# **Grails 开发手册**

2017年05月10日 16:26:19

1. 简介 1

2. 安装 Grails 1

3. 基本命令 1

4. 目录结构 2

5. Domain 2

6. Controller 5

7. Service 7

8. Filter 8

9. Plugin 10

10. 整合前端框架 JQuery EasyUI 11

11. 配置说明 11

12. 碰到的问题 14

## 1. ****简介****

Grails 是根据约定优于配置的原则来搭建，整个框架其实也是对java比较流行的框架进行了封装，MVC部分是用的Spring MVC，持久层则是用的Hibernate。但整个开发的模式还有很大的变化，它的开发语言是Groovy，并且不需要自己去写数据访问层来实现基本CRUD功能，这些都已经包含在Domain里。

## 2. ****安装 Grails****

a) 下载 [http://grails.org/download](http://grails.org/download" \t "http://blog.csdn.net/hxpjava1/article/details/_blank)

b) 创建一个GRAILS\_HOME的环境变量

c) 将 $GRAILS\_HOME/bin 添加到PATH中

安装完成后可以输入grails –version进行检查。

## 3. ****基本命令****

d) 创建项目

grails create-app [appName]

生成一个基础的Grails项目目录。

e) grails create-domain-class [className]

创建一个Domain类

f) grails create-controller [controllerName]

创建一个Controller类

g) grails create-service [serviceName]

创建一个service类

h) grails compile

编译Groovy 及 Java 源码

i) clean

清除编译文件，在idea下开发Grails项目时新加jar包后可能会出现项目启动缺少jar包的情况，需要clean一下。

j) grails list-plugins

列出所有可用的插件

k) grails install-plugin [pluginName]

安装插件

l) grails uninstall-plugin  [pluginName]

删除已安装的插件

m) grails war

将项目打成war包，方便部署到web应用服务器

## 4. ****目录结构****

l grails-app 项目源码的顶级目录

l conf 配置文件

l controllers 控制器文件

l domain 域文件

l i18n 国际化支持

l services 服务层文件

l taglib 页面自定义标签库

l views 视图层文件

l lib   jar包

l scripts 脚本文件

l src  Groovy与Java源文件

l test 单元测试文件

## 5. ****Domain****

n) 定义属性

i. 例子

class Person {

        String name

        Integer age

        Date lastVisit

}

o) CRUD操作

i. Create新增

def p=**new** Person(name:"Fred", age:40, lastVisit:**new**Date())

p.save()

ii. Read读取

Grails 是为domain类默认一个id属性。

def p = Person.get(1)

加载一个只读的对象

def p = Person.read(1)

iii. Update更新

def p = Person.get(1)

p.name = "Bob"

p.save()

iv. Delete删除

def p = Person.get(1)

p.delete()

p) 关联

i. one-to-one

class Face {

    Nose nose

}

class Nose {

        Face face

}

ii. one-to-many

class Author {

static hasMany = [ books : Book ]

String name

}

class Book {

String title

 }

对于 hasMany 设置，Grails将自动注入一个java.util.Set类型的属性到domain类。 用于迭代集合:

def a = Author.get(1)

a.books.each { println it.title }

默认的级联行为是级联保存和更新，但不删除，除非 belongsTo 被指定:

class Author {

static hasMany = [ books : Book ]

String name

}

class Book {

static belongsTo = [author:Author]

String title

}

iii. many-to-many

通过在关联双方定义 hasMany ，并在关联拥有方定义 belongsTo

class Book {

 static belongsTo = Author

 static hasMany = [authors:Author]

 String title

}

class Author {

static hasMany = [books:Book]

String name

}

在这种情况下，Author 负责持久化关联，并且是唯一可以级联保存另一端的一方 。例如，下面这个可以进行正常级联保存工作:

new Author(name:"Stephen King").addToBooks(new Book(title:"The Stand")).addToBooks(new Book(title:"The Shining")).save()

iv. domain 映射配置

static mapping = {

        id column:'goods\_id'  //id字段与数据库goods\_id字段关联

        goodsSuppCode column: 'goodssuppcode'

        goodsSuppName column: 'goodssuppname'

        goodsSuppLink column: 'goodssupplink'

        table 'who\_goods'  //表名（默认是类名）

        version false //是否加version字段，用于乐观锁

        datasources(['brand','home']) //数据源配置

}

v. domain 数据验证

在domain中对属性设置一些验证规则，在执行save操作时会进行验证。如：

static constraints = {

        tmallId(unique: true) //唯一性

        goodsSn(blank: true) //可否为空

  login(size:5..15, blank:**false**, unique:**true**)

  password(size:5..15, blank:**false**)

  email(email:**true**, blank:**false**) age(min:18, nullable:**false**)

}

vi. 基于domain的查询

1) 使用Hibernate CriteriaBuilder 进行查询，例子如下：

def searchClosure = {

if (params.qStatus) {

       eq('status',Integer.parseInt(params.qStatus))

             }

}//过滤条件status  = 传入的qStatus

def c = GoodsTmallInfo.createCriteria()

def relates = c.list(params, searchClosure)

2) 使用domain中的find进行查询

GoodsTmallRelate.findAllByIsAvaliableAndUpdateTimeLessThanEquals(1,time,[offset:0,max:20,sort:"updateTime",order: "desc"])

## 6. ****Controller****

控制器处理请求并返回响应，调用业务逻辑返回视图或者文本。控制器位于 grails-app/controllers 目录下，创建控制器的命令是 grails create-controller User，这就创建了一个控制器类UserController

class UserController {

    def index() { }

}

一个控制器里可以定义多个操作，用于处理不同的请求url。默认情况下，一个控制器名和一个操作名组合而成的url，对应了相应的控制器和处理方法，比如 /user/list对应的处理类就是UserController和处理方法list。也可以通过修改配置文件grails-app/conf/UrlMappings.groovy来修改默认的映射关系。默认情况下，每次请求都会新建一个Controller实例，也可以修改默认的生命周期。

3) ****控制器中可用的变量****

a) [request](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/request.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – http请求对象

b) [response](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/response.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – http响应对象

c) [session](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/session.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – 会话对象

d) [servletContext](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/servletContext.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) -  ServletContext 对象，整个应用共享的对象

e) [flash](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/flash.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – flash对象，该对象的生命周期是两次请求，下次请求到来后即消失

f) [params](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/params.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – 请求参数对象，可以从中获取参数

g) [actionName](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/actionName.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – 当前正在被调用的操作名

h) [controllerName](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/controllerName.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – 当前正在被调用的控制器名

i) [grailsApplication](C:/Documents and Settings/Administrator/%E6%A1%8C%E9%9D%A2/grailsDoc/grails-doc/ref/Controllers/grailsApplication.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – 当前正在运行的应用对象，可以从中获取application.properties文件中的配置

j) [applicationContext](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/context/ApplicationContext.html" \t "http://blog.csdn.net/hxpjava1/article/details/_blank) – Spring上下文对象

4) **请求转发渲染**render****

a) 直接返回文本 ： renders “Hello”

b) 转发到一个视图，并且传入model参数。render(view: "viewName", model: [book: theShining])

c) 转发到一个视图，并且以当前控制器作为model参数。render(view: "viewName")

d) 返回json格式的文本。render Book.list(params) as JSON

e) 返回xml格式的文本。render Book.get(params.id) as XML

5) **重定向**redirect****

a) 重定向到当前控制器的一个操作 :redirect(action: "show")，表示重定向到当前控制器的show方法

b) 重定向到某个控制器的某个方法：redirect(controller: "book", action: "list")

c) 带参数的重定向：redirect(action: "show", id: 4, params: [author: "Stephen King"])

d) 重定向到uri：redirect(uri: "book/list")

e) 重定向到其他域名：redirect(url: "http://www.baidu.com")

6) **URL映射**

默认情况下一个url：/controller/action/id对应了处理的控制器和处理方法，以及传入的id参数。通过修改grails-app/conf/UrlMappings.groovy文件来修改默认配置。

a) 映射到控制器和操作："/product"(controller: "product", action: "list")

b) 映射到uri："/hello"(uri: "/hello.dispatch")

c) 映射到视图："/hello"(uri: "/hello.dispatch")，直接映射到GSP文件grails-app/views/index.gsp

d) 映射到响应码：

static mappings = {

   "403"(view: "/errors/forbidden")

   "404"(view: "/errors/notFound")

   "500"(view: "/errors/serverError")

}

当http响应码为不同的数字时，对应不同的gsp页面。

e) 映射到http方法：

static mappings = {

   "/product/$id"(controller:"product") {

       action = [GET:"show", PUT:"update", DELETE:"delete", POST:"save"]

   }

}

当请求方法为GET时对应的处理方法为show，PUT对应的处理方法是update，以此类推。

## 7. ****Service****

一般复杂的业务逻辑处理放在服务层，在Controller中调用。

创建服务的命令是grails create-service user

该命令在目录grails-app/services/下创建一个服务Class，UserService.groovy

默认情况下，grails只创建一个实例，并且调用方法是非同步的。grails自动定义一个bean，实例名即是将类名首字母小写的字符串。可以定义scope变量来设置Service的生命周期。

· prototype – 每次调用都创建一个新的实例

· request –每一个请求创建一个新的实例

· flash – 只有当前请求和下次请求对以一个实例

· flow – 同一个webflow对以一个实例

· conversation – 同一个webflow会话对以一个实例

· session – 同一个用户会话对应一个实例

· singleton (default) – 整个应用公用一个实例，默认

**事务处理**：默认情况下，Service中的每一个方法都在同一个事务管理之下，默认事务的传播级别是[PROPAGATION\_REQUIRED](http://blog.csdn.net/hxpjava1/article/details/71545599" \l "PROPAGATION_REQUIRED" \t "http://blog.csdn.net/hxpjava1/article/details/_blank)。可以通过设置变量

static transactional = false 取消事务。

注意事项：

l 只有通过依赖注入的Service才有事务处理，即只有通过def userService这种方式获取才有事务处理，如果是通过New UserService()新建的Service对象没有事务处理。

l Service中的方法只有抛出java.lang.RuntimeException，事务才会回滚。即Service中抛出的异常必须是java.lang.RuntimeException的子类，而不能是java.lang. Exception子类，否则事务将失效。

**Service调用**：可以直接在Controller中引用Service

class UserController {

    def userService

    …

}

也可以在其他的Servcie或者Domain中直接引用。

也可以在java中引用Service bean，通过grails-app/conf/spring/resources.xml中依赖注入

<bean id="bookConsumer" class="bookstore.BookConsumer">

    <property name="userStore" ref="userService" />

</bean>

groovy sql事务：

def sql = new Sql(dataSource\_brand)

sql.withTransaction {

            sql.executeUpdate("update who\_goods\_stock g set g.num = g.num - :outNum where g.num - g.lock\_num >= :outNum and g.id = :stockid",[outNum:params.outNum,stockid:params.stockId])

        }

抛出异常时回滚

## 8. ****Filter****

过滤器即servlet中的Filter，通过设置规则来限定某些url请求和响应必须经过Filter的拦截，一般用于登陆验证、编码转换或者日志处理等。过滤器是位于grails-app/conf 目录下的groovy类。Grails中创建过滤器的命令是

grails create-filters Security

class SecurityFilters {

    def filters = {

        loginCheck(controller: '\*', action: '\*') {

            before = {

                if (!session.user && !actionName.equals('login')) {

                    redirect(action: 'login')

                    return false

                }

            }

        }

    }

}

控制器Controller能使用的内置变量，在Filter中都可以使用。

过滤器匹配规则可以是按照controller名字或者action名字，也可以根据uri匹配

a) 根据控制器名和操作名：

all(controller: '\*', action: '\*') {

}

匹配所有的控制器和操作

b) 只匹配控制器：

justBook(controller: 'book', action: '\*') {

}

只要控制器名为book即匹配

c) 使用反转关键字

notBook(controller: 'book', invert: **true**) {

}

只要控制器名不是book即匹配

d) 根据操作名匹配

saveInActionName(action: '\*save\*', find: **true**) {

}

只要操作名包含save即匹配

e) 组合匹配

actionBeginningWithBButNotBad(action: 'b\*', actionExclude: 'bad\*', find: **true**) {

}

操作名以b开头，但是不能以bad开头

f) 根据uri匹配

someURIs(uri: '/book/\*\*') {

}

以book开头的uri即可

g) 匹配所有的uri

allURIs(uri: '/\*\*') {

}

h) 排除部分uri

all(controller:'\*', action:'\*',uriExclude:'/tbAuth/\*') {

}

排出了以/tbAuth/开头的uri

关键字说明：

controller ：控制器名，\*表示所有控制器

controllerExclude：排出的控制器名

action：匹配的操作名

actionExclude ：排除的操作名

regex ：是否启用正则表达式，true或者false，默认true

uri ：匹配的uri，如 /user/list

uriExclude：排除的uri，等价于uri:/user/list,invert:true

invert:是否反转规则，默认false

find ：是否按照正则表达式匹配，默认true

注意：全部用\*，不要用.\*

过滤器可以在请求处理过程中多个不同位置进行拦截处理

拦截器类型：

before:请求到来后，操作执行之前执行

after:操作执行后，视图渲染之前执行，可以修改model参数

afterView:视图渲染之后，布局之前执行，可以处理异常参数

## 9. ****Plugin****

q) **插件机制是**Grails的一个重要特性，它提供了大量的插件可以对项目方便的进行扩展。****

r) ****关于插件的命令：****

i. ****grails list-plugins 查看可用的插件****

ii. ****grails install-plugin [插件名] 安装插件****

iii. ****grails uninstall-plugin [插件名]  删除插件****

s) **实例使用**Quartz定时插件****

i. **首先安装**Quartz 插件****

ii. **安装完后会在**grails-app目录下生成jobs目录****

iii. **创建定时任务，**grails create-job [任务名]****

iv. ****定义执行时间规则****

1) **定义间隔时间：**static triggers = {simple repeatInterval: 28800000l // 执行间隔 8小时}****

2) **或用**Cron表达式更灵活的配置：****

****static triggers = {cron name: 'myTrigger', cronExpression: "0 0/1 \* \* \* ?"}****

**推荐一下**Cron表达式在线生成器：****

****http://www.becron.com/****

## 10. ****整合前端框架 JQuery EasyUI****

t) ****在GSP页面引入 EasyUI的js包。****

u) ****在不修改Easy UI js包的前提下，修改Easy UI关于分页的参数与GSP页面默认的分页参数进行结合。****

****//分页参数****

Integer num = params.rows as Integer//每页记录数

****params.max = Math.min(num, 100) //限制每页最大100条****

****params.offset =  page == 1?0:(page-1)\*num //开始条数****

v) ****返回json格式的数据，并调整为适应Easy UI的格式****

****render(contentType: "text/json") {****

****rows = array { //列表内容****

****for (GoodsTmallRelate b in relates) {****

****tmallInfo id:b.id,****

****goodsId: b.goodsId ,****

****}****

****}****

****total = relates.totalCount  //总记录数****

****}****

## 11. ****配置说明****

w) Config.groovy

i. 配置GSP页面修改实时生效，可区分环境进行分别配置

environments {

    development {

        grails.logging.jul.usebridge = true

        //使修改的页面实时生效

        grails.gsp.enable.reload = true

        def  classpath = "${appName}/WEB-INF/grails-app";

        grails.reload.location=classpath

    }

    production {

        grails.logging.jul.usebridge = false

    }

}

ii. log4j的相关配置

log4j = {

   appenders {

        rollingFile name:"file", maxFileSize:1024, file:"web-app/log/info.log"

        console name: 'stdout'  ,layout: pattern(conversionPattern: "%d [%p] %l%n%m%n%n")

        appender new DailyRollingFileAppender(

                name: "dailyFile",

                file: "web-app/log/info.log",

               datePattern: "'.'yyyy-MM-dd",

              layout: pattern(

                        conversionPattern:

                                "%d [%p] %l%n%m%n%n"

                )

        )

    }

    environments {

        development { //开发环境

            root {

                info  'stdout'

                additivity = true

            }

        }

        production {  //生产环境

            root {

                info 'stdout','dailyFile'

                additivity = true

            }

        }

        test {    //测试环境

            root {

                debug 'dailyFile'

                additivity = true

            }

        }

    }

error  'org.codehaus.groovy.grails.web.servlet',        // controllers

            'org.codehaus.groovy.grails.web.pages',          // GSP

            'org.codehaus.groovy.grails.web.sitemesh',       // layouts

            'org.codehaus.groovy.grails.web.mapping.filter', // URL mapping

            'org.codehaus.groovy.grails.web.mapping',        // URL mapping

            'org.codehaus.groovy.grails.commons',            // core / classloading

            'org.codehaus.groovy.grails.plugins',            // plugins

            'org.codehaus.groovy.grails.orm.hibernate',      // hibernate integration

            'org.springframework',

            'org.hibernate',

            'net.sf.ehcache.hibernate'

}

iii. hibernate 缓存配置

grails.hibernate.cache.queries = false

默认不使用缓存，但可在重要需要缓存的查询后面加上 cache: true 进行缓存

x) DataSource.groovy  数据源配置文件

1.可以把数据库地址用户名等配置到一个properties文件中，方便部署后的更改。

def dataConf = PropertiesLoaderUtils.loadProperties(new ClassPathResource('application.properties'))

2. 可以分为开发环境，生产环境，测试环境分别配置相应的数据源。

3. 一个环境可以配置多个数据源，在domain类中声明可用于哪几个数据源。持久化操作时可以选择相应的数据源进行操作。

static mapping = {

        datasources(['brand','home']) //在domain中进行多数据源的配置

}

Goods.brand.get(id)  //使用时可以选择不同的数据源进行操作

4. 数据源配置可以开启使用连接池，默认使用DBCP连接池。

  pooled = true  //开启连接池

            properties{   //连接池相关配置

                maxActive = 50

                maxIdle = 25

                minIdle = 5

                initialSize = 5

                minEvictableIdleTimeMillis = 60000

                timeBetweenEvictionRunsMillis = 60000

                maxWait = 10000

                validationQuery = "/\* ping \*/"

            }

y) UrlMappings.groovy

i. 配置访问连接与资源的映射。

默认为 http://domain/controller/action/param

z) resources.groovy

i. 定义一些需要spring控制的bean

## 12. ****碰到的问题****

aa) 不通过domain直接操作数据库

i. 直接在service里定义def dataSource变量，Grails会自动注入。

ii. 然后获取数据库链接 def conn = dataSource.getConnection()

iii. 通过jdbc操作数据库

1) PreparedStatement ps = conn.prepareStatement("SELECT s.shipping\_id,s.`shipping\_code`,s.`shipping\_remark`,s.`logo\_image`,s.`calc\_rule`,s.`weight\_limit`,s.`lenth\_limit`,s.`first`,s.`additional`,sa.shipping\_area\_id,sa.first\_price,s.shipping\_name FROM `who\_shipping` s RIGHT JOIN (SELECT \* FROM `who\_shipping\_area` sa WHERE sa.`enabled`=1 AND sa.`shipping\_area\_id` IN (SELECT ar.`shipping\_area\_id` FROM `who\_shipping\_area\_region` ar WHERE ar.`region\_id` = ?)) AS sa ON s.shipping\_id = sa.`shipping\_id` ORDER BY s.`sort`");

2) ResultSet rs = ps.executeQuery();

ab) 定时任务重复执行

i. 通过自动部署工具部署Grails时，会发生Grails定时插件Quartz定时任务重复执行两次的情况。

1) 手工部署就没有这个问题

2) 修改tomcat的server.xml

a) <Host name="localhost" debug="0" appBase="" unpackWARs="true" autoDeploy="true">将 appBase留空。

## 13. ****其它****

a) 修改groovy代码后自动编译生效

i. 在Idea的Edit configurations 中配置VM options 如下：

-server

-javaagent:/opt/local/share/java/grails/lib/org.springsource.springloaded/springloaded-core/jars/springloaded-core-1.1.1.jar

-noverify

-Dspringloaded=profile=grails

黄色部分路径根据实际情况修改