Eusko Jaurlaritzaren Informatika Elkartea
Sociedad Informática del Gobierno Vasco

UDA - Utilidades de desarrollo de aplicaciones

Gestión de excepciones

Fecha: 13/03/2012 | **Referencia:**

EJIE S.A.
Mediterráneo, 14
Tel. 945 01 73 00*
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01080 Vitoria-Gasteiz
www.ejie.es

Control de documentación

Título de documento: Gestión de properties e internacionalización (i18n)

Histórico de versiones

Código:	Versión:	Fecha:	Resumen de cambios:
	1.0.0	13/03/2012	Primera versión.
	1.0.1	11/02/2012	Añadida gestión de excepciones de Acceso denegado y Validaciones

Cambios producidos desde la última versión

Control de difusión

Responsable: Ander Martínez

Aprobado por:

Firma: Fecha:

Distribución:

Referencias de archivo

Autor:

Nombre archivo:

Localización:

Contenido

1 Introducción

El presente documento pretende explicar la manera en la que se ha implementado la nueva gestión de excepciones y su funcionamiento básico.

Las aplicaciones desarrolladas con UDA contaban con una serie de excepciones por defecto dentro de la librería x38. Mediante el uso de estas clases se conseguía una gestión homogénea en todas las aplicaciones, pero era labor del desarrolador la gestión de cada excepción de la aplicación.

La nueva gestión independiza el código de las clases (ya sean controllers, services o daos) de las excepciones ya que esta se gestionan a través de una clase externa denominada *handler* (en caso de necesidad también se puede realizar la gestión en el propio controller). UDA por defecto aporta un handler genérico que realiza la gestión por defecto de las excepciones pero será labor del desarrollador decidir si quiere realizar una gestión particular de cierto tipo de excepciones.

A continuación se detalla la arquitectura que componen las nuevas clases para gestión de excepciones, su configuración y un ejemplo de cómo implementar un *handler*.

2 Arquitectura

En el presence capítulo se van a detallar las clases que se incluían en la librería x38 para la gestión de excepciones y las nuevas clases.

2.1 *Clases obsoletas (deprecated)*

La siguiente lista contiene las clases de la librería x38 que se han quedado obsoletas con la nueva gestión y por tanto se han definido como *deprecated*:

- com.ejie.x38.control.ExceptionResolver
- com.ejie.x38.control.exception.ControlException
- com.ejie.x38.control.exception.MethodFailureException
- com.ejie.x38.control.exception.ResourceNotFoundException
- com.ejie.x38.control.exception.ServiceUnavailableException

2.2 *Nuevas clases*

A continuación se detallan las nuevas clases incluidas en x38 para gestionar las excepciones:

- com.ejie.x38.control.exception.handler.MvcExceptionHandler:
  - Clase de UDA que se encarga de la gestión por defecto de las excepciones.
- com.ejie.x38.control.exception.handler.MvcAccessDeniedExceptionHandler:
  - Clase de UDA que se encarga de la gestión de las excecpciones de acceso denegado.
- com.ejie.x38.control.exception.handler.MvcValidationExceptionHandler:
  - Clase de UDA que se encarga de la gestión de las excepciones de validación.
- com.ejie.x38.control.exception.MvcExceptionResolver:
  - Clase que se encarga de decidir qué *handler* se va a encargar de gestionar la excepción producida.
- com.ejie.x38.control.exception.MvcExceptionResolverConfig:
  - Clase que se encarga de configurar la gestión de excepciones en tiempo de despliegue

3 Configuración

La gestión de excepciones se define a nivel de WAR. Por tanto cada aplicación deberá realizar tantas configuraciones como módulos web tenga.

El código necesario para configurar las clases de x38 se define en el fichero mvc-config.xml:

```
<!-- Exception -->
<bean class="com.ejie.x38.control.exception.MvcExceptionResolverConfig"/>
```

Con esta simple línea se configura la gestión por defecto de UDA y se utiizará su handler para la resolución de las excepciones generadas. También pueden deshabilitarse los handlers por defecto de UDA para las excepciones producidas en gestión de acceso y validaciones (por defecto están activados value="false"). En el caso de querer definir handlers propios bastará con añadir una lista de clases bajo la propiedad "handlers":

```
<!-- Exception -->
<bean class="com.ejie.x38.control.exception.MvcExceptionResolverConfig">
  <property name="disabled_accessDenied" value="true" />
  <property name="disable_validation" value="true" />
  <property name="handlers">
    <list>
        <bean class="com.ejie.xxx.exception.MyExceptionHandler" />
    </list>
  </property>
</bean>
```

Es muy importante el orden en el que se definen los handlers ya que será el orden en el que se irá comprobando si dicho handler es capaz de resolver la excepción generada.

3.1 Requisitos

La configuración del bean MvcExceptionConfig se realiza en la fase de despliegue y en caso de no ser capaz de configurarse se producirá un error indicando en las trazas el origen del problema.

Existen dos requisitos por los que podría ser posible un error a la hora de configurar el gestor de excepciones:

- AnnotationMethodHandlerAdapter:

  Se debe definir un bean de este tipo que definirá los *messageConverters* requeridos para las diferentes transformaciones de los datos. UDA (por defecto) define al menos el convertidor de Jackson propio denominado "udaMappingJacksonHttpMessageConverter".

  El gestor de excepciones (*MvcExceptionResolver*) internamente contiene una lista de *messageConverters* que son necesarios a la hora de transformar la respuesta cuando se produce una excepcion, por lo que se deberán configurar los mismos definidos en la aplicación. Para ello, cuando se instancie el bean *MvcExceptionConfig* buscará en el contexto de Spring un bean de este tipo para obtener los messageConverters y asignarlos al gestor de excepciones.

```
<bean class="org.springframework.web.servlet.mvc.annotation.
      AnnotationMethodHandlerAdapter">
      ...
      <property name="messageConverters">
            <list>
                  <ref bean="udaMappingJacksonHttpMessageConverter"/>
            </list>
      </property>
</bean>
```

- messageSource (ReloadableResourceBundleMessageSource):

  El handler por defecto de UDA (MvcExceptionHandler) utiliza los ficheros idiomáticos (i18n) para detallar los mensajes de respuesta en caso de excepción. Buscará en dichos ficheros claves cuyo valor sea el nombre simple (sin paquete) de la excepción generada por lo que es necesario que cuando se cargue desde el MvcExceptionResolverConfig reciba una referencia al bean cuyo identificador será messageSource.

```
<bean id="messageSource" class="org.springframework.context.support.
      ReloadableResourceBundleMessageSource">
      ...
</bean>
```

4 Funcionamiento

El código por defecto generado por UDA captura las excepciones a nivel de controlador aunque estas se hayan generado en la capa de acceso a datos, en la de servicios o en el propio controlador (pueden capturarse en cualquier capa mediante try/catch pero se centraliza su gestión en el controlador). Para ello se define un método con la anotación @ExceptionHandler con lo que dicho método captura cualquier excepción producida. Ejemplo:

```
@ExceptionHandler
public @ResponseBody String handle (Exception e) {
	...
}
```

La anotación @ExceptionHandler permite definir qué excepción se captura (si no se especifica ninguna en concreto capturará todas):

```
@ExceptionHandler(value=DuplicateKeyException.class)
public @ResponseBody String handleDuplicateKey (DuplicateKeyException e) {
	...
}
```

Sin la incorporación del nuevo sistema de gestión de excepciones, existía la limitación de que una excepción producida en un cotrolador (o en las clases a las que llama: servicio, dao) debía ser gestionada por el propio controlador. Por tanto el método de gestión de una excepción debía duplicarse en cada controlador que pudiera producir dicha excepción.

La nueva gestión de excepciones define una jerarquía de clases en las que buscar el método encargado de tratar la excepción. El orden resolución de una excepción sería el siguiente:

- Método donde se genera la excepción (try/catch)
- Controlador
- Handler de excepción de Acceso denegado (propio de UDA) si no se deshabilita
- Handler de excepción de Validación (propio de UDA) si no se deshabilita
- Lista de handlers definidos en mvc-config.xml (MvcExceptionResolverConfig)
  - Primer handler definido
  - Segundo handler definido
  - …
- MvcExceptionHandler (handler por defecto de UDA)

Con la inclusión de esta jerarquía de resolución de excepciones, los nuevos controladores generados por UDA ya no no incluirán ningún try/catch ya que la gestión por defecto la definirá la clase *MvcExceptionHandler*. De este modo se simplifica el código generado y su comprensión.

A continuación se comentan brevemente el funcionamiento de las nuevas clases incorporadas en x38.

4.1 MvcExceptionResolverConfig

La clase *MvcExceptionResolverConfig* se instancia desde el fichero de configuración del WAR (mvc-config.xml) y realiza las siguientes funciones:

- Comprobar los requsitos necesarios en fase de despliegue.
- Configurar la gestión de excepciones

- Registrar los *handlers* definidos por el desarrollador en el fichero de configuración.
- Registrar el *handler* por defecto de UDA

4.2 MvcExceptionResolver

La clase *MvcExceptionResolver* es la encargada de decidir qué clase se encarga de la gestión de una excepción. Cuando se produce una excepción Spring invoca el método *getExceptionHandlerMethod(…)* de esta clase que irá de manera escalonada comprobando que clase contiene el método requerido para resolver la excepción.

En primera instancia se comprueba si la clase donde se ha producido la excepción (controller) contiene un método que pueda resolver la excepción. En caso de ser así, se invocará dicho método (al igual que sin la nueva gestión de excepciones) y si no se continuará buscando.

En segundo lugar se comprueban los handlers definidos por el desarrollador, en orden de declaración en el fichero mvc-config.xml, hasta encontrar un método válido.

En última instancia y si no se ha encontrado ningún método apto, se tratará la excepción mediante la clase *MvcExceptionHandler*.

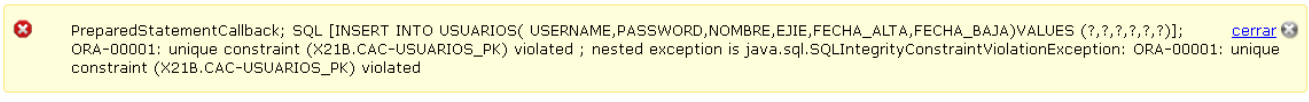
Un detalle importante a tener en cuenta es que si en alguno de las clases candidatas a resolver la excepción (controller o handler propio) se declara un método con la anotación @ExceptionHandler sin definir qué excepción se captura, dicho método se encargará de resolver todas las excepciones que le lleguen sin posibilidad alguna de que la excepción llegue a un habdler posterior.

4.3 MvcExceptionHandler

La clase *MvcExceptionHandler* es la clase que incluye UDA por defecto para la resolución de excepciones en el caso de que el desarrollador no defina ningún otro handler. Contiene un único método anotado con @ExceptionHandler por lo que captura cualquier tipo de excepción. Tiene un tratamiento diferenciado para las peticiones AJAX de las peticiones comunes:

- Peticiones AJAX: devolverán el código de respuesta (HttpStatusCode 406 NOT ACCEPTABLE) y el nombre simple (sin paquete) de la clase que ha producido el error. Por ejemplo en el caso de que se produzca una excepción de tipo “org.springframework.dao.DuplicateKeyException” se devolverá el mensaje de la excepción. De cara a facilitar la personalización del mensaje de error, se ha decidido que se busque en los ficheros de recursos idiomáticos (i18n) una clave cuyo valor sea el nombre de la excepción.

  Ejemplo sin definir la propiedad:

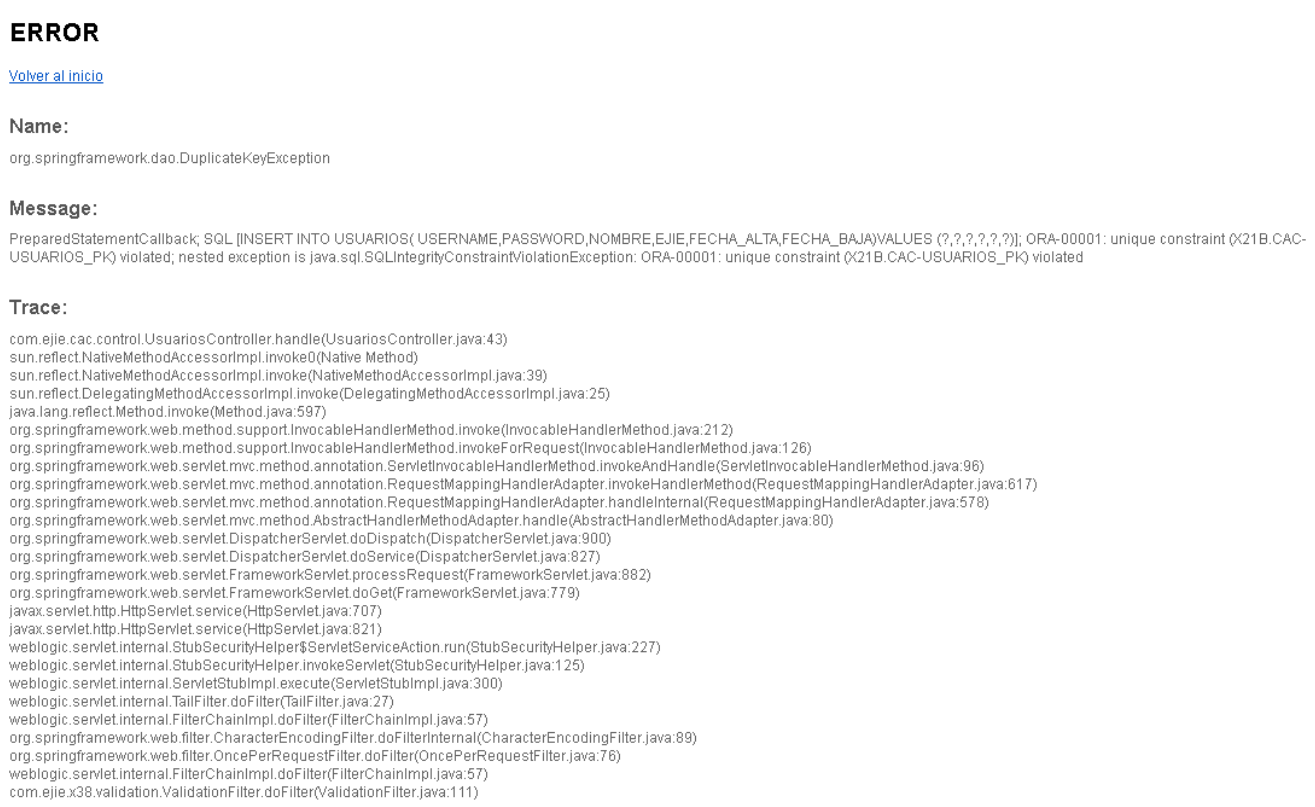
  

  Ejemplo definiendo la propiedad:

```
[xxxYYYWar\WebContent\WEB-INF\resources\xxxYYYWar.18n_es.properties]
DuplicateKeyException = Clave primaria duplicada
```

- Peticiones NO AJAX: redirigen a la vista “error” (en UDA por defecto la página error.jsp) con el nombre de la excepción, su descripción y el *stacktrace* en el modelo.



ERROR

Volver al inicio

Name:

org.springframework.dao.DuplicateKeyException

Message:

PreparedStatementCallback; SQL [INSERT INTO USUARIOS(USERNAME,PASSWORD,NOMBRE,EJIE,FECHA_ALTA,FECHA_BAJA)VALUES (?,?,?,?,?,?)]; ORA-00001: unique constraint (X21B.CAC-USUARIOS_PK) violated; nested exception is java.sql.SQLIntegrityConstraintViolationException: ORA-00001: unique constraint (X21B.CAC-USUARIOS_PK) violated

Trace:

com.ejie.cac.control.UsuariosController.handle(UsuariosController.java:43)
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
java.lang.reflect.Method.invoke(Method.java:597)
org.springframework.web.method.support.InvocableHandlerMethod.invoke(InvocableHandlerMethod.java:212)
org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:126)
org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:96)
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:617)
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:578)
org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:80)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:900)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:827)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:882)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:779)
javax.servlet.http.HttpServlet.service(HttpServlet.java:707)
javax.servlet.http.HttpServlet.service(HttpServlet.java:821)
weblogic.servlet.internal.StubSecurityHelper$ServletServiceAction.run(StubSecurityHelper.java:227)
weblogic.servlet.internal.StubSecurityHelper.invokeServlet(StubSecurityHelper.java:125)
weblogic.servlet.internal.ServletStubImpl.execute(ServletStubImpl.java:300)
weblogic.servlet.internal.TailFilter.doFilter(TailFilter.java:27)
weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:57)
org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterEncodingFilter.java:89)
org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:76)
weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:57)
com.ejie.x38.validation.ValidationFilter.doFilter(ValidationFilter.java:111)

4.4 MvcAccessDeniedExceptionHandler

La clase *MvcAccessDeniedExceptionHandler* es la clase que incluye UDA por defecto para la resolución de excepciones en el caso de que no se tenga acceso. Contiene un único método anotado con @ExceptionHandler que captura las excepciones de tipo “AccessDeniedException” Tiene un tratamiento diferenciado para las peticiones AJAX de las peticiones comunes:

- Peticiones AJAX: igual que el *MvcExceptionHandler* salvo que el código de error devuelto será un HttpStatusCode 401 UNAUTHORIZED
- Peticiones NO AJAX: igual que el *MvcExceptionHandler*.

4.5 MvcValidationExceptionHandler

La clase *MvcValidationExceptionHandler* es la clase que incluye UDA por defecto para la resolución de excepciones en el caso de que salte una validación con la excepción “MethodArgumentNotValidException” o “BindException” Tiene un tratamiento diferenciado para las peticiones AJAX de las peticiones comunes:

- Peticiones AJAX: igual que el *MvcExceptionHandler*.
- Peticiones NO AJAX: igual que el *MvcExceptionHandler*.

5 *Handler* propio

UDA provee al desarrollador de un sistema de gestión de excepciones completamente modular. El handler por defecto de UDA (*MvcExceptionHandler*) tiene como objetivo proveer de funcionalidad básica en el tratamiento de excepciones, es decir, las excepciones son capturadas y mostradas en la pantalla de error. **Será labor del desarrollador implementar un *handler* propio y crear una pantalla de error corporativa acorde con los estilos de la aplicación.**

Para implementar un handler, bastará con definirse una clase Java con diferentes métodos anotados con @ExceptionResolver indicando las excepciones que se desean capturar con dicho método.

Las características de los métodos del *handler* son las siguientes:

- El nombre del método es arbitrario ya que no se utiliza. Simplemente cada método deberá tener un nombre diferente para que la clase pueda compilarse.
- Un mismo método puede capturar una o más excepciones:
  - Capturar una excepción:
    ```
    @ExceptionHandler(value=XXXException.class)
    ```
  - Capturar varias excepciones:
    ```
    @ExceptionHandler({YYYException.class, ZZZException.class})
    ```
- Se puede anotar el método con `@ResponseStatus` para indicar el código HTTP de la respuesta (o definirlo mediante la *response*).
- El método recibirá al menos un parámetro, la excepción. Si es necesario se pueden añadir como parámetros la *request (*`HttpServletRequest` *)* y la *response (*`HttpServletResponse`*).*
- El método podrá propagar una excepción en caso de necesidad añadiéndole la cláusula **`throws`**. En este caso la excepción será capturada por el filtro de UDA.
- El tipo de retorno del método es variable y dependerá de la gestión realizada:
  - void: Cuando la respuesta se mande a través de la *response*.
    ```
    //Ejemplo devolución Mensaje
    response.sendError(406, message);

    //Ejemplo devolución Objeto JSON
    String json = ...;
    PrintWriter out = response.getWriter();
    out.write(json);
    ```
  - ModelandView: Cuando se quiere redirigir a una vista con datos en el modelo.
    ```
    ModelAndView modelAndView = new ModelAndView("viewName");
    modelAndView.addObject("obj1", "value1");
    ...
    return modelAndView;
    ```
- Se podrán inyectar los objetos necesarios por ejemplo messageSource para la resolución de literales (i18n):
  ```
  @Autowired
  private ReloadableResourceBundleMessageSource messageSource;
  ```

Ejemplo de *handler*:

```
public class MyExceptionHandler {

	@ExceptionHandler(value=DateIntervalException.class)
	@ResponseStatus(value=HttpStatus.NOT_ACCEPTABLE)
	public void handleDateInterval (DateIntervalException die) {
		...
	}

	@ExceptionHandler(value=DuplicateKeyException.class)
	@ResponseStatus(value=HttpStatus.NOT_ACCEPTABLE)
	public ModelandView handleDuplicateKey (DuplicateKeyException dke) {
		...
	}

	...
}
```

6 Manual de migración

El capítulo que nos ocupa pretende recopilar los cambios a realizar en una aplicación para utilizar la nueva gestión de idioma. A continuación se detalla proyecto a proyecto los cambios necesarios (en el siguiente capítulo se detallará las mejoras aportadas con dichos cambios):

- <xxxNombreWAR>
  - Modificar en el fichero **mvc-config.xml**, añadiendo un identificativo al bean que resuelve las peticiones:

```
<bean id="requestMappingHandlerAdapter"
class="org.springframework.web.servlet.mvc.annotation.
AnnotationMethodHandlerAdapter">
```

  - Configurar las excepciones en el fichero **mvc-config.xml**, añadiendo el bean que configura la resolución de las mismas:

```
<bean
class="com.ejie.x38.control.exception.MvcExceptionResolverConfig" />
```

  - Modificar los métodos de los Controllers para que las excepciones lleguen hasta el handler propio o aportado por la x38 (eliminando los bloques try/match):

```
@RequestMapping(method = RequestMethod.POST)
public @ResponseBody MyBean add(@RequestBody MyBean myBean,
      HttpServletResponse response) {
      try {
            ...
            MyBean myBeanAux= this.myBeanService.add(MyBean);
            logger.info("Entity correctly inserted!");
            return myBean;
      } catch (Exception e) {
            if (e instanceof DuplicateKeyException){
                  throw new MethodFailureException("Method failed");
            }
            throw new MethodFailureException("Method failed");
      }
}
```

  - Modificar en el fichero **rup.base-x.x.x.jsp** añadiendo las siguientes líneas destacadas en negrita al comienzo del fichero en la declaración del método *rup_ajax* :

```
      }else if(xhr.status==503){
            ...

      }
            //Excepción
            else if (xhr.status==406){ //Código de error de UDA
                  var obj="";
                  try{
                        obj = JSON.parse(xhr.responseText);
                  } catch (error){
                        errorText=xhr.responseText;
                  }
            }
```

```
}
```

- Modificar en el fichero **rup.maint-x.x.x.jsp** añadiendo las siguientes líneas destacadas en negrita en el método *saveMaint* (línea 916 apróx.):

```
if ((mant.prop.jQueryGrid.rup_grid("isEditable") &&
!$.rup_utils.compareObjects(init,dt))
                                        || (
!mant.prop.jQueryGrid.rup_grid("isEditable") &&
(mant.prop.detailForm && mant.prop.detailForm.serialize() !==
mant.prop.detailForm.data('initialData')))) {
      dt = $.toJSON(dt);
      var errores = false;

      $.ajaxUDA=$.ajax;
      $.ajax=$.rup_ajax;

      mant.prop.detailForm.rup_form("ajaxFormSubmit",{
            ...

      });

      $.ajax=$.ajaxUDA;
      return errores;
}
```

- [Opcional] Modificar la página error.jsp para que cuando se produzca un fallo no controlado se muestre información que facilite la localización del error para su corrección (solo en desarrollo, en producción se debería definir una página de error propia):

```
<%@include file="/WEB-INF/includeTemplate.inc"%>

<h2>ERROR</h2> <br>
<a href="<%= request.getContextPath() %>/"><spring:message
code="error.volver" /></a>

<h3>Name: </h3>${(empty param)? exception_name :
param.exception_name}<br>

<h3>Message: </h3>${(empty param)? exception_message :
param.exception_message}<br>

<h3>Trace: </h3>${(empty param)? exception_trace :
param.exception_trace}<br>
```

7 Nuevas funcionalidades

La nueva gestión de excepciones de UDA proporciona una serie de nuevas funcionalidades que se comentan a continuación:

- Página de error *amigable*:
  - Muestra la excepción que se ha producido, su causa y la traza para seguir la ejecución.
  - Añadido un enlace para volver al inicio de la aplicación
  - NOTA: Esta página es para facilitar la labor del desarrollador, se debería implementar una página de error propia que se ajuste a las necesidades del negocio.
- Control de excepciones por defecto:
  - En el caso de que se produzca una excepción no recogida por el desarrollador, la clase interna de x38 llamada MvcExceptionHandler se encargará de recogerla y tratarla debidamente:
    - Redireccionando a la página de error en problemas con validaciones y excepciones.
    - Mostrando la excepción en el feedback de la página en el caso de peticiones AJAX.
- Modularidad y evitar código duplicado:
  - Al poder definir una clase externa al controlador para la gestión de cierto tipo de excepción se puede reutilizar en diferentes puntos de la aplicación sin necesidad de duplicar código.
  - Pueden definirse diferentes clases según las necesidades en cada una de las partes del negocio.