



# Groovy & Grails: Desarrollo rápido de aplicaciones

## Sesión 7: Controladores

© 2008-2011 Depto. Ciencia de la Computación e IA

# Controladores

- Introducción
- Mejorando los controladores
- Interceptadores de acciones
- Filtros

© 2008-2011 Depto. Ciencia de la Computación e IA

# Introducción

- Ámbitos
- El método render()

# Introducción

- Los controladores:
  - reciben órdenes por parte del usuario
  - gestionan la ejecución de la lógica de negocio
  - actualizan la vista para actualizar el modelo de datos

© 2008-2011 Depto. Ciencia de la Computación e IA

# Introducción

- Los controladores (aplicaciones web):
  - interceptan peticiones HTTP
  - generan la respuesta correspondiente en HTML, XML, JSON, etc
  - o delegan la creación de la vista a una página GSP

© 2008-2011 Depto. Ciencia de la Computación e IA

# Introducción

- Convenio sobre los controladores
  - Se ubican en el directorio *grails-app/controllers*
  - Su nombre termina por la palabra *Controller*

© 2008-2011 Depto. Ciencia de la Computación e IA

# Introducción

- Grails determina que controlador debe actuar en cada momento siguiendo las reglas establecidas en el archivo *grails-app/conf/UrlMappings.groovy*

- Por defecto, las urls siguen el patrón *[controlador]/[accion]/[id]*

# Ámbitos

- Los controladores disponen de una serie de objetos implícitos
- Estos objetos son mapas
- Coinciden con los ámbitos que se pueden encontrar en cualquier aplicación web

© 2008-2011 Depto. Ciencia de la Computación e IA

# Ámbitos

## *servletContext*

Contiene los datos del ámbito de la aplicación. Cualquier dato que almacenemos en este objeto estará disponible globalmente desde cualquier controlador o acción

# Ámbitos

## *session*

Permite asociar un estado a cada usuario, habitualmente mediante el envío de cookies

# Ámbitos

## *request*

Estos valores sólo estarán disponibles durante la ejecución de la solicitud actual

# Ámbitos

## *params*

Contiene todos los parámetros de la petición actual, tanto los de la url como los del formulario. El mapa *params* puede ser modificado en cualquier momento para añadir valores o modificar los existentes

# Ámbitos

## *flash*

Este ámbito es un almacén temporal para atributos que necesitaremos durante la petición actual y la siguiente, y que serán eliminados cuando ambas se hayan procesado. Se suele utilizar para almacenar el código de error en caso de redirecciones.

© 2008-2011 Depto. Ciencia de la Computación e IA

# El método render()

- Si un controlador no incluye ninguna llamada al método *render()*, Grails buscará la vista predeterminada para dicha acción en el archivo *grails-app/views/[controlador]/[accion]*

- Esto sucedía en el método *login()* de la clase *Usuario* donde no se especificaba ninguna llamada al método *render()*

© 2008-2011 Depto. Ciencia de la Computación e IA

# El método render()

- Si una acción devuelve un mapa con el modelo a mostrar, los campos definidos estarán disponibles en la vista GSP como variables locales

- Si no se devuelve nada, la página GSP sólo tendrá acceso a las variables locales definidas en el propio controlador

© 2008-2011 Depto. Ciencia de la Computación e IA

# El método render()

- Se utiliza para enviar respuestas al cliente de varias formas
- Acepta varios parámetros en función de la salida que necesitemos

# El método render()

| Parámetros | Descripción |
|---|---|
| *text* | La respuesta será enviada en texto plano |
| *builder* | Se puede enviar un builder para generar la respuesta |
| *view* | Se indica la vista que queremos procesar para generar la respuesta |
| *template* | Se indica la plantilla que queremos procesar para generar la respuesta |
| *plugin* | Se indica el plugin donde buscar la plantilla, si ésta no pertenere a nuestra aplicación |
| *bean* | Un objeto con los datos para generar la respuesta |

© 2008-2011 Depto. Ciencia de la Computación e IA

# El método render()

| Parámetros | Descripción |
|---|---|
| *var* | El nombre de la variable con la que accederemos al *bean.* Si no se indica este parámetro, se utilizará la variable *it* |
| *model* | Un mapa con el modelo para usar en la vista |
| *collection* | Una colección para procesar una plantilla con cada elemento |
| *contentType* | El tipo mime de la respuesta |
| *encoding* | El juego de caracteres de la respuesta |
| *converter* | Un objeto del tipo *grails.converters.** para generar la respuesta |

© 2008-2011 Depto. Ciencia de la Computación e IA

# El método render()

```
//Enviar texto
render "un texto devuelto"

//especificar el tipo mime y codificación
render (
        text:"<error>Ha habido un error</error>",
        contentType:"text/xml",
        encoding:"UTF-8"
)

//procesar una plantilla con un modelo
render (
        template:'listado',
        model:[lista:Usuario.list()]
)
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# El método render()

```
//o una colección
render (
        template:'listado',
        collection:[u1,u2,u3]
)

//o con un objeto
render (
        template:'listado',
        bean:Usuario.get(2),
        var:'u'
)
```

# El método render()

```
//procesar una vista con un modelo
render (
        view:'listado',
        model:[lista:Usuario.list()]
)

//o con el propio controlador como modelo
render (view:'usuario')
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# El método render()

```
//con un builder
render {
        div(id:'miDiv','Contenido del div')
}

render (contentType:'text/xml'){
        listado {
                Usuario.list().each {
                        usuario(
                                nombre:it.nombre,
                                apellidos:it.apellidos
                        )
                }
        }
}
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# El método render()

```
//generando JSON
def u = Usuario.get(1)
render (contentType:'text/json') {
            usuario(nombre:u.nombre, apellidos:u.apellidos)
}

//generar XML y JSON automáticamente
import grails.converters.*

render Usuario.list(params) as XML
render Usuario.get(params.id) as JSON
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

- Hasta ahora hemos utilizado el *scaffolding dinámico*
- Grails permite también un *scaffolding estático*, donde el código se genere de forma offline
- Necesitamos generar tanto las vistas como los controladores para el scaffolding estático

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

```
grails generate-views biblioteca.Usuario
```

- Se utilizan unas plantillas para crear las vistas de las cuatro operaciones básicas
- Se crean en el directorio *grails-app/views/usuario* los archivos *create.gsp, edit.gsp, list.gsp y show.gsp*

# Mejorando los controladores

```
grails generate-controller biblioteca.Usuario
```

- Antes de ejecutar este comando, realizar una copia de seguridad del controlador para guardar los métodos *login(), handleLogin()* y *logout()*
- El comando crea un nuevo controlador con los métodos *index(), list(), show(), delete(), edit(), update(), create()* y *save()*
- Debemos copiar también los métodos *login(), handleLogin()* y *logout()*

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

- Necesitamos controlar que un usuario sólo pueda editar sus propios datos

```
def edit = {
   def usuarioInstance = Usuario.get( params.id )
      if(!usuarioInstance) {
         flash.message = "${message(code: 'default.not.found.message', args:
               [message(code: 'usuario.label', default: 'Usuario'), params.id])}"
         redirect(action:list)
      }
      else {
         return [ usuarioInstance : usuarioInstance ]
      }
}
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

```
def edit = {
  if (session?.usuario?.id as String != params.id){
    flash.message = "Sólo puedes editar tu información"
    redirect(action:list)
    return
  }
  def usuarioInstance = Usuario.get( params.id )
  if(!usuarioInstance) {
    flash.message = "${message(code: 'default.not.found.message',
      args: [message(code: 'usuario.label', default: 'Usuario'), params.id])
    redirect(action:list)
  }
  else { return [ usuarioInstance : usuarioInstance ]  }
}
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

- Sólo los administradores pueden crear usuarios

```
def create = {
    if (session?.usuario?.tipo != "administrador"){
      flash.message = "Sólo los administradores pueden crear usuarios"
      redirect(action:list)
      return
    }

    def usuarioInstance = new Usuario()
    usuarioInstance.properties = params
    return ['usuarioInstance':usuarioInstance]
}
```

# Mejorando los controladores

- Los administradores también pueden modificar los datos de los usuarios
- Debemos modificar la función *edit()*, *update()* y *delete()*

# Mejorando los controladores

```
def edit = {
  if ((session?.usuario?.id as String != params.id) &&
    (session?.usuario?.tipo != "administrador")){
      flash.message = "Sólo puedes editar tu información"
      redirect(action:list)
      return
  }
  def usuarioInstance = Usuario.get( params.id )
  if(!usuarioInstance) {
    flash.message = "${message(code: 'default.not.found.message',
      args: [message(code: 'usuario.label', default: 'Usuario'), params.id])}"
    redirect(action:list)
  }
  else { return [ usuarioInstance : usuarioInstance ] }
}
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

- La variable *params* es un mapa que recoge los valores pasados en la petición
- Podemos utilizar el operador punto (.id) o bien el operador corchete [‘id’]

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

- Componentes de la dirección url http://localhost:8080/biblioteca/usuario/show/1
  - *dominio:* http://localhost
  - *puerto: 8080*
  - *Nombre de la aplicación:* biblioteca
  - *Nombre del controlador:* usuario
  - *Nombre de la acción ejecutada:* show
  - *Parámetro ID:* 1

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

- Cuando se utiliza la sentencia *return [usuarioInstance: usuarioInstance]* se está devolviendo un mapa de valores

```
<tr class="prop">
        <td valign="top" class="name">
                <g:message code="usuario.login.label" default="Login" />
        </td>
        <td valign="top" class="value">
                ${fieldValue(bean:usuarioInstance, field:'login')}
        </td>
</tr>
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Mejorando los controladores

- La función *fieldValue()* también puede ser utilizada en las páginas GSP

```
<g:fieldValue bean="${usuarioInstance}" field="login" />
```

# Mejorando los controladores

- El método *save()* se encarga de persistir los valores introducidos en la base de datos

```
def save = {
  def usuarioInstance = new Usuario(params)
  if(!usuarioInstance.hasErrors() && usuarioInstance.save()) {
    flash.message = "${message(code: 'default.created.message',
      args::[message(code: 'usuario.label', default: 'Usuario'), usuarioInstance.id])}"
    redirect(action:show,id:usuarioInstance.id)
  }
  else {
    render(view:'create', model:[usuarioInstance:usuarioInstance])
  }
}
```

# Mejorando los controladores

- Los usuarios que no sean *administradores* no podrán modificar el tipo de usuario

```
<g:if test="${session?.usuario?.tipo == 'administrador' }">
  <tr class="prop"><td valign="top" class="name">
    <label for="tipo"><g:message code="usuario.tipo.label" default="Tipo" /><
/label>
    </td><td valign="top" class="value ${hasErrors(bean:usuarioInstance,
      field:'tipo', 'errors')}">
      <g:select id="tipo" name="tipo"
          from="${usuarioInstance.constraints.tipo.inList}"  value="
${usuarioInstance.tipo}" >
      </g:select>
    </td> </tr>
</g:if>
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

- Son métodos que se ejecutan antes y después de cualquier acción de un controlador
- Se suelen utilizar para la generación de archivos de log con todas las acciones ejecutadas junto con los datos de entrada
- Los métodos en cuestión son *beforeInterceptor()* y *afterInterceptor()*

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

- Vamos a utilizarlos para la generación de un archivo de logs con las acciones ejecutadas
- Añadimos el método *beforeInterceptor()* al controlador de la clase *Usuario*

```
def beforeInterceptor = {
        log.trace("${session?.usuario?.login}
                Empieza la acción ${controllerName}
                Controlador.${actionName}() : parámetros $params")
}
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

- Añadimos el método *afterInterceptor()* al controlador de la clase *Usuario*
- Se le debe pasar también el modelo de datos para que pueda imprimirlos en el log

```
def afterInterceptor = { model ->
        log.trace("${session?.usuario?.login}
                Termina la acción ${controllerName}
                Controlador.${actionName}() : devuelve $model")
}
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

- Es posible indicarle sólo algunos métodos

```
def beforeInterceptor = [action:this.&beforeAudit,except:['list']]
def afterInterceptor = [action:{model ->this.&afterAudit(model)}, except:['list']]
def beforeAudit = {
        log.trace("${session?.usuario?.login}
                Empieza la acción ${controllerName}
                Controlador.${actionName}() : parámetros $params")
}
def afterAudit = { model ->
        log.trace("${session?.usuario?.login}
                Termina la acción ${controllerName}
                Controlador.${actionName}() : devuelve $model")
}
```

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

- Necesitamos configurar *log4j*, una librería de Apache utilizada en este tipo de tareas
- Con esta librería se acabó utilizar *println()* a lo largo de nuestra aplicación
- Permite establecer niveles de granularidad en los mensajes emitidos

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

| Nivel | Descripción |
|---|---|
| *fatal* | Se utiliza para mensajes críticos del sistema, generalmente después de guardar el mensaje, el programa abortará |
| *error* | Se utiliza en mensajes de error de la aplicación que se desea guardar. Estos eventos afectan al programa pero lo dejan seguir funcionando. |
| *warn* | Se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero no afectan al funcionamiento. |
| *info* | Se utiliza para mensajes similares al modo *verbose* de otras aplicaciones |
| *debug* | Se utiliza para escribir mensajes de depuración. No debe estar activado en aplicaciones en producción |
| *trace* | Se utiliza para mostrar mensjes con más detalle que *debug* |

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

- *Log4j* puede enviar mensajes a una o varias salidas de destino con los llamados *appenders*
- Existen varios appenders disponibles y configurados
- Se pueden crear *appenders* personalizados

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

- Los típicos *appenders* son aquellos que redirigen la salida a un fichero de texto (*FileAppender, RollingFileAppender)*
- También es posible almacenar estos registros en un servidor remoto con *SocketAppender*
- O a una dirección de correo electrónico con *SMTPAppender*
- O incluso una base de datos con *JDBCAppender*

© 2008-2011 Depto. Ciencia de la Computación e IA

# Interceptadores de acciones

- Nosotros generaremos un archivo de log llamado *mylog.log*
- Debemos configurar el archivo *grails-app/conf/Config.groovy*
- Añadiremos un *appender* de tipo *file*

# Interceptadores de acciones

```
// log4j configuration
log4j = {
        appenders {
                file name:'file', file:'mylog.log'
        }
        root {
                info 'file'
                additivity = true
        }
        trace "grails.app.controller.UsuarioController"
        ....
}
```

# Interceptadores de acciones

```
// log4j configuration
log4j = {
            ....
    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'
     warn 'org.mortbay.log'
}
```

# Interceptadores de acciones

- Especificamos que el nivel de granularidad será *trace*
- Las acciones ejecutadas se añaden al archivo *mylog.log*

© 2008-2011 Depto. Ciencia de la Computación e IA

# Filtros

- Similares a los *interceptadores de acciones*
- Se ejecutan antes y después de una acción
- Pueden ser utilizados en múltiples controladores

# Filtros

- Crearemos un filtro para centralizar la comprobación de que un usuario sólo pueda editar sus propios datos
- Los filtros se deben crear en el directorio *grails-app/conf*
- Deben terminar por la palabra *Filters* como por ejemplo *UsuarioFilters.groovy*

© 2008-2011 Depto. Ciencia de la Computación e IA

# Filtros

```
package biblioteca

class UsuarioFilters {
        def filters = {
                chequeoModificacionUsuario(controller: 'usuario', action: '*') {
                        ...
                }
                otroFiltro(uri: '/usuario/*') { }
        }
}
```

# Filtros

- En el método *chequeoModificacionUsuario()* se ejecutarán todas las acciones gracias al *
- En el método *otroFiltro()* se declaran el ámbito de las acciones especificando la *uri,* que en este caso también son todas las acciones
- Debemos especificar también cuando debemos ejecutar estos filtros (*before, after* o *afterView)*

© 2008-2011 Depto. Ciencia de la Computación e IA

# Filtros

```
package biblioteca
class UsuarioFilters {
    def filters = { chequeoModificacionUsuario(controller: 'usuario', action: '*') {
            before = {
                if (actionName == 'edit' || actionName == 'update' ||
                actionName == 'delete') {
                    if (session?.usuario?.id as String != params?.id) {
                        flash.message = "Sólo puedes editar tu información"
                        redirect(controller:'usuario', action: 'list')
                        return false
                    }
                }
            }
        }
    }
}
```

# Filtros

```
def filters = {
        chequeoModificacionUsuario(controller: 'usuario', action: '*') {
            before = {
                if (actionName == 'edit' || actionName == 'update' ||
                actionName == 'delete') {
                    if ((session?.usuario?.id as String != params?.id) &&
                    (session?.usuario?.tipo != “administrador”)) {
                        flash.message = "Sólo puedes editar tu información"
                        redirect(controller:'usuario', action: 'list')
                        return false
                    }
                }
            }
        }
}
```

# Filtros

- Con los filtros, se centraliza la lógica que gestiona las comprobación de usuario en un sólo fichero y una sola función

- Añadiremos otro filtro para que sólo los administradores puedan crear usuarios

# Filtros

```
chequeoCreacionUsuario(controller: 'usuario', action:'*'{
        before = {
            if (actionName == 'create' || actionName == 'save'){
                if (session?.usuario?.tipo != "administrador") {
                    flash.message = "Sólo los administradores pueden crear usuarios"
                    redirect(controller:'usuario', action: 'list')
                    return false
                }
            }
        }
}
```