Service Builder

Introduccion

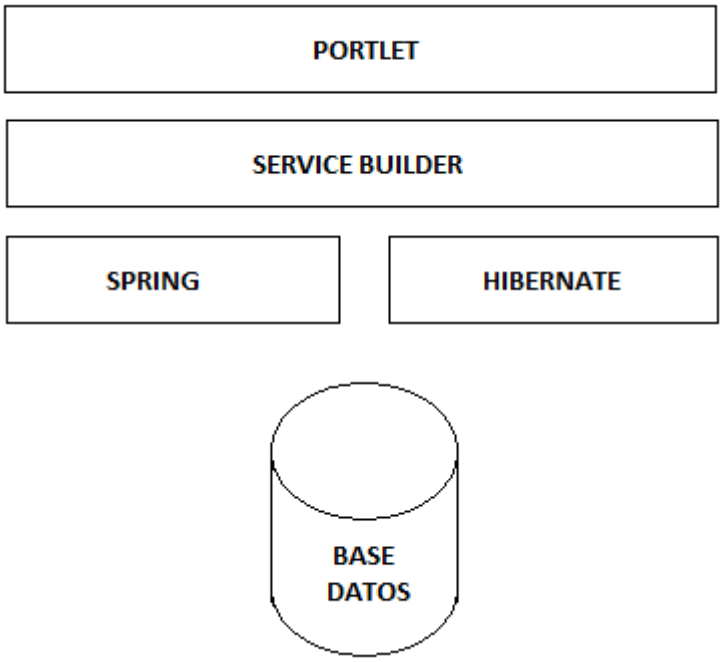
Permite definir una capa de Persistencia y de Servicio de forma automatica, considerando las funcionalidades mas basicas.

La idea de Service Builder, es homogeneizar la capa de servicios y de persistencia de las aplicaciones de Liferay, para ello proporciona un API y una herramienta (dentro del SDK de Liferay) que permite la creación de las clases de estas capas de forma automática siguiendo el patrón model-driven.

El propio Liferay esta creado empleando esta herramienta y la arquitectura que propone.

La herramienta, creará la capa de modelo y la de servicio de forma independiente, incluyendo las funcionalidades básicas de CRUD, permitiendo expandir dichas funcionalidades definiendo relaciones, Finders, Custom SQL, ...

Esta tecnología, emplea por debajo **Hibernate** y **Spring**, aunque en principio no se tiene porque tocar las configuraciones de estos frameworks, siendo su uso transparente para el desarrollador de **service-builder**.



No hay un tipo de proyecto propio para Service Builder, sino que se creará a partir de un proyecto de Portlets. Esto tiene su sentido ya que cualquier modelo de datos que se quiera incluir en el Portal, tendrá que ser consumido por algún Portlet.

Los proyectos Maven, crean la siguiente estructura de proyectos

- **XX-Service-Builder**: Proyecto padre Maven, que aglutina configuraciones comunes a los otros dos proyectos.

- **XX-Service-Builder-portlet**: Proyecto destinado a tener los Portlets, la implementación del servicio y donde se configurará el propio servicio con el fichero service.xml
- **XX-Service-Builder-portlet-service**: Proyecto con las interfaces y clases autogeneradas del servicio, de la cual depende el proyecto **XX-Service-Builder-portlet**.

Entidades

Se definen las entidades en el fichero **WEB-INF/service.xml**. Que empleará el siguiente esquema

```
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder 6.2.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_6_2_0.dtd">
```

Una vez definidas las entidades, se ha de lanzar la tarea **build-service** de Maven para que se autogeneren las clases necesarias.

Las clases **-Impl** generadas, tanto para **Model**, **Service** y **Persistence**, son modificables, pudiendo cambiar los comportamientos definidos por la herramienta de generacion de código.

Cualquier modificación sobre estas clases, que añada un nuevo método o cambie la firma de uno existente, exigirá que se vuelva a lanzar la tarea de Maven **build-service**.

Service.xml

El nodo principal de este fichero será <**service-builder**>, que tendrá como atributos

- **package-path**: Paquete java donde se encontrarán las entidades definidas.
- **auto-namespace-tables**: Crear el namespace de forma automática para las tablas.

Y como nodos hijos

- **author**: Autor del modelo.
- **namespace**: Prefijo que se incluye en el nombre de la tabla con el formato.
- **exceptions**: Excepciones propias que se pueden lanzar al trabajar con el modelo.
- **service-builder-import**: Referencia a otro fichero que defina entidades, para partir en varios ficheros la definición de la capa de servicios.
- **entity**: Etiqueta principal, que permite definir las entidades que define el modelo.

El nodo <**entity**> tiene como atributos

- **name**: Nombre de la entidad.
- **uuid**: (booleano) Indica si se ha de generar este valor de forma automática.
- **local-service**: Si se permite acceder con la interface local.
- **remote-service**: Si se permite acceder con la interface remota, creando el servicio web.
- **table**: Nombre de la tabla a emplear para diferenciarla de la clase que representa la entidad.

- **cache-enabled**: Permite activar el cacheo de las entidades.
- **deprecated**: Si se considera la entidad en desuso.
- **human-name**: Nombre legible de la entidad empleado a la hora de generar la documentación.
- **json-enabled**: Si se permite el acceso a través del servicio web con JSON.
- **persistence-class**: Nombre de la clase creada hija de XXPersistenceImpl.
- **session-factory**: Bean de Spring que gestiona las sesiones de hibernate.
- **data-source**: Bean de Spring que gestiona las conexiones a la base de datos.
- **tx-manager**: Bean de Spring que gestiona las transacciones.
- **trash-enabled**: Si se habilita la posibilidad de envar a la papelera estas entidades.
- **uuid-accessor**: (booleano) indica si se genera el Getter para el uuid

Y como nodos hijos

- **column**: Definición de un campo de la entidad.
- **finder**: Nueva funcionalidad de búsqueda relacionada con la entidad, dará lugar a unos nuevos métodos en la clase de implementación de la persistencia XXPersistenceImpl.
- **order**: Establece como se ordenan los registros retornados por las consultas por defecto.
- **reference**: Indica referencias a servicios ya existentes en Liferay o definidos en otro service.xml pero que carguen en el mismo classpath, para que estén accesibles directamente en la clase de implementación de servicio.
- **tx-required**: Patrón de nombre para todos aquellos métodos que necesiten transacciones, por defecto los siguientes patrones ya están incluidos*: add*, check*, clear*, delete*, set*, and update*, el resto serán de solo lectura.

El nodo <**column**>, puede definir relaciones con otras entidades del portal, así como unos campos particulares de la arquitectura de Liferay

Portal and site scope columns

Name	Type	Primary
companyId	long	no
groupId	long	no

User column

Name	Type	Primary
userId	long	no

Audit columns

Name	Type	Primary
userId	long	no
createDate	Date	no
modifiedDate	Date	no

Los campos

- **companyId**: hace referencia a la instancia del Portal
- **groupId**: hace referencia al Sitio

La referencia a estos campos permite que los distintos Sitios e instancias de Portal tengan sus propios datos de estas tipologías.

El nodo column tiene como atributos

- **name**: Nombre del campo empleado en los Get y Set.
- **type**: Tipo Java del campo.
- **primary**: Si es clave primaria, pueden existir varias columnas formado la clave primaria.
- **entity**: Cuando el type es Collection, indica el tipo de objetos en la colección.
- **accessor**: (booleano) Si se accede por Get y Set o por propiedad.
- **convert-null**: (booleano) Convertir a null al insertar en BD.
- **db-name**: Nombre del campo en BD.
- **filter-primary**: (booleano) Permite indicar una clave primaria para los finder, sino se indica será la clave primaria por defecto.
- **id-type**: Se emplea para definir el método de generación de la clave primaria, puede ser*: class, increment, sequence o identity.
- **id-param**: Se necesita para definir la clase o la secuencia que genera la clave primaria en el tipo de generación class o sequence. En caso de emplear la secuencia, esta ha de definirse en el fichero

- **json-enabled**: (booleano) Si está habilitada la conversión a JSON
- **lazy**: (booleano) Si la carga es perezosa.
- **localized**: (booleano) Si acepta traducciones.
- **mapping-table**: Indica el nombre de la tabla intermedia en relaciones m-n.

El nodo <**order**> tiene como atributos

- **by**: Indica si la ordenación es asc o desc.

El nodo <**order**> como sub-nodos puede tener

- **order-column**: Que indica las columnas que participan en la ordenación, pudiendo haber una o más.

El nodo <**order-column**> tiene como atributos

- **name**: Nombre de la columna.
- **case-sensitive**: (boolean) Si la ordenación contempla mayúsculas y minúsculas.
- **order-by**: En lugar del atributo by del nodo order, permite indicar un tipo de ordenación distinto para cada columna.

Relaciones

Se pueden definir relaciones

- 1-1
- 1-n
- n-m

1-1

Basta con añadir una columna a la entidad que refleje la FK con la otra entidad.

```
<entity name="Subasta">
    <column name="subastaId" type="long" primary="true" />
    <column name="productoId" type="long"/>
</entity>
```

Con esto se consigue que en el Servicio se genere un método **getProducto**, que habrá que implementar

```
public Producto getProducto() throws PortalException, SystemException{
    return ProductoLocalServiceUtil.getProducto(getProductoId());
}
```

1-n

Similar a la anterior, pero en este caso habrá que definir el tipo de la entidad a la que se referencia y como **type** de la columna **Collection**

```
<entity name="Puja">
    <column name="pujaId" type="long" primary="true" />
</entity>
<entity name="Subasta">
    <column name="subastaId" type="long" primary="true" />
    <column name="pujas" type="Collection" entity="Puja" mappingKey="pujaId"/>
</entity>
```

n-m

Similar a la anterior, pero en este caso se ha de definir en las dos entidades la relacion, haciendo referencia a la tabla que mantiene la relacion

```
<entity name="Asignatura">
    <column name="asignaturaId" type="long" primary="true" />
    <column name="alumnos" type="Collection" entity="Alumno" mappingKey="alumnoId"
mapping-table="Asignaturas_Alumnos"/>
</entity>
<entity name="Alumno">
    <column name="alumnoId" type="long" primary="true" />
    <column name="asignaturas" type="Collection" entity="Asignatura"
mappingKey="asignaturaId" mapping-table="Asignaturas_Alumnos"/>
</entity>
```

Finders

Métodos que se generan en la capa de persistencia, que definen consultas sencillas, con clausulas de **where**.

Se emplea el nodo **finder**.

```
<finder name="producto" return-type="Collection">
    <finder-column name="producto" />
</finder>
```

El nodo **finder-column** define los campos por los que se realiza la búsqueda, definiendo los atributos que recibirán los métodos generados, si se aplican múltiples **finder-column** al **finder**, se aplica por defecto la cláusula **AND**.

El nodo **finder** tiene los siguientes atributos

- **db-index**: (boolean) Si es true, se genera automáticamente un índice de BD.
- **name**: Nombre del método generado en la capa de persistencia.
- **return-type**: Tipo retornado, puede ser Collection o una Entidad.
- **unique**: Indica que se retorna solo una entidad.
- **where**: Permite definir una clausula de where estatica, que no se ve afectada por los parametros de la consulta **id != 1**

El nodo **finder-column** tiene como atributos:

- **arrayable-operator**: Permite definir el valor AND o OR, y generara un nuevo método de finder, que aceptará un array por parámetro y creara una clausula concatenando todos los valores del array con AND o OR.
- **case-sensitive**: Solo si la columna es de tipo String.
- **comparator**: Indica el tipo de comparación implementada por el método finder, puede tomar como valores: =, !=, <, ⇐, >, >=, o LIKE.
- **name**: Nombre de la columna.