RESTFul CARM

Control de versiones del documento

Versión	Fecha	Autor	Motivo del cambio
1.0	14/05/2015	Antonio Morcillo Martínez	Creación del documento
1.1	18/06/2015	Antonio Morcillo Martínez	Información sobre el pool de conexiones
1.2	05/08/2015	Antonio Morcillo Martínez	Extensión para consultas parametrizadas

Figuras

RestFulCARM

El objetivo de este componente es ofrecer una capa de servicios web *REST* que, como resultado de su invocación, ejecuten determinadas sentencias *SQL* sobre alguna de las *BBDD* configuradas y que como resultado de esa ejecución devuelvan a la aplicación llamante el resultado de esa ejecución en formato *JSON* o *XML*.

Diagrama de clases

A continuación se presenta el modelo de clases que gestiona la solicitud de las peticiones *HTTP* y el acceso a base de datos

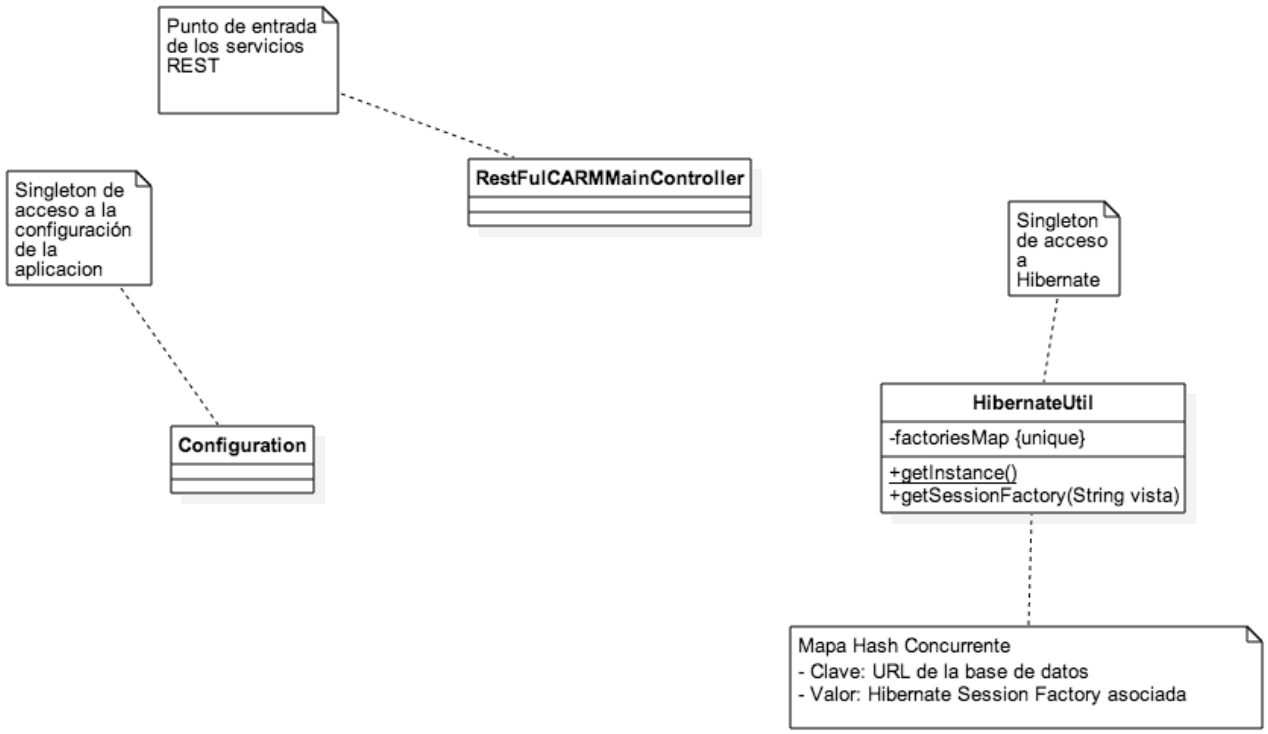

Figura 1. Diagrama de clases

Conexiones a las Bases de Datos

La aplicación realiza las conexiones a las distintas bases de datos configuradas mediante *pools* de conexiones *C3P0* creados directamente por la aplicación bajo demanda.

Para más información sobre *C3P0* (http://www.mchange.com/projects/c3p0/*)*

La principal ventaja de utilizar el esquema de *pools* en lugar de crear conexiones únicas en cada petición recibida es la ganancia de velocidad al tener un conjunto de conexiones abiertas y disponibles a *BBDD* para ser utilizadas inmediatamente por las peticiones *HTTP* que llegan desde los clientes de *RestFulCARM*, inicialmente, los servidores *Liferay*.

Pools dinámicos

RESTFul CARM tiene como requisito fundamental la flexibilidad de configuración para poder añadir conexiones y sentencias *SQL* a distintas *BBDD* de forma dinámica modificando únicamente la configuración de la aplicación.
Este requisito hace que no se puedan definir a nivel de servidor de aplicación un catálogo de *DataSources* estáticos que sea accedido por las aplicaciones mediante *JNDI* o de forma programática.
Por tanto la configuración de la aplicación contendrá:

- *URL* de la base de datos
- Usuario de base de datos
- Contraseña de base datos
- Sentencias *SQL* a ejecutar sobre la conexión definida por los tres campos anteriores

La aplicación se responsabiliza de crear un *pool* de conexiones a cada una de las distintas *URLs* de *BBDD* configuradas **la primera vez que recibe una petición a una *URL*.**

La creación del *pool* implica los siguientes pasos:

- Creación del *pool*
- Asociación del *pool* de conexiones a una *URL* concreta de *BBDD*. Esta asociación se almacena en un *ConcurrentHashMap* del *JDK 7,* con lo que el control y orden ante posibles problemas de concurrencia y creación de *pools* debe estar garantizado
- Creación del número mínimo de conexiones a *BBDD* definido en el *pool (Ver apartado "Configuración básica de los pool")*
- Se hace disponible el *pool* a *Hibernate*

Por la propia definición de *pool* las conexiones sólo se liberarán al parar el servidor de aplicaciones.

Configuración Básica de los pool

Como ya se ha comentado anteriormente la aplicación es la responsable de crear los *pools* de conexiones y ponerlos a disposición de *Hibernate.*

Los parámetros básicos de configuración de los *pools* están definidos en el fichero de configuración de la aplicación *hibernate.cfg.xml* y son los siguientes:

- *c3p0.acquire_increment:* Especifica el número de conexiones que el pool va a intentar crear cuando todas las conexiones están ocupadas.
- *c3p0.idle_test_period:* Indica cada cuantos **segundos** el *pool* va a probar las conexiones ociosas. Equivale a mandar un *keep-alive* a la *BBDD*
- *c3p0.min_size:* El número mínimo de conexiones del *pool*
- *c3p0.max_size:* El número máximo de conexiones del *pool*

- *c3p0.max_statements:* El número máximo de *statements* que va a mantener el *pool* cacheados
- *c3p0.timeout:* El número de segundos que una conexión puede estar abierta sin ser usada antes de ser descartada

Extensión de RESTFul CARM para paginación de resultados

En ocasiones, debido a la gran cantidad de datos devueltos por las llamadas a los servicios nos interesa que se devuelvan los datos en bloques de N elementos en lugar de todo el conjunto de datos completo ya que se de esta última forma sería inmanejable.

Como la finalidad última de este componente es la de devolver el resultado de una consulta a una fuente de datos, principalmente BBDD, en formato JSON o XML entendible por algún otro módulo que consuma estos JSON o XML es necesario modelar la salida en base a lo que esperan estos módulos en especial a lo relativo a la paginación y a cómo proporcionan los datos para hacer la consulta.

Por tanto se ha desarrollado un módulo que permita ir creciendo en funcionalidad implementando distintos adaptadores para distintos módulos.

Aquellos servicios que no necesiten paginación no necesitan hacer ningún cambio pudiendo mantener sus llamadas a *RESTFul CARM* de la forma habitual

Modelo de clases

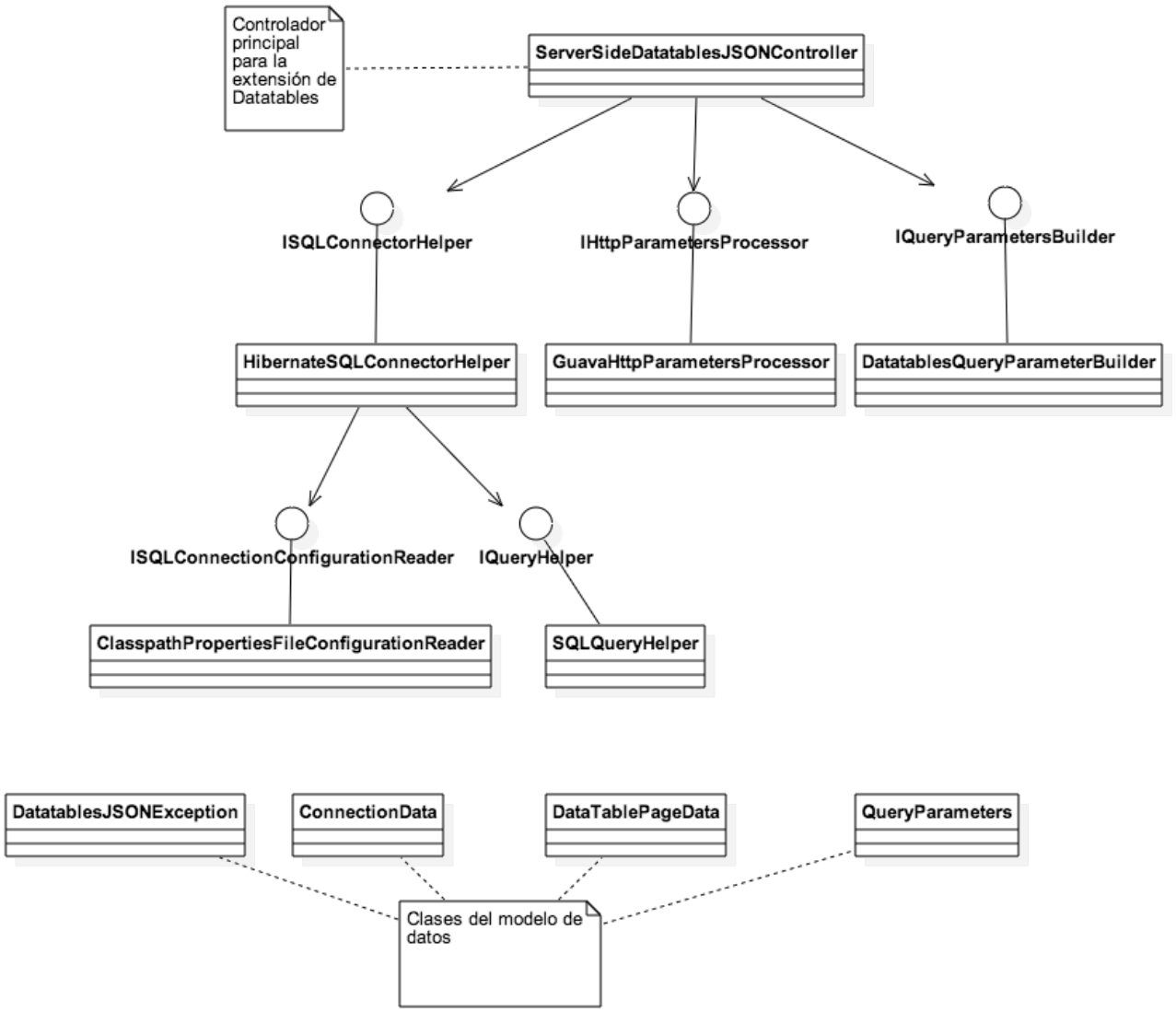

Figura 2. Modelo de clases del mecanismo de extension para Datatables

A continuación se describen las clases e interfaces más relevantes:

- **ServerSideDatatablesJSONController**: Punto de entrada para obtener JSON para las peticiones de datos paginados realizadas por los módulos *Datatables*
- **ISQLConnectorHelper**: Interfaz que gestiona las conexiones a las fuentes de datos
- IHttpParametersProcessor: Interfaz que gestiona la lectura de los parámetros de la petición HTTP
- **IqueryParametersBuilder**: Interfaz que se encarga de leer los parámetros HTTP a parámetros en el formato proporcionado por *Datatables* y convertirlo al formato esperado por la capa de datos
- **ISQLConnectionConfigurationReader**: Interfaz para obtener los datos de conexión para acceder a una BBDD (cadena de conexión, driver, usuario, password, etc)
- **IQueryHelper**: Interfaz que se encarga de construir consultas derivadas (*count(*)*) a partir de la consulta principal

- **HibernateSQLConnectionHelper**: Implementación de la interfaz *ISQLConnectorHelper* que realiza la conexión a BBDD utilizando Hibernate
- **GuavaHttpParametersProcessor**: Implementación de la interfaz *IHttpParametersProcessor* que procesa la cadena de parámetros HTTP mediante *Google Guava*
- **DatatablesQueryParameterBuilder**: Implementación de la interfaz *IqueryParametersBuilder* que obtiene los parámetros enviados por *Datatables* en la petición para saber como paginar, filtrar y ordenar
- **ClassPathPropertiesFileConfigurationReader**: Implementación de la interfaz *ISQLConnectionConfigurationReader* que lee los datos de conexión de un fichero de propiedades que reside en el classpath de la aplicación
- **ConnectionData**: Clase que almacena todos los parámetros necesarios para hacer la conexión a BBDD
- **DataTablePageData**: Clase que almacena todos los datos necesarios para generar el JSON en el formato esperado por *Datatables*.
- **QueryParameters**: Clase que almacena los parámetros de la consulta a realizar a la BBDD (datos de paginación, filtrado y ordenación)

Todas las dependencias entre clases están manejadas por *Spring*, por tanto es muy sencillo cambiar una implementación por otra y construir sobre esta base cualquier adaptador para cualquier modulo consumidor.

Formato de URLs

Para no obligar a todos los servicios que ya están consumiendo datos proporcionados por *RESTFul CARM*, las extensiones de paginación colgarán de la ruta /services. Por tanto cualquier servicio de paginación tendrá como url base

```
/RestFulCARM/services
```

Las extensiones se implementará como controladores *Spring MVC* que atiendan a un path específico a partir de /services. Por tanto la ruta completa a cualquier servicio de paginación será

```
/RestFulCARM/services/${PATH_DEL_SERVICIO_PAGINACION}
```

Extensión para *Datatables*

El primer módulo externo que se ha integrado en esta extensión es la librería Javascript *Datatables* (www.datatables.net) utilizada por los portales de Transparencia, IMIDA y CARMEuropa entre otros.

El formato del JSON que espera este componente puede verse en la siguiente URL https://datatables.net/examples/server_side/simple.html , dentro del apartado AJAX.

El propio *DataTables* se encarga de proporcionarnos:

- El número de registros por página
- El número de página
- El texto de consulta a la base de datos

- El nombre de las columnas sobre las que se debe consultar
- El campo sobre el que se ordena
- El criterio de ordenación: ascendente o descendente

Por tanto la extensión para *Datatables* recibirá directamente estos parámetros no soportando los parámetros de *RESTFul CARM* para paginar, ordenar o filtrar, ya que *Datatables* posee los suyos.

Además tampoco se soporta la generación de XML ya que *Datatables* espera los resultados en formato JSON haciendo innecesario generar la salida en XML.

Formato de URLs

Tomando como base lo definido en la definición general del formato de URLs, el módulo para paginación de *Datatables* ofrece la siguiente URL

```
/RestFulCARM/services/datatables/paginate/${NOMBRE_VISTA}
```

Donde ${NOMBRE_VISTA} es el nombre de alguna de las vistas definidas en el fichero de configuración de *RESTFulCARM*.

Datatables añadirá automáticamente a esta URL todos sus parámetros de control de paginación, filtrado y ordenación

Extensión de RESTFul CARM para parametrización de consultas

Esta extensión ofrece soporte a consultas SQL con parámetros posiciones expresados con el carácter '?'

La extensión recibirá una lista de parámetros separados por , y los sustituirá en la consulta SQL respetando la posición

```
select
ID_ANUNCIO_ORIGEN,SUMARIO_ORIGEN,FECHA_PUBLICACION_ORIGEN,RANGO,ID_TIPO_R
EF,NOMBRE_TIPO_REF,DESC_TIPO_REF,DESC_REF,ID_ANUNCIO_REFERENCIADO,RANGO_R
EF,TITULO_COMPACTO,ENLACE_ANUNCIO,FECHA_PUBLICACION,SUMARIO,VIGENTE_REF
from borm.TR_REFERENCIAS WHERE (id_anuncio_origen = ? and id_tipo_ref in
(3,4)) or (id_anuncio_referenciado = ? and id_tipo_ref in (1,2))
```

Por tanto la petición debe tener el mismo número de parámetros que la consulta SQL y deben tener el mismo orden en que deben sustituir en la sentencia SQL

Sólo están soportados parámetros de tipo numérico

Formato de URLs

Tomando como base lo definido en la definición general del formato de URLs, el módulo para pametrización de consultas ofrece la siguiente URL

```
/RestFulCARM/services/query/parameter/${NOMBRE_VISTA}/${PARAMETERS}
```

Donde ${NOMBRE_VISTA} es el nombre de alguna de las vistas definidas en el fichero de configuración de *RESTFulCARM*.

Dependencias

A continuación se detallan las dependencias incluidas en el pom.xml

- **guava v 18.0**: Empleado para procesar la cadena de parámetros HTTP mediante sus clases de procesamiento de cadenas *Splitter* y *Joiner*
- **spring-mvc y spring-portlet-mvc v 4.1.0 Release**: Empleado para crear controladores REST mediante anotaciones y asociarlos a rutas de forma sencilla
- **spring-core v 4.1.0 Release**: Empleado para inyección de dependencias entre las distintas clases e interfaces del módulo
- **jackson-databind**: Empleado para generar salida JSON de forma automática a partir de clases Java