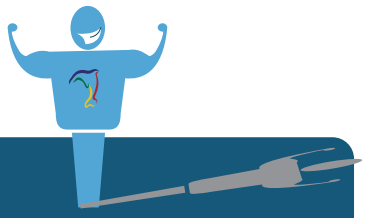
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

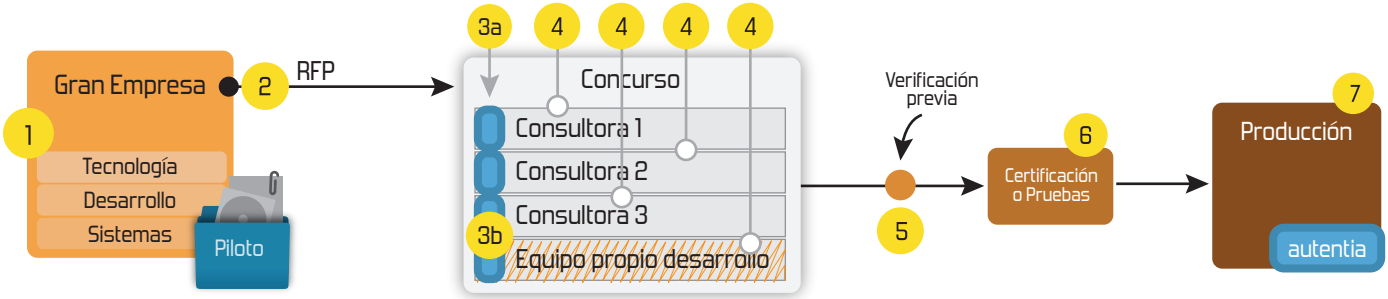
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Alejandro Perez García 2003-2007
Alejandro es Socio fundador de Autentia y nuestro experto en J2EE, Linux y optimización de aplicaciones empresariales.

Si te gusta lo que ves, **puedes contratarle** para impartir **cursos presenciales** en tu empresa o para ayudarte en proyectos (Madrid).

Contacta: alejandropg@autentia.com.

Fecha de creación del tutorial: 2007-06-25

Hibernate y Joins con la clase Criteria

Creación: 20-06-2007

Índice de contenidos

1. Introducción

Dentro de Hibernate (http://www.hibernate.org/) podemos encontrar la interfaz `Criteria` (`org.hibernate.Criteria`). Esta interfaz nos permite especificar consultas programáticamente (en base a clases y métodos de estas clases) sobre nuestras entidades definiendo un conjunto de restricciones.

El uso de la clase `Criteria` puede ser muy conveniente cuando tenemos que componer consultas de forma dinámica, por ejemplo con las típicas pantallas de búsqueda, donde el usuario introduce una serie de criterios. En estos casos en vez de ir componiendo un String con el HQL en función de los datos introducidos por el usuario, puede resultar más cómodo usar la clase `Criteria`.

En este tutorial vamos a ver como hacer "joins" entre entidades relacionadas, y las implicaciones que esto puede tener.

Este tutorial se basa en el tutorial http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateAnnotations. El tutorial esta desarrollado con JUnit 4 (http://www.junit.org/), la sesión de Hibernate se abre y se cierra por cada método de test, y todas las operaciones se hacen dentro de una transacción.

Aquí podéis encontrar un archivo comprimido con todo el código. Es un proyecto de Maven (http://maven.apache.org/), así que las dependencias necesarias para compilar y ejecutar se os bajarán de Internet.

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Asus G1 (Core 2 Duo a 2.1 GHz, 2048 MB RAM, 120 GB HD).
- Sistema Operativo: GNU / Linux, Debian (unstable), Kernel 2.6.21, KDE 3.5
- Máquina Virtual Java: JDK 1.6.0-b105 de Sun Microsystems
- Eclipse 3.2.2
- Hibernate 3.2.2.ga
- Hibernate Tools 3.2.0 Beta9
- MySql 5.0.41-2
- JUnit 4.3.1
- Maven 2.0.7

3. Usando Criteria.createCriteria()

En la primera aproximación vamos a usar el siguiente código.

```
@Test
public void criteriaTripleJoin() {
   log.info("\n\n*** criteriaTripleJoin ***\n");

   Criteria criteria = session.createCriteria(Campaign.class)
      .createCriteria("subcampaigns")
      .createCriteria("budgets")
      .setResultTransformer(CriteriaSpecification.DISTINCT_ROOT_ENTITY)
      ;

   List<Campaign> campaigns = criteria.list();
   prettyPrinter(campaigns);
}
```

Si observamos el resultado volcado por prettyPrinter (también se están volcando las consultas que hace Hibernate, ya que en el fichero de configuración `hibernate.cfg.xml` tenemos activa la opción `<property name="show_sql">true</property>`):

```
*** criteriaTripleJoin ***

Hibernate: select this_.id as id6_2_, this_.name as name6_2_, subcampaig1_.id as id7_0_,
subcampaig1_.campaignId as campaignId7_0_, subcampaig1_.name as name7_0_,budget2_.id as id8_1_,
budget2_.name as name8_1_, budget2_.subcampaignId as subcampa3_8_1_ from curso.Campaign this_
inner join curso.Subcampaign subcampaig1_ on this_.id=subcampaig1_.campaignId inner join
curso.Budget budget2_ on subcampaig1_.id=budget2_.subcampaignId

- Campaign { id=1, name=CampaÃ±a 1 }

Hibernate: select subcampaig0_.campaignId as campaignId1_, subcampaig0_.id as id1_,
subcampaig0_.id as id7_0_, subcampaig0_.campaignId as campaignId7_0_,
subcampaig0_.name as name7_0_ from curso.Subcampaign subcampaig0_
where subcampaig0_.campaignId=?

-    Subcampaign { id=2, name=CampaÃ±a 1 SubcampaÃ±a 2 }

Hibernate: select budgets0_.subcampaignId as subcampa3_1_, budgets0_.id as id1_,
budgets0_.id as id8_0_, budgets0_.name as name8_0_, budgets0_.subcampaignId as subcampa3_8_0_
from curso.Budget budgets0_ where budgets0_.subcampaignId=?

-    Subcampaign { id=3, name=CampaÃ±a 1 SubcampaÃ±a 3 }

Hibernate: select budgets0_.subcampaignId as subcampa3_1_, budgets0_.id as id1_,
budgets0_.id as id8_0_, budgets0_.name as name8_0_, budgets0_.subcampaignId as subcampa3_8_0_
from curso.Budget budgets0_ where budgets0_.subcampaignId=?

-      Budget { id=6, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 3 }
-      Budget { id=5, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 2 }
-      Budget { id=4, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 1 }
-    Subcampaign { id=1, name=CampaÃ±a 1 SubcampaÃ±a 1 }

Hibernate: select budgets0_.subcampaignId as subcampa3_1_, budgets0_.id as id1_,
```

```
budgets0_.id as id8_0_, budgets0_.name as name8_0_, budgets0_.subcampaignId as subcampa3_8_0_
from curso.Budget budgets0_ where budgets0_.subcampaignId=?

-      Budget { id=1, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 1 }
-      Budget { id=3, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 3 }
-      Budget { id=2, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 2 }
- Campaign { id=2, name=CampaÃ±a 2 }

Hibernate: select subcampaig0_.campaignId as campaignId1_, subcampaig0_.id as id1_,
subcampaig0_.id as id7_0_, subcampaig0_.campaignId as campaignId7_0_,
subcampaig0_.name as name7_0_ from curso.Subcampaign subcampaig0_
where subcampaig0_.campaignId=?

-    Subcampaign { id=4, name=CampaÃ±a 2 SubcampaÃ±a 1 }

Hibernate: select budgets0_.subcampaignId as subcampa3_1_, budgets0_.id as id1_,
budgets0_.id as id8_0_, budgets0_.name as name8_0_, budgets0_.subcampaignId as subcampa3_8_0_
from curso.Budget budgets0_ where budgets0_.subcampaignId=?

-      Budget { id=7, name=CampaÃ±a 2 SubcampaÃ±a 1 Presupuesto 1 }
-    Subcampaign { id=5, name=CampaÃ±a 2 SubcampaÃ±a 2 }

Hibernate: select budgets0_.subcampaignId as subcampa3_1_, budgets0_.id as id1_,
 budgets0_.id as id8_0_, budgets0_.name as name8_0_, budgets0_.subcampaignId as subcampa3_8_0_
 from curso.Budget budgets0_ where budgets0_.subcampaignId=?
```

Podemos observar como, cada vez que se entra en una relación se vuelve a realizar una consulta contra la base de datos para recuperar la información. Esto se debe a que las relaciones están configuradas con `lazy=true` (ver tutorial http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateJoin), y cada vez que se navega por la relación (`getSubcampaigns()` o `getBudgets()`) se vuelve a hacer una consulta para recuperar la información.

Pero entonces ¿para que sirve el método `createCriteria()`? Bien, el método `createCriteria()` sirve para definir nuevas restricciones sobre las clases de la relación, pero el uso de `createCriteria()` no quiere decir que se recuperen todos los datos de la relación, de hecho no se recuperará ninguno.

De hecho, en nuestro ejemplo, como no especificamos ninguna restricción sobre ninguna de las clases relacionadas, podríamos haber quitado el `createCriteria("subcampaigns")` y el `createCriteria("budgets")`.

4. Usando Criteria.setFetchMode

Este segundo ejemplo es muy similar al anterior, salvo que hemos cambiado los `createCriteria()` por `setFetchMode()`. Cabe destacar en este caso que para hacer referencia los presupuestos lo hacemos a través de "`subcampaigns.budgets`", es decir, los `setFetchMode()` siempre se refieren a la entidad definida en el último createCriteria (en nuestro ejemplo `session.createCriteria(Campaign.class)`):

```
@Test
public void criteriaTripleJoinWithFetch() {
   log.info("\n\n*** criteriaTripleJoinWithFetch ***\n");

   Criteria criteria = session.createCriteria(Campaign.class)
      .setFetchMode("subcampaigns", FetchMode.JOIN)
      .setFetchMode("subcampaigns.budgets", FetchMode.JOIN)
      .setResultTransformer(CriteriaSpecification.DISTINCT_ROOT_ENTITY)
      ;

   List<Campaign> campaigns = criteria.list();
   prettyPrinter(campaigns);
}
```

Si ahora observamos la salida, veremos:

```
*** criteriaTripleJoinWithFetch ***

Hibernate: select this_.id as id6_2_, this_.name as name6_2_,
subcampaig2_.campaignId as campaignId4_, subcampaig2_.id as id4_,
subcampaig2_.id as id7_0_, subcampaig2_.campaignId as campaignId7_0_,
subcampaig2_.name as name7_0_, budgets3_.subcampaignId as subcampa3_5_,
 budgets3_.id as id5_, budgets3_.id as id8_1_, budgets3_.name as name8_1_,
  budgets3_.subcampaignId as subcampa3_8_1_ from curso.Campaign this_
  left outer join curso.Subcampaign subcampaig2_ on this_.id=subcampaig2_.campaignId
left outer join curso.Budget budgets3_ on subcampaig2_.id=budgets3_.subcampaignId

- Campaign { id=1, name=CampaÃ±a 1 }
-    Subcampaign { id=2, name=CampaÃ±a 1 SubcampaÃ±a 2 }
-    Subcampaign { id=1, name=CampaÃ±a 1 SubcampaÃ±a 1 }
-      Budget { id=3, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 3 }
-      Budget { id=2, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 2 }
-      Budget { id=1, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 1 }
-    Subcampaign { id=3, name=CampaÃ±a 1 SubcampaÃ±a 3 }
-      Budget { id=4, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 1 }
-      Budget { id=5, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 2 }
-      Budget { id=6, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 3 }
- Campaign { id=2, name=CampaÃ±a 2 }
-    Subcampaign { id=5, name=CampaÃ±a 2 SubcampaÃ±a 2 }
-    Subcampaign { id=4, name=CampaÃ±a 2 SubcampaÃ±a 1 }
-    Budget { id=7, name=CampaÃ±a 2 SubcampaÃ±a 1 Presupuesto 1 }
```

Esta vez parece que si está funcionando correctamente, se está haciendo una única consulta al principio, y luego se navega por todas las

relaciones, pero esta vez ya no hace nuevas consultar a la base de datos. Con `setFetchMode()` estamos definiendo como se quieren recuperar los objetos de las relaciones, ignorando la configuración que tengamos en los atributos `lazy` o `fetch`.

Sin embargo no podemos cantar victoria tan rápido. Si os habéis fijado en ambos ejemplos usamos `setResultTransformer (CriteriaSpecification.DISTINCT_ROOT_ENTITY)`, con esto estamos indicando a Hibernate que queremos que nos devuelva sólo las entidades diferentes de la entidad raíz (`Campaign.class`). El problema que tiene esto es que la búsqueda de entidades diferentes se hace en memoria, es decir, Hibernate en vez de usar un "`distinct`" en la sentencia SQL, se trae todos los resultado para luego hacer la búsqueda en memoria. Esto puede ser muy desaconsejable, ya que se podría dar el caso de traernos multitud de resultados cuando sólo un pequeño porcentaje son realmente diferentes.

Solucionar esto con la clase `Criteria` no es tarea fácil. En el próximo ejemplo se ve una aproximación.

5. Usando proyecciones

Con la clase Projection, podemos definir consultar escalares, es decir, consultas donde no nos traemos entidades enteras, sino que definimos uno a uno los distintos campos que queremos recuperar de la base de datos

```
@Test
public void projection() {
   log.info("\n\n*** projection ***\n");

   Criteria criteria = session.createCriteria(Campaign.class)
      .setProjection(Projections.distinct(Projections.projectionList()
         .add(Projections.property("id"), "id")
         .add(Projections.property("name"), "name")
      )).setResultTransformer(Transformers.aliasToBean(Campaign.class))
      ;

   List<Campaign> campaigns = criteria.list();
   prettyPrinter(campaigns);
}
```

Si vemos la salida:

```
*** projection ***

Hibernate: select distinct this_.id as y0_, this_.name as y1_ from curso.Campaign this_

- Campaign { id=1, name=CampaÃ±a 1 }
- Campaign { id=2, name=CampaÃ±a 2 }
```

Podemos observar como sólo se ha traído la entidad raíz (`Campaign.class`), y como además las relaciones no son navegables!!! Es decir, aunque hemos usado un `Transformer` para convertir las columnas sueltas en una entidad correspondiente, estos objetos no se ven como objetos de Hibernate, por lo que las operaciones que hacemos sobre ellos son ignoradas por Hibernate.

En el foro http://forum.hibernate.org/viewtopic.php?t=941669, podemos encontrar más información sobre el uso de `Projection`, pero veremos que su uso es complicado, y rara vez conseguiremos obtener lo que realmente buscamos.

6. Usando HQL

Parece que en esta ocasión, cuando queremos hacer joins y navegar por las relaciones sin que Hibernate vuelva a lanzar consultas contra la base de datos, la clase `Criteria` se nos queda un poco corta. En este último ejemplo mostramos como hacerlo con HQL:

```
@Test
public void hqlJoin() {
   log.info("\n\n*** hqlJoin ***\n");

   String hql = "select distinct c from Campaign c join fetch c.subcampaigns s join fetch s.budgets";

   Query query = session.createQuery(hql);

   List<Campaign> campaigns = query.list();
   prettyPrinter(campaigns);
}
```

Si inspeccionamos la salida nos encontraremos con:

```
*** hqlJoin ***

Hibernate: select distinct campaign0_.id as id6_0_, subcampaig1_.id as id7_1_,
 budgets2_.id as id8_2_, campaign0_.name as name6_0_,
 subcampaig1_.campaignId as campaignId7_1_, subcampaig1_.name as name7_1_,
 subcampaig1_.campaignId as campaignId0__, subcampaig1_.id as id0__,
 budgets2_.name as name8_2_, budgets2_.subcampaignId as subcampa3_8_2_,
 budgets2_.subcampaignId as subcampa3_1__, budgets2_.id as id1__ from
 curso.Campaign campaign0_ inner join curso.Subcampaign subcampaig1_ on
 campaign0_.id=subcampaig1_.campaignId inner join curso.Budget budgets2_
 on subcampaig1_.id=budgets2_.subcampaignId

- Campaign { id=1, name=CampaÃ±a 1 }
-    Subcampaign { id=3, name=CampaÃ±a 1 SubcampaÃ±a 3 }
-       Budget { id=6, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 3 }
```

```
-      Budget { id=5, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 2 }
-      Budget { id=4, name=CampaÃ±a 1 SubcampaÃ±a 3 Presupuesto 1 }
-    Subcampaign { id=1, name=CampaÃ±a 1 SubcampaÃ±a 1 }
-      Budget { id=3, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 3 }
-      Budget { id=2, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 2 }
-      Budget { id=1, name=CampaÃ±a 1 SubcampaÃ±a 1 Presupuesto 1 }
- Campaign { id=2, name=CampaÃ±a 2 }
-    Subcampaign { id=4, name=CampaÃ±a 2 SubcampaÃ±a 1 }
-      Budget { id=7, name=CampaÃ±a 2 SubcampaÃ±a 1 Presupuesto 1 }
```

Como podemos ver es justo lo que necesitábamos. Con una sola consulta a la base de datos, obtenemos las entidades con las relaciones debidamente rellenas, y el `distinct` se está haciendo en la base de datos, y no en memoria.

7. Conclusiones

Ya hemos visto varias capacidades de Hibernate, y la clase `Criteria` es otra de ellas. También hemos comentado las ventajas de la clase `Criteria` y en que ocasiones puede ser conveniente su uso. Pero siempre tenemos que tener presente el antipatrón del Martillo de Oro (Golde Hammer): Cuando tu única herramienta es un martillo todo parecen clavos.

Es decir, no debemos querer solucionar todos los problemas de la misma manera. Hay que intentar elegir la solución más adecuada para cada ocasión, y en esta ocasión se ha demostrado que la clase `Criteria` no es la solución más adecuada, sino que es mucho más sencillo y efectivo usar HQL.

8. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software)

Socio fundador de Autentia (Formación, Consultoría, Desarrollo de sistemas transaccionales)

mailto:alejandropg@autentia.com

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo"

http://www.autentia.com

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Creación automática de recursos Hibernate con Middlegen	En este tutorial aprendereis como utilizar la herramienta middlegen para generar distintas capas de persistencia (CMP 2.0, JDO, Hibernate, Torque), a partir de un modelo físico de datos, de un modo automático, mediante el uso de la herramienta middlegen
Hibernate Tools y la generación de código	En este tutorial vamos a ver como usar estas herramientas para hacer el esqueleto de una pequeña aplicación, de manera muy sencilla, generando código a partir de las tablas creadas en la base de datos.
Hibernate 3.1, Colecciones, Fetch y Lazy	En este tutorial vamos a ver cómo se comportan ciertas relaciones, y cómo podemos optimizar las consultas a la base de datos con Hibernate
Hibernate y las anotaciones de EJB 3.0	En este tutorial Alejandro Pérez nos muestra las ventajas que nos aporta Hibernate y las anotaciones de EJB 3.0
Hibernate 3 y los tipos de datos para cadenas largas	En este tutorial se contará la experiencia que hemos tenido a la hora de manejar los diferentes tipos de datos existentes para grandes cadenas de texto, tales como el tipo Clob de Oracle, o los tipos TEXT de MySQL y SQLServer, utilizando la última versión
Introducción a Hibernate	Cesar Crespo nos enseña como utilizar unos de los sistemas más extendidos de mapeo de objetos a estructuras relacionales (tablas de base de datos)
Manejar dos bases de datos distintas con Hibernate	Alejandro Pérez nos enseña como manejar dos bases de datos distintas con Hibernate