**Groovy窥探-高级语法**

本章主要讲解基础中高级部分，主要内容为代码，如若不感兴趣者，可以大致浏览标题即可

* **groovy json操作详解**
  1. json转换为实体类对象(JsonSlurper)
  2. JsonSlurper slurper=**new** JsonSlurper()
  3. slurper.parse()

复制代码

* 1. 实体类对象转换为json(JsonOutput)
  2. //普通格式输出
  3. **def** list=[**new** Person(name: 'john',age: 20),**new** Person(name: 'Jane',age: 18)]
  4. println JsonOutput.toJson(list)
  5. //按照json格式输出的结果
  6. **def** json=JsonOutput.toJson(list)
  7. println JsonOutput.prettyPrint(json)

复制代码

* 大体介绍下第三方json的引用方式:
* 首先在和src同级的文件夹上创建libs文件夹，然后将jar文件复制到libs文件夹中，
* 然后在jar文件上右键选择add the library，到此就可以愉快的使用第三方lib了。
* 复制代码
  1. 实战演练：使用jsonSluper进行网络数据解析
  2. **def** getNetworkData(String url){
  3. //发送http请求 java
  4. **def** connection=**new** URL(url).openConnection()
  5. connection.setRequestMethod('GET')
  6. connection.connect()
  7. **def** response=connection.content.text
  8. //将json转化为实体对象 groovy
  9. **def** jsonSluper=**new** JsonSlurper()
  10. **return** jsonSluper.parseText(response)
  11. }
  12. **def** reponse=getNetworkData('请求地址')
  13. //输出你想要的结果
  14. println (reponse.data.head.name)

复制代码

* **xml文件操作详解**  
  在Java中我们对xml进行处理通常有两种方式：  
  ①DOM文档驱动处理  
  ②SAX事件处理（较为节省内存，操作复杂）
  1. 解析xml格式数据(XmlSlurper)
  2. **final** String xml = '''
  3. <response version-api="2.0">
  4. <value>
  5. <books id="1" >
  6. <book id="1" avaliable="10">
  7. <title>Android开发艺术探索</title>
  8. <author id="1">任玉刚</author>
  9. </book>
  10. <book id="2" avaliable="100">
  11. <title>第一行代码</title>
  12. <author id="2">郭霖</author>
  13. </book>
  14. <book id="3" avaliable="100">
  15. <title>疯狂Android讲义</title>
  16. <author id="3">李刚</author>
  17. </book>
  18. <book id="4" avaliable="100">
  19. <title>Vue从入门到精通</title>
  20. <author id="3">李刚</author>
  21. </book>
  22. </books>
  23. </value>
  24. </response>
  25. '''
  26. //开始解析xml数据
  27. **def** xmlSpluper=**new** XmlSlurper()
  28. **def** response=xmlSpluper.parseText(xml)
  29. //打印title
  30. println( response.value.books.book[0].title.text())
  31. //打印出属性节点的信息使用@
  32. println response.value.books.book[1].**@avaliable**
  33. //开始解析xml数据
  34. **def** xmlSpluper = **new** XmlSlurper()
  35. **def** response = xmlSpluper.parseText(xml)
  36. //普通遍历
  37. **def** list1 = []
  38. response.value.books.book.each{books->
  39. //对书进行遍历
  40. **def** author=books.author.text()
  41. **if** (author.equals('李刚')){
  42. list1.add(books.title.text())
  43. }
  44. }
  45. println list1.toListString()
  46. //深度遍历depthFirst()等价于"\*\*"
  47. **def** title = response.depthFirst().findAll { book -> **return** book.author.text() == '李刚' ? true : false }
  48. println title.toListString()
  49. //广度遍历children()等价于"\*"
  50. **def** resName=response.value.books.children().findAll{
  51. node->
  52. node.name()=='book' && node.**@id**=='2'
  53. }.collect{
  54. node->**return** node.title.text()
  55. }
  56. println resName

复制代码

* 1. 创建xml格式数据(MarkupBuilder)
  2. //静态生成
  3. **def** sw= **new** StringWriter()
  4. //MarkupBuilder生成xml格式的数据核心类
  5. **def** xmlBuilder=**new** MarkupBuilder(sw)
  6. //创建根节点langs
  7. xmlBuilder.langs(type:'current',count:'3',
  8. mainstream:'true'){
  9. language(flavor:'static',version:'1.5','java'){age('16')}
  10. language(flavor:'dynamic',version:'1.6','Groovy')
  11. language(flavor:'dynamic',version:'1.9','JavaScript')
  12. }
  13. println sw
  14. //实体类动态生成
  15. **class** **Langs**{
  16. String type='current'
  17. **int** count=3
  18. **boolean** mainstream=true
  19. **def** language=[
  20. **new** Language(flavor:'static',version:'1.5',value:'java'),
  21. **new** Language(flavor:'dynamic',version:'1.6',value:'Groovy'),
  22. **new** Language(flavor:'dynamic',version:'1.9',value:'JavaScript')
  23. ]
  24. }
  25. **class** **Language**{
  26. String flavor
  27. String version
  28. String value
  29. }
  30. **def** sw1= **new** StringWriter()
  31. ////MarkupBuilder生成xml格式的数据核心类
  32. **def** xmlBuilder=**new** MarkupBuilder(sw)
  33. **def** langs=**new** Langs()
  34. xmlBuilder.langs(type:langs.type,count:langs.count,mainstream:langs.mainstream){
  35. langs.language.each {
  36. //遍历所有的子节点
  37. lang->language(
  38. flavor:lang.flavor,version:lang.version,lang.value
  39. )
  40. }
  41. }
  42. println sw1

复制代码

**每日一练:**

* 通过上面的json和xml的操作，有兴趣的小伙伴可以尝试将json数据转换成xml的数据格式
* 复制代码
* \*\*\* groovy 文件操作\*\*  
  java中处理文件的方式最常见的就是流处理(inputstream、outputstream等）。  
  所有Java对文件的处理类，groovy都可以使用  
  **实战演练:文本文件的读写和copy**
* **def** file1 ='/Users/用户名/IDEProjects/MyGroovy/MyGroovy.iml'
* **def** file2 = '/Users/用户名/IDEProjects/MyGroovy/MyGroovy2.iml'
* **def** result=copy(file1,file2)
* println result
* **def** copy(String sourcePath, String destationPath) {
* **try** {
* //创建目标文件
* **def** desFile = **new** File(destationPath)
* **if** (!desFile.exists()) {
* desFile.createNewFile()
* }
* //开始拷贝
* **new** File(sourcePath).withReader { reader ->
* **def** lines = reader.readLines()
* desFile.withWriter {
* write ->
* lines.each {
* line ->
* write.append(line+"\r\n")
* }
* }
* }
* **return** true
* }
* **catch** (Exception e) {
* e.printStackTrace()
* }
* **return** false
* }
* 复制代码
* **实战演练：对象的读写和copy**
* **def** person=**new** Person(name:'Haha',age:25)
* saveObject(person,'../../person.bin')
* readObject(''../../person.bin'')
* //对象保存
* **def** saveObject(Object object, String path) {
* **try** {
* //创建目标文件
* **def** desFile = **new** File(path)
* **if** (!desFile.exists()) {
* desFile.createNewFile()
* }
* //开始拷贝
* desFile.withObjectOutputStream {
* out -> out.writeObject(object)
* }
* **return** true
* }
* **catch** (Exception e) {
* e.printStackTrace()
* }
* **return** false
* }
* //对象读取
* **def** readObject(String path){
* **def** obj=null
* **try** {
* **def** file=**new** File(path)
* **if** (file==null||!file.exists())**return** null
* file.withObjectInputStream {
* input->obj=input.readObject()
* }
* }**catch**(Exception e){
* }
* **return** obj
* }
* 复制代码

在Groovy中我们不需要自己手动关闭流的操作。

好了，到此为此，我们就已经将groovy的基础部分讲述完毕，基础很重要，希望大家能多练习下前面的基础内容部分。