Groovy & Grails: Desarrollo rápido de aplicaciones

Sesión 12: Configuración y despliegue de aplicaciones

Configuración y despliegue de aplicaciones

- Configuración de aplicaciones
- Empaquetamiento de aplicaciones
- Actualización de aplicaciones
- Tareas programadas con Quartz
- Otros comandos interesantes de Grails

Configuración de aplicaciones

- El archivo *Config.groovy*
- El archivo *DataSource.groovy*
- El archivo *BootStrap.groovy*
- El archivo *UrlMappings.groovy*

Configuración de aplicaciones

- **El archivo *Config.groovy***
  - Contiene los parámetros de configuración general de nuestra aplicación
  - Las variables aquí declaradas estarán disponibles en cualquier artefacto de la aplicación a través del objeto global *grailsApplication.config*

```
com.biblioteca.miParametro = "dato"

grailsApplication.config.com.biblioteca.miParametro
```

Configuración de aplicaciones

- **El archivo *Config.groovy***
  - Grails tiene una serie de variables definidas

Variable	Descripción
grails.config.location	Ubicaciones donde encontrar otros archivos de configuración que se fundirán con el principal *Config.groovy*
grails.enable.native2ascii	Indica si Grails utiliza native2ascii para convertir los archivos *properties* al formato unicode
grails.views.default.codec	Especifica el formato por defecto de nuestras páginas GSP. Puede ser 'none', 'html' o 'base64'
grails.views.gsp.encoding	Codificación de las páginas GSP

Configuración de aplicaciones

- El archivo *Config.groovy*

Variable	Descripción
grails.converters.encoding	Codificación de los convertidores
grails.mime.file.extensions	Habilita el uso de la extensión en la ur para fijar el *content-type* de la respuesta.
grails.mime.types	Indica un mapa con los posibles tipos mime soportados en nuestra aplicación
grails.serverUrl	La parte “fija” de nuestros enlaces cuando queremos generar rutas absolutas

Configuración de aplicaciones

- **El archivo *DataSource.groovy***
  - Varios entornos que las aplicaciones deben superar
    - *Entorno de desarrollo*
    - *Entorno de tests*
    - *Entorno de producción*

Configuración de aplicaciones

- **El archivo *DataSource.groovy***
  - Cada entorno tendrá su propia configuración
  - En el archivo *grails-app/conf/DataSource.groovy* podemos establecer estas diferencias de configuración para cada entorno

Configuración de aplicaciones

- **El archivo *DataSource.groovy***

```
dataSource {
        pooled = true
        driverClassName = "org.hsqldb.jdbcDriver"
        username = "sa"
        password = ""
}
hibernate {
        cache.use_second_level_cache=true
        cache.use_query_cache=true
        cache.provider_class=
        'com.opensymphony.oscache.hibernate.OSCacheProvider'
}
```

Configuración de aplicaciones

- **El archivo *DataSource.groovy***

```
// environment specific settings
environments {
        development {
                dataSource {
                        dbCreate = "create-drop"
                        url = "jdbc:hsqldb:mem:devDB"
                }
        }
        ....
}
```

Configuración de aplicaciones

- **El archivo *DataSource.groovy***

```
environments {
        ....
        test {
                dataSource {
                        dbCreate = "update"
                        url = "jdbc:hsqldb:mem:testDb"
                }
        }
        ....
}
```

Configuración de aplicaciones

- **El archivo *DataSource.groovy***

```
environments {
        ....
        production {
                dataSource {
                        pooled = true
                        dbCreate = "update"
                        url = "jdbc:mysql://localhost/biblioteca"
                        driverClassName = "com.mysql.jdbc.Driver"
                        username = "user_biblioteca"
                        password = "pwdbiblio"
                }
        }
}
```

Configuración de aplicaciones

- **El archivo *DataSource.groovy***
  - Por defecto Grails crea los tres entornos que comentábamos anteriormente
  - Con *grails run-app* el entorno que se ejecuta es el de desarrollo

Entorno	Comando
Desarrollo	*grails dev run-app* o *grails run-app*
Test	*grails test run-app*
Producción	*grails prod run-app*

Configuración de aplicaciones

- **El archivo *DataSource.groovy***
  - Para el entorno de producción utilizaremos una base de datos MySQL
  - Especificamos:
    - URI
    - Controlador
    - Nombre de usuario
    - Contraseña

Configuración de aplicaciones

- **El archivo *DataSource.groovy***
  - *dbCreate*, especifica la forma en la que Grails crea el esquema de la base de datos

Valor	Descripción
create-drop	El entorno se crea y se destruye al arrancar y parar la aplicación
create	Crea la base de datos si no existe, pero si existe, simplemente borra los datos
update	Crea la base de datos si no existe, y actualiza los campos en caso de cambios

Configuración de aplicaciones

- **El archivo *BootStrap.groovy***
  - Nos ha servidor para insertar datos de ejemplo
  - Se definen dos *closures, init()* y *destroy()*
  - Permite diferenciar entornos de ejecución gracias al paquete *grails.util.GrailsUtil* y la variable *GrailsUtil.environment*

Configuración de aplicaciones

- **El archivo *BootStrap.groovy***

```
import grails.util.GrailsUtil
class BootStrap {
        def init = {
                servletContext ->
                        switch (GrailsUtil.environment){
                                case "development": configuracionDesarrollo()
                                        break;
                                case "test":configuracionTest()
                                        break;
                                case "production":configuracionProduccion()
                                        break;
                        }
        }....
}
```

Configuración de aplicaciones

- **El archivo *BootStrap.groovy***

```
class BootStrap {
        ....
        def destroy = {
                switch (GrailsUtil.environment){
                        case "development": salirDesarrollo()
                                break;
                        case "test":salirTest()
                                break;
                        case "production":salirProduccion()
                                break;
                }
        }
}
```

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***
  - Este archivo nos sirve para definir nuevas relaciones entre las URLs y los controladores
  - Por defecto, el primer parámetro que sigue al nombre de la aplicación se refiere al controlador
  - El segundo parámetro a la acción
  - El tercer parámetro al identificador de la instancia de la clase de dominio

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***

```
class UrlMappings {
        static mappings = {
                "/$controller/$action?/$id?"{
                        constraints {
                                // apply constraints here
                        }
                }
                "/"(view:"/index")
                        "500"(view:'/error')
        }
}
```

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***
  - Se suele utilizar para permitir URL más limpias y sencillas

http://localhost:8080/biblioteca/libro/3

http://localhost:8080/biblioteca/libro/show/3

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***

```
"/libro/$id"{
        controller = "libro"
        action = "show"
}
```

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***
  - Facilita la internalización de las URLs

http://localhost:8080/biblioteca/libro/3

http://localhost:8080/biblioteca/book/3

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***

```
"/book/$action/$id"{
        controller = "libro"
}

"/user/$action/$id"{
        controller = "usuario"
}

"/operation/$action/$id"{
        controller = "operacion"
}
```

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***
  - Incluir restricciones que deben cumplir las URLs

http://localhost:8080/blog/2009/06/2

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***

```
"/blog/$anyo/$mes/$id" {
        controller = "blog"
        action = "show"
        constraints {
                anyo(matches:/d{4}/)
                mes(matches:/d{2}/)
        }
}
```

Configuración de aplicaciones

- **El archivo *UrlMappings.groovy***
  - Capturar códigos de error

```
"404"(view:'/error')


"404"(controller:'errores', action:'notFound')
```

Empaquetamiento de aplicaciones

- Generación de un archivo *war* con la aplicación completa preparada para ser desplegada en un servidor

```
grails war
```

Empaquetamiento de aplicaciones

1. Actualizar código fuente del repositorio de control de versiones

2. Ejecutar tests de integración, unitarios y funcionales

3. Incrementar la variable *app.version* del archivo *application.properties* con el comando *grails set-version 0.2*

Empaquetamiento de aplicaciones

4. Limpiar el proyecto de temporales con *grails clean*

5. Generar el archivo *war* indicándole el entorno donde queremos desplegar este war, *grails prod war*

Empaquetamiento de aplicaciones

- Un *war* puede ser desplegado en servidores de aplicaciones Java EE:
  - JBoss
  - GlassFish
  - Apache Geronimo
  - BEA Weblogic
  - IBM Websphere
- O en contenedores web como
  - Apache Tomcat
  - Jetty

Empaquetamiento de aplicaciones

- Cada servidor o contenedor web tiene sus propias especificaciones para desplegar las aplicaciones
- Quizás por eso en Grails no existe el comando *grails deploy*
- Sin embargo, podemos crear nuestros propios comandos

Empaquetamiento de aplicaciones

- El directorio *scripts* de la instalación de Grails se ubican todos los scripts que hemos visto hasta ahora
- Nuestro nuevo script lo almacenaremos también en este directorio
- También podríamos guardarlo en *USER_HOME/.grails/scripts, PROJECT_HOME/scripts, PROJECT_HOME/plugins/*/scripts* o en *GRAILS_HOME/scripts*

Empaquetamiento de aplicaciones

- Crearemos un script para automatizar el proceso de despliegue de una aplicación
- El servidor de destino será JBoss
- JBoss permite el despliegue de aplicaciones copiar el archivo *WAR* en un determinado directorio

Empaquetamiento de aplicaciones

```
/** * Script Gant que copia un archivo WAR
* en un directorio de despliegue de un servidor
*/
Ant.property(environment:"env")
grailsHome = Ant.antProject.properties."env.GRAILS_HOME"

includeTargets << new File ( "${grailsHome}/scripts/War.groovy" )
target ('default':'"Copia un archivo WAR al directorio de despliegue de un
servidor.
 Ejemplo:
   grails deploy
   grails prod deploy
  "') {
      deploy()
}
```

Empaquetamiento de aplicaciones

```
target (deploy: "Despliegue en el servidor") {
        depends( war )

        def deployDir = "jbossserver"

        Ant.copy(todir:"${deployDir}", overwrite:true) {
                fileset(dir:"${basedir}", includes:"*.war")
        }

        event("StatusFinal", ["Archivo WAR copiado en ${deployDir}"])
}
```

Actualización de aplicaciones

```
grails upgrade
```

- Si la versión de Grails con la que se empaquetó la aplicación, al arrancar la aplicación, nos aparecerá un mensaje para que actualicemos la instalación de Grails

Tareas programadas con Quartz

- La automatización de tareas es algo habitual en las aplicaciones web
- Estas tareas se deben lanzar a determinadas horas del día o cada cierto tiempo
- Normalmente esta automatización se debe realizar fuera de la propia aplicación (*crontab)*

Tareas programadas con Quartz

- En Grails podemos utilizar el plugin *Quartz* para esta automatización de tareas

```
grails install-plugin quartz
```

- Este plugin genera dos nuevos comandos
  - *grails create-job*
  - *grails install-quartz-config*

Tareas programadas con Quartz

```
grails install-quartz-config
```

- Al ejecutar este comando se creará un archivo de configuración en el directorio *grails-app/conf* llamado *QuartzConfig.groovy*
- Podemos controlar las variables *autoStartup* indica si se arranca *quartz* al mismo tiempo que la aplicación
- La variable *jdbcStore* indica si los trabajos deben ser persistidos en la base de datos

Tareas programadas con Quartz

```
grails create-job
```

- El sistema nos pedirá el nombre del trabajo que queremos crear

- Se creará una nueva clase en el directorio *grails-app/jobs* con el nombre dado seguido de la palabra *Job*

Tareas programadas con Quartz

```
class TareasJob {
        def startDelay = 30000
        def timeout = 1000

        def group = "GrupoDeTareas1"

        def execute(){
                print "Ejecuto la tarea programada!"
        }
}
```

Tareas programadas con Quartz

- *startDelay* indica cuanto tiempo en milisegundos debe esperar la tarea por primera vez
- *timeout* indicamos cuanto tiempo debe esperar en milisegundos para volver a ejecutar la tarea
- El método *execute()* se ejecutará cada vez que se lance la tarea

Tareas programadas con Quartz

- Podemos agrupar las tareas gracias a la variable *group*
- Los valores por defecto de *startDelay* y *timeout* son 30 y 60 segundos respectivamente
- Podemos especificar la frecuencia de las tareas en formato *cron* (utilizado en el crontab de linux)

Tareas programadas con Quartz

```
class TareasCronJob {
        def cronExpression = "0 0 15 * * ?"

        def group = "GrupoDeTareas1"

        def execute(){
                print "Ha llegado la hora!"
        }
}
```

Tareas programadas con Quartz

- Las expresiones de tipo cron tienen 6 campos obligatorios más uno opcional

Campo	Valores permitidos	Valores especiales permitidos
Segundos	0-59	, - * /
Minutos	0-59	, - * /
Horas	0-23	, - * /
Día del mes	1-31	, - * / L W
Mes	1-12 o JAN-DEC	, - * /
Día de la semana	1-7 o SUN-SAT	, - * / L #
Año (opcional)	1970-2099	, - * /

Tareas programadas con Quartz

Carácter	Significado
*	Todos los valores
?	Ningún valor determinado
-	Especifica rangos
,	Valores adicionales
/	Especifica incrementos
L	Último valor del campo dado
W	Próximo día de la semana más cercano a un valor dado
#	Sirve para combiar día de la semana con la semana del mes

Tareas programadas con Quartz

Expresión	Significado
0 0 12 * * ?	Todos los días a las 12h
0 15 10 * * ? *	Todos los días a las 10:15h
0 15 10 * * ? 2009	Todos los días a las 10:15h en el 2009
0 * 14 * * ?	Cada minuto desde las 14:00h hasta las 14:59h
0 0/5 14,18 * * ?	Cada 5 minutos desde las 14:00h hasta las 14:59h y cada 5 minutos desde las 18:00h hasta las 18:59h
0 15 10 ? * MON-FRI	Cada lunes, martes, miércoles, jueves y viernes a las 10:15h
0 15 10 L * ?	El última día del mes a las 10:15h
0 15 10 ? * 6L	El último viernes de cada mes a las 10:15h
0 15 10 ? * 6#3	El tercer viernes de cada mes a las 10:15h

Tareas programadas con Quartz

- Podemos ejecutar múltiples tareas en una sola clase
- Debemos configurar la variable *triggers*
  - *simpleTrigger*
  - *cronTrigger*
  - *customTrigger*

Tareas programadas con Quartz

```
class TareasTriggerJob {
        static triggers = {
                simpleTrigger startDelay:10000, timeout: 30000, repeatCount: 10
                cronTrigger startDelay:10000, cronExpression: '0/6 * 15 * * ?'
                customTrigger claseTrigger:MiClaseTrigger,
miParametro:miValor, miOtroParametro:miOtroValor
        }
        def execute() {
                println "Ejecuto una tarea!"
        }
}
```

Envío de notificaciones automaticas

- Nuestra aplicación debe mandar notificaciones automaticas dirigida a aquellos usuarios que no han devuelto el libro a tiempo o a aquellos que tienen reservas a su disposición

- Aprovecharemos el trabajo realizado en el servicio de mensajería

Envío de notificaciones automaticas

```
grails create-job NotificacionOperacion
```

- En esta tarea necesitamos obtener aquellos préstamos activos cuando préstamo haya caducado

Envío de notificaciones automaticas

```
class NotificacionOperacionJob {
        def cronExpression = "0 0 0 * * ?"
        def notificadorService
        def execute(){
        def operacionesCaducadas = Operacion.findAll("from Operacion
as o where o.fechaFin < ? and o.tipo = ? and o.estado = ?", [new Date(),
"prestamo", true])
            if (operacionesCaducadas.size()>0){
                operacionesCaducadas.each {
                    notificadorService.mandarMails(it.usuario.email,"Aviso
de la Biblioteca","El prestamo del libro ${it.libro.titulo} ha caducado")
                }
            }
        }
}
```

Otros comandos interesantes de Grails

Expresión	Significado
grails bug-report	Genera un archivo comprimido en ZIP con los archivos fuente de nuestro proyecto para el caso de que queramos informar de un bug
grails clean	Limpia el directorio *tmp* de nuestra aplicación. Este comando puede ser combinado con otros comandos como por ejemplo *grails clean run-app*
grails console	Nos muestra la consola de Groovy que veíamos en la primera sesión del curso para que podamos hacer nuestras pruebas.
grails doc	Genera la documentación completa de nuestro proyecto.

Otros comandos interesantes de Grails

Expresión	Significado
grails help	Muestra un listado de comandos disponibles en Grails. Si le pasamos como parámetro uno de esos posibles comandos, nos mostrará información adicional sobre el comando dado. Por ejemplo *grails help doc*
grails list-plugins	Muestra un listado completo tanto de los plugins disponibles como de los ya instalados en la aplicación.
grails plugin-info	Muestra la información completa del plugin pasado como parámetro al comando.
grails run-app -https	Ejecuta el comando grails run-app utilizando como servidor Jetty pero sobre un servidor seguro *https*. El puerto por defecto es 8443 y puede ser modificado añadiendo al comando -*Dserver.port.https=<numero_puerto>*

Otros comandos interesantes de Grails

Expresión	Significado
grails schema-export	Genera un fichero con las sentencias SQL necesarias para exportar la base de datos.
grails set-version	Establece la versión de la aplicación. Por ejemplo *grails set-version 1.0.4*
grails stats	Nos muestra una serie de datos referentes a nuestro proyecto, con respecto al número de controladores, clases de dominio, servicios, librerías de etiquetas, tests de integración, etc. y al número de líneas totales en cada apartado.
grails uninstall-plugin	Desinstala el plugin pasado como parámetro de la aplicación.