# Gradle实践

## 文档结构

Gradle课程资料

有用软件

配套好书

第2节 gradle快速入门

第3节 开发环境搭建

第4节 Gradle核心语法讲解及实战

第5节 Gradle高级用法实战

第6节 Gradle生命周期探索

第7节 Gradle核心之Project详解及实战

第8节 Gradle核心之Task详解及实战

第9节 Gradle其它模块讲解与自定义Plugin实战

第10节 Gradle持续集成与打包

第11节 课程总结

第2节 gradle快速入门

2-1 本节概述

2-2 什么是领域特定语言DSL？

2-3 groovy初探

第4节 Gradle核心语法讲解及实战

4.1 本节概述

4.10 列表学习(上)

4.11 列表学习(下)

4.12 映射详解(上)

4.13 映射详解(下)

4.16 面向对象学习(中)

4.17 面向对象学习(下)

4.18 本节小结

4.2 基础语法讲解

4.3 String讲解

4.4 字符串方法详解

4.5 逻辑控制

4.6 闭包讲解之基础讲解

4.7 闭包使用讲解

4.8 字符串与闭包结合使用

4.9 闭包进阶讲解

第5节 Gradle高级用法实战

5-1 本节概述

5-2 json操作详解

5-3 xml文件解析讲解

5-4 xml文件生成

5-5 groovy文件处理详解（上）

5-6 groovy文件处理详解（下）

5-7 本节小结

第6节 Gradle生命周期探索

6-1 gradle概念及优势

6-2 gradle执行流程讲解

6-3 gradle生命周期监听

6-4 本节小结

第7节 Gradle核心之Project详解及实战

7-1 project详解

7-10 本节小结

7-2 project核心api分解

7-3 project相关api详解上

7-4 project api讲解下

7-5 属性相关api讲解上

7-6 属性相关api详解下

7-7 文件属性操作讲解

7-8 依赖相关api讲解及实战

7-9 gradle执行外部命令实战

第8节 Gradle核心之Task详解及实战

8-1 Task内容分类

8-2 Task定义和配置

8-3 Task执行讲解与实战

8-4 Task执行顺序详解及实战

8-5 Task依赖详解及实战

8-6 Task输入输出

8-7 挂接自定义Task到构建过程中

8-8 Task类型

8-9 Task学习总结

第9节 Gradle其它模块讲解与自定义Plugin实战

9-1 本节概述

9-2 Settings类讲解

9-3 SourceSet讲解

9-4 自定义Plugin实战上

9-5 自定义Plugin实战下

9-6 Android及Java插件对gradle的扩展

9-7 如何迁移到gradle

9-8 gradle全面总结

第10节 Gradle持续集成与打包

10-1 持续集成介绍及Jenkins作用

10-2 Jenkins环境变量搭建

10-3 Jenkins配置

第11节 课程总结

11-1 groovy学习总结.baiduyun.downloading

11-2 gradle学习总结.baiduyun.downloading

11-3 gradle实战总结.baiduyun.downloading

## 第2节Gradle快速入门

### 2.1 什么是领域特定语言DSL

Uml、matlab、xml、html、

### 2.2 groovy初探

是一种基于JVM的敏捷开发语言

结合了Python、Ruby、Smalltalk的许多强大的特性

Groovy可以与java完美结合，而且可以使用java所有的类库

思想：求转不求全，解决特定问题

语法上支持动态类型、闭包等新一代语言

面向对象、面向过程

## 第3节 开发环境搭建

### 3.1 linux下环境搭建

1）安装JDK；

2）下载Groovy SDK

$unzip apache-groovy-binary-2.4.15.zip

$yum install unzip

$sudo yum install unzip

$unzip apache-groovy-binary-2.4.15.zip

$ln -s groovy-2.4.15/ groovy

$rm -rf apache-groovy-binary-2.4.15.zip

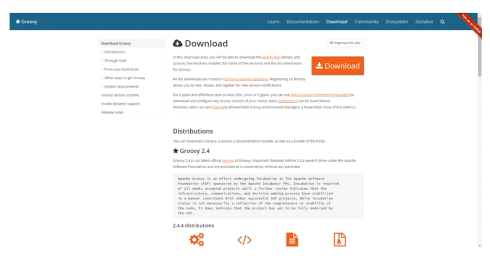
$sudo vi /etc/profile

$source /etc/profile

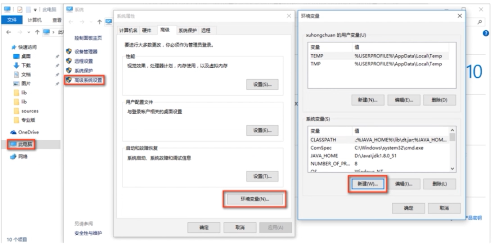
$groovy -v

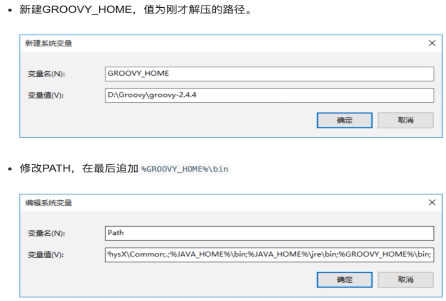
### 3.2 windows下环境搭建

1）下载Groovy SDK

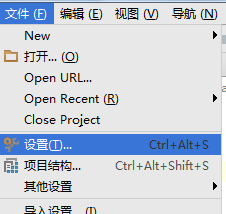


1. 配置classpath环境变量

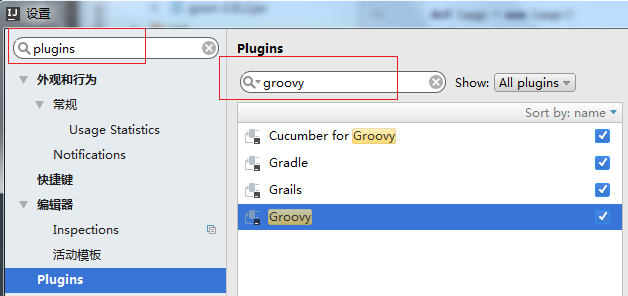




1. 在控制台输入groovy -version，校验是否正确安装
2. IntelliJ中配置groovy
3. 打开idea--preferences--

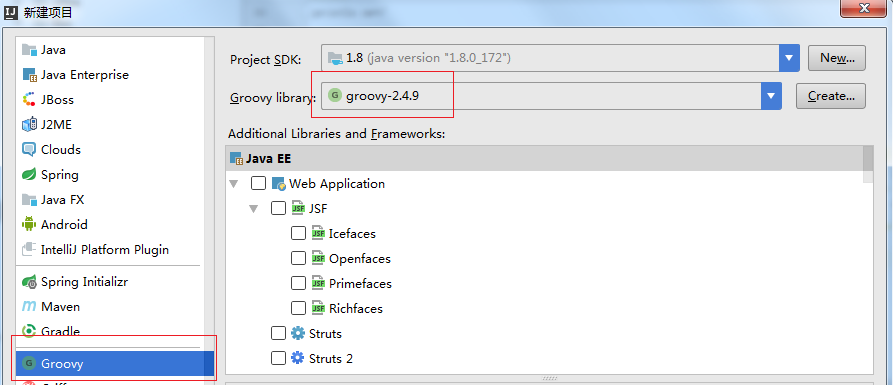


2.搜索plugins，在plugins中搜索groovy

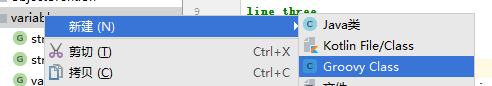


### 3.3 groovy工程创建

1）新建项目，选择Groovy，并选择本地解压的Groovy SDK根目录



1. 新建groovy class



3）输入println “Hello Groovy”，直接运行

在Windows下的cmd窗口中运行grails RunApp命令，但是报错了，错误如上，而且在开发工具中是可以正常启动的，找了很多原因都不对，最后都重新装系统了，后来发现了问题：在cmd的编码会影响命令行的运行，默认情况下cmd的编码是gbk，我改成了utf-8，才导致了这个问题，真的是太纠结了，下次引以为戒吧！附改cmd编码的命令：

ERROR: JAVA\_HOME is set to an invalid directory: D:\\*\*\soft\jdk1.8\jdk

Please set the JAVA\_HOME variable in your environment to match the

location of your Java installation.

Please set the JAVA\_HOME variable in your environment to match the

location of your Java installation.

chcp 查看当前编码

chcp 65001 就是换成UTF-8代码页，还需要点击边框右键-属性-字体-改为Lucida Console

chcp 936 可以换回默认的GBK

chcp 437 是美国英语

## 第4节 Gradle核心语法讲解及实战

4.2 基础语法讲解

4.3 String讲解

4.4 字符串方法详解

4.5 逻辑控制

4.6 闭包讲解之基础讲解

4.7 闭包使用讲解

4.8 字符串与闭包结合使用

4.9 闭包进阶讲解

4.10 列表学习(上)

4.11 列表学习(下)

4.12 映射详解(上)

4.13 映射详解(下)

4.16 面向对象学习(中)

4.17 面向对象学习(下)

### 4.14范围详解

|  |
| --- |
| **package** datastruct  **def** range = 1..10 println range[0] println range.contains(10) println range.from println range.to |
| 1  true  1  10 |

|  |
| --- |
| **def** result = getGrade(75) println result  **def** getGrade(Number number) {  **def** result  **switch** (number) {  **case** 0..<60:  result = **'不及格'  break  case** 60..<70:  result = **'及格'  break  case** 70..<80:  result = **'良好'  break  case** 80..100:  result = **'优秀'  break** }   **return** result } |
| **良好** |

### 4.15面向对象

定义一个Person

|  |
| --- |
| **package** objectorention  */\*\*  \* 1.groovy中默认都是public  \*/* **class** Person **implements** Serializable {  String **name** Integer **age  def** increaseAge(Integer years) {  **this**.**age** += years  }  */\*\*  \* 一个方法找不到时，调用它代替  \** ***@param*** *name  \** ***@param*** *args  \** ***@return*** *\*/* **def** invokeMethod(String name, Object args) {  **return "the method is** ${name}**, the params is** ${args}**"** }  **def** methodMissing(String name, Object args) {  **return "the method** ${name} **is missing"** } } |

定义一个脚本

|  |
| --- |
| **package** objectorention  *//def person = new Person(name: 'Qndroid', age: 26) //println person.cry()* ExpandoMetaClass.*enableGlobally*() *//为类动态的添加一个属性* Person.metaClass.sex = **'male' def** person = **new** Person(**name**: **'Qndroid'**, **age**: 26) println person.sex person.sex = **'female'** println **"the new sex is:"** + person.sex *//为类动态的添加方法* Person.metaClass.sexUpperCase = { -> sex.toUpperCase() } **def** person2 = **new** Person(**name**: **'Qndroid'**, **age**: 26) println person2.sexUpperCase() *//为类动态的添加静态方法* Person.metaClass.static.createPerson = {  String name, **int** age -> **new** Person(**name**: name, **age**: age) } **def** person3 = Person.createPerson(**'renzhiqiang'**, 26) println person3.name + **" and "** + person3.age |
| **male**  **the new sex is:female**  **MALE**  **renzhiqiang and 26** |

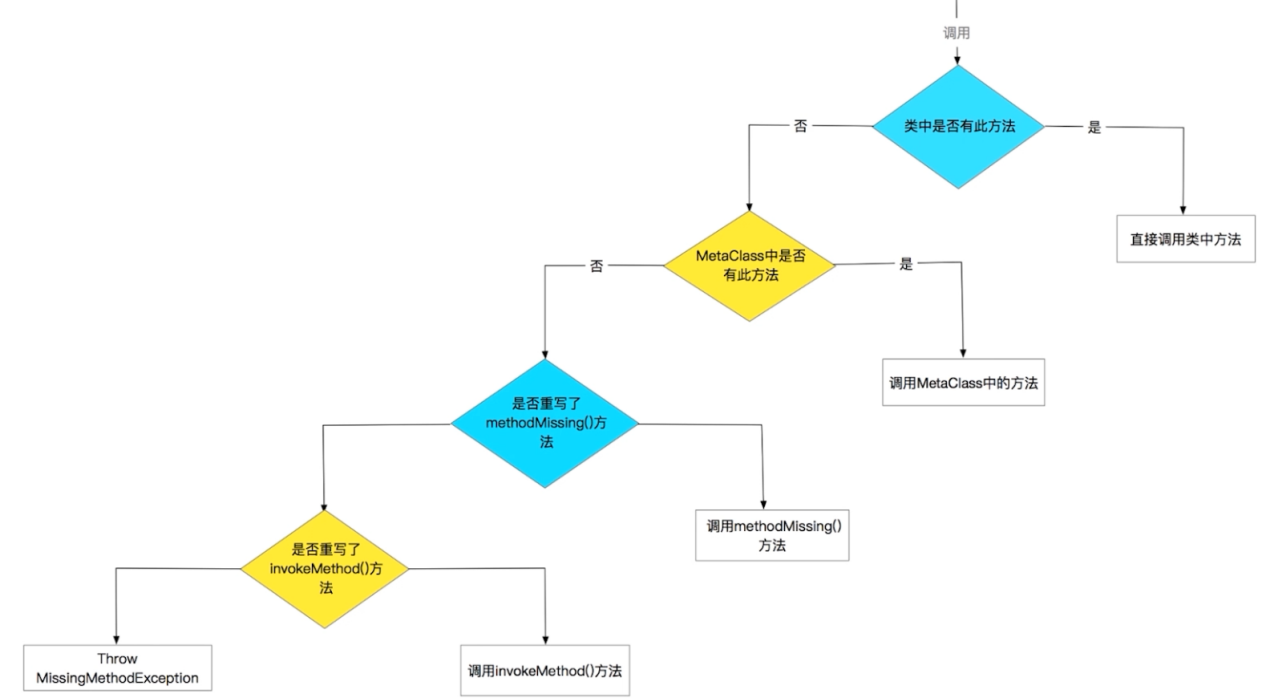
定义一个接口

|  |
| --- |
| **package** objectorention */\*\*  \* 接口中不许定义非public的方法  \*/* **interface** Action {  **void** eat()  **void** drink()  **void** play() } |

定义一个trait接口（适配器模式）

|  |
| --- |
| **trait** DefualtAction {  **abstract void** eat()  **void** play() {  println **' i can play.'** } } |

Groovy的运行时编译



定义一个Person管理器

|  |
| --- |
| **package** objectorention.expand  **import** objectorention.Person  */\*\*  \* Person类的管理器  \*/* **class** PersonManager {  **static** Person createPerson(String name, **int** age) {  **return** Person.createPerson(name, age)  } } |

定义一个应用的init

|  |
| --- |
| **package** objectorention.expand  **import** objectorention.Person  */\*\*  \* 模拟一个应用的管理类  \*/* **class** ApplicationManager {   **static void** init() {  ExpandoMetaClass.*enableGlobally*()  *//为第三方类添加方法* Person.metaClass.static.createPerson = { String name, **int** age ->  **new** Person(**name**: name, **age**: age)  }  } } |

定义一个Main

|  |
| --- |
| **package** objectorention.expand  **class** Entry {  **static void** main(**def** args) {  println **'应用程序正在启动...'** *//初始化* ApplicationManager.*init*()  println **'应用程序初始化完成...'   def** person = PersonManager.  *createPerson*(**'renzhiqiang'**, 26)  println **"the person name is** ${person.name} **"** +  **"and the age is** ${person.age}**"** } } |
| **应用程序正在启动...**  **应用程序初始化完成...**  **the person name is renzhiqiang and the age is 26** |

## 第5节 Gradle高级用法实战

### 5.1JSON操作详解

JsonOutput、JsonSplur

|  |
| --- |
| **package** file  **import** groovy.json.JsonOutput **import** groovy.json.JsonSlurper **import** objectorention.Person   **def** list = [**new** Person(**name**:**'xiaohong'**,**age**: 23),  **new** Person(**name**:**'xiaoming'**,**age**: 24) ] **def** json = JsonOutput.*toJson*(list)  **def** jsonSlpur = **new** JsonSlurper() println jsonSlpur.parseText(json) println json println JsonOutput.*prettyPrint*(json) |
| **[[age:23, name:xiaohong], [age:24, name:xiaoming]]**  **[{"age":23,"name":"xiaohong"},{"age":24,"name":"xiaoming"}]**  **[**  **{**  **"age": 23,**  **"name": "xiaohong"**  **},**  **{**  **"age": 24,**  **"name": "xiaoming"**  **}**  **]** |

从jsonplaceholder获取json格式数据

|  |
| --- |
| **package** file  **import** groovy.json.JsonSlurper  **def** reponse =  getNetworkData(  **'http://jsonplaceholder.typicode.com/albums/6'**)  println reponse println JsonOutput.*prettyPrint*(JsonOutput.*toJson*(reponse))  **def** getNetworkData(String url) {  *//发送http请求* **def** connection = **new** URL(url).openConnection()  connection.setRequestMethod(**'GET'**)  connection.connect()  **def** response = connection.content.text  *//将json转化为实体对象* **def** jsonSluper = **new** JsonSlurper()  **return** jsonSluper.parseText(response) } |
| **[userId:1, id:6, title:natus impedit quibusdam illo est]**  **[userId:1, id:6, title:natus impedit quibusdam illo est]**  **{**  **"userId": 1,**  **"id": 6,**  **"title": "natus impedit quibusdam illo est"**  **}** |

### 5.2xml文件解析详解

1. 解析一个xml文件

|  |
| --- |
| **package** file  **import** groovy.xml.MarkupBuilder  **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>  </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> '''** *//开始解析此xml数据* **def** xmlSluper = **new** XmlSlurper() **def** response = xmlSluper.parseText(xml)  println response.value.books[0].book[0].title.text() println response.value.books[0].book[0].author.text() println response.value.books[1].book[0].@available |
| **疯狂Android讲义**  **李刚**  **10** |

遍历xml上的节点

|  |
| --- |
| **def** list = [] response.value.books.each { books ->  *//下面开始对书结点进行遍历* books.book.each { book ->  **def** author = book.author.text()  **if** (author.equals(**'李刚'**)) {  list.add(book.title.text())  }  } }  println list.toListString() |
| **[疯狂Android讲义, Vue从入门到精通]** |

深度遍历xml数据.depthFirst(). 等价于 .'\*\*'.

|  |
| --- |
| *//深度遍历xml数据.depthFirst(). 等价于 .'\*\*'.* **def** titles = response.depthFirst().findAll { book ->  **return** book.author.text() == **'李刚'** ? **true** : **false** } println titles.toListString() |
| **[疯狂Android讲义李刚, Vue从入门到精通李刚]** |

广度遍历xml上的节点 .children(). 等价于 .’\*’.

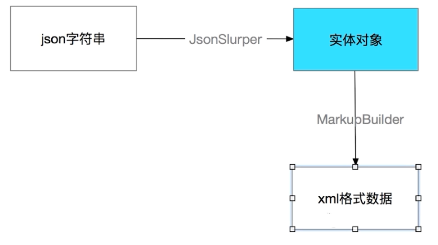
|  |
| --- |
| *//广度遍历xml数据* **def** name = response.value.books.children().findAll { node ->  node.name() == **'book'** && node.@id == **'2'** }.collect { node ->  **return** node.title.text() }  println name |
| **[第一行代码]** |

1. 创建一个xml文件

|  |
| --- |
| */\*\*  \* 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 //根结点langs创建成功* xmlBuilder.langs(**type**: **'current'**, **count**: **'3'**,  **mainstream**: **'true'**) {  *//第一个language结点* language(**flavor**: **'static'**, **version**: **'1.5'**) {  age(**'16'**)  }  language(**flavor**: **'dynamic'**, **version**: **'1.6'**) {  age(**'10'**)  }  language(**flavor**: **'dynamic'**, **version**: **'1.9'**, **'JavaScript'**) }  println sw |
| **<langs type='current' count='3' mainstream='true'>**  **<language flavor='static' version='1.5'>**  **<age>16</age>**  **</language>**  **<language flavor='dynamic' version='1.6'>**  **<age>10</age>**  **</language>**  **<language flavor='dynamic' version='1.9'>JavaScript</language>**  **</langs>** |

对象转xml

|  |
| --- |
| **def** langs = **new** Langs() xmlBuilder.langs(**type**: langs.type, **count**: langs.count,  **mainstream**: langs.mainstream) {  *//遍历所有的子结点* langs.languages.each { lang ->  language(**flavor**: lang.flavor,  **version**: lang.version, lang.value)  } } println sw *//对应xml中的langs结点* **class** Langs {  String **type** = **'current'  int count** = 3  **boolean mainstream** = **true  def languages** = [  **new** Language(**flavor**: **'static'**,  **version**: **'1.5'**, **value**: **'Java'**),  **new** Language(**flavor**: **'dynamic'**,  **version**: **'1.3'**, **value**: **'Groovy'**),  **new** Language(**flavor**: **'dynamic'**,  **version**: **'1.6'**, **value**: **'JavaScript'**)  ] } *//对应xml中的languang结点* **class** Language {  String **flavor** String **version** String **value** } |
| **<langs type='current' count='3' mainstream='true'>**  **<language flavor='static' version='1.5'>Java</language>**  **<language flavor='dynamic' version='1.3'>Groovy</language>**  **<language flavor='dynamic' version='1.6'>JavaScript</language>**  **</langs>** |



### 5.3groovy文件处理详解