），在groovy中也同样可以导入使用，但一般不建议这么做，groovy本身提供的工具箱就已经满足日常开发需求了。

下面是groovy将网络json数据转成实体对象，并访问对应属性字段的例子：

def response = getNetworkData("http://xxx/data.json")

println response.data.name // 不需要定义具体的实体类，就可以直接访问对应的属性字段

def getNetworkData(String url){

// 发送http请求

def connection = new URL(url).openConnection()

connection.setRequestMethod('GET')

connection.connect() // 该方法会阻塞线程

def response = connection.content.text

// 将json转化为实体对象

def jsonSluper = new JsonSlurper()

return jsonSluper.parseText(response)

}

二、xml文件操作

java xml处理：DOM文档驱动处理方式、SAX事件驱动处理方式。

groovy则为我们提供了xml工具箱：

groovy.xml.XmlSlurper：将xml原数据转成实体对象

groovy.xml.MarkupBuilder：将实体对象转成xml数据

1、解析xml

final String xml = '''

<response version-api="2.0">

<value>

<books id="1" classification="android">

<book available="20" id="1">

<title>疯狂Android讲义</title>

<author id="1">李刚</author>

</book>

<book available="14" id="2">

<title>第一行代码</title>

<author id="2">郭林</author>

</book>

<book available="13" id="3">

<title>Android开发艺术探索</title>

<author id="3">任玉刚</author>

</book>

<book available="5" id="4">

<title>Android源码设计模式</title>

<author id="4">何红辉</author>

</book>

</books>

<books id="2" classification="web">

<book available="10" id="1">

<title>Vue从入门到精通</title>

<author id="4">李刚</author>

</book>

</books>

</value>

</response>

'''

def xmlSlurper = new XmlSlurper()

def response = xmlSlurper.parseText(xml) // 读取并解析xml数据

println response.value.books[0].book[0].title.text() // 获取标签内容

println response.value.books[0].book[0].@available // 获取标签属性

// 获取所有作者是"李刚"的书籍名称

// 方法一：平行遍历xml数据

def list = []

response.value.books.each { books ->

books.book.each { book ->

def author = book.author.text()

if(author.equals('李刚')){

list.add(book.title.text())

}

}

}

println list.toListString()

// 方法二：深度遍历xml数据

// response.depthFirst().findAll 相当于 response.'\*\*'.findAll

def titles = response.depthFirst().findAll { book ->

return book.author.text() == '李刚' ? true : false

}

// 获取id为2的book节点的title内容

// 广度遍历xml数据

// response.value.books.children().findAll 相当于 response.value.books.'\*'.findAll

def names = response.value.books.children().findAll { node ->

node.name() == 'book' && node.@id == '2'

}.collect { node ->

return node.title.text()

}

李刚

20

[疯狂Android讲义, Vue从入门到精通]

其他：

深度遍历 depthFirst()可以使用'\*\*'表示。

广度遍历 children()可以使用'\*'表示。

2、生成xml

/\*\*

\* 生成xml格式数据

\* <langs type='current' count='3' mainstream='true'>

\* <language flavor='static' version='1.5'>java</language>

\* <language flavor='dynamic' version='1.6.0'>Groovy</language>

\* <language flavor='dynamic' version='1.9'>JavaScript</language>

\* </langs>

\*/

def sw = new StringWriter()

def xmlBuilder = new MarkupBuilder(sw) // 用来生成xml数据的核心类

// 使用伪方法创建 根结点langs，在参数括号中指定标签属性key-value

xmlBuilder.langs(type: 'current', count: '3', mainstream: 'true'){

// 同样，使用伪方法创建 language结点，不指定key时，该value将作为标签内容

language(flavor: 'static', version: '1.5', 'Java')

language(flavor: 'dynamic', version: '1.6.0', 'Groovy')

language(flavor: 'dynamic', version: '1.9', 'JavaScript')

}

println sw

上面是使用MarkupBuilder直接编写输出xml数据，但实际开发中，往往是将实体对象转成xml，在明白MarkupBuilder的用法之后，这样的需求处理也是差不多的：

def sw = new StringWriter()

def xmlBuilder = new MarkupBuilder(sw)

def langs = new Langs()

xmlBuilder.langs(type: langs.type, count: langs.count, mainstream: langs.mainstream) {

langs.languages.each { lang->

language (flavor: lang.flavor, version: lang.version, lang.value)

}

}

class Langs{

String type = 'current'

String count = '3'

String mainstream = 'true'

def Languages = [

new Language(flavor: 'static', version: '1.5', value: 'java'),

new Language(flavor: 'dynamic', version: '1.6.0', value: 'Groovy')

new Language(flavor: 'dynamic', version: '1.9', value: 'JavaScript')

]

}

class Language{

String flavor

String version

String value

}

...

三、文件操作

java文件处理：节点流（InputStream、OutputSteam及其子类）、处理流（Reader、Writer及其子类）

groovy文件处理：所有java对文件的处理类，groovy都可以使用；groovy扩展了许多更加快捷和强大的方法（ResourceGroovyMethods）。

def file = new File('../../GroovySpecification.iml')

file.eachLine { line -> // 遍历文件中的每一行

println line

}

def text = file.getText() // 返回文件中所有行内容的文本

def result = file.readLines() // 返回一个一行行文本内容的List

def readerBuffer = file.withReader { reader -> // 读取文件中前100个字符

char[] buffer = new char[100]

reader.read(buffer)

return buffer

}

// copy文件

def copy(String srcPath, String destPath) {

try{

// 创建目标文件

def destFile = new File(destPath)

if(!destFile.exists()){

destFile.createNewFile()

}

// 开始拷贝

new File(srcPath).withReader{ reader ->

def lines = reader.readLines()

destFile.withWriter{ writer ->

lines.each{ line ->

writer.append(line+"\r\n")

}

}

}

}catch(Exception e){

e.printStackTrace()

}

}

// 对象保存到文件

def saveObject(Object object, String path){

try{

// 创建目标文件

def destFile = new File(path)

if(!destFile.exists()){

destFile.createNewFile()

}

destFile.withObjectOutputStream { out ->

out.writeObject(object)

}

}catch(Exception e){

e.printStackTrace()

}

}

// 从文件读取对象

def readObject(String path){

def obj = null

try{

def file = new File(path)

if(file == null || !file.exists()) return null

// 从文件中读取对象

file.withObjectInputStream{ input ->

obj = input.readObject()

}

}catch(Exception e){

e.printStackTrace()

}

return obj

}

四、与java对比及总结

写法上，没有java那么多限制（如：分号、return）

功能上，对java已有的功能进行了极大的扩展

作用上，即可以编写应用，也可以编写脚本