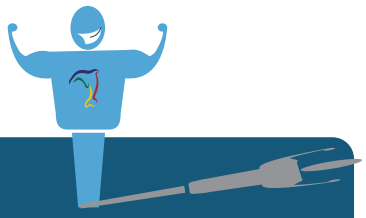
autentia

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...

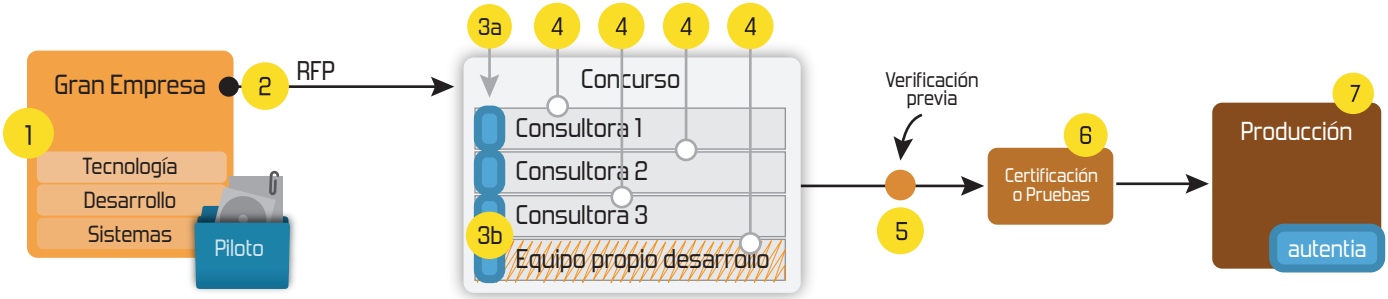
1. Desarrollo de componentes y proyectos a medida



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

NUEVO ¿Quieres saber cuánto ganas en relación al mercado? pincha aquí...

Ver cursos que ofrece Autentia

Descargar comics en PDF y alta resolución

[¡NUEVO!] 2008-02-03 2008-01-29 2008-01-27 2008-01-23

Estamos escribiendo un libro sobre la profesión informática y estas viñetas formarán parte de él. Puedes opinar en la seccion comic.

Tutorial desarrollado por

Alejandro Pérez García

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). Puedes encontrarme en Autentia

Catálogo de servicios de Autentia

Descargar (6,2 MB)

Descargar en versión comic (17 MB)

AdictosAlTrabajo.com es el Web de difusión de conocimiento de Autentia.

autentia real business solutions

Catálogo de cursos

Descargar este documento en formato PDF: hibernateEquals.pdf

Fecha de creación del tutorial: 2008-02-05

Como implementar el método equals(Object) en objetos persistentes de Hibernate, y otras consideraciones

Creación: 13-01-2008

Índice de contenidos

Últimos tutoriales

1. Introducción

En la clase `java.lang.Object` (y por lo tanto, o mejor dicho, y por herencia, en todas las demás clases) tenemos dos métodos que suelen ser dos grandes olvidados:

- `public boolean equals(Object obj)`
- `public int hashCode()`

Estos métodos son especialmente importantes si vamos a guardar nuestros objetos en cualquier tipo de colección: listas, mapas (aquí es especialmente importante el método `hashCode`), ... y más aun si los objetos que vamos a guardar en la colección son serializables. En este último caso es especialmente necesarios reescribir estos métodos ya que la implementación proporcionada por la clase Object trabaja con las referencias, y si un objeto lo serializamos y lo deserializamos, tendrá diferente referencia, sin embargo seguirá siendo el mismo objeto. Es decir, si tengo un objeto persona donde tengo guardados mis datos, y serializo este objeto a un fichero y luego lo vuelvo a leer, tendré una instancia diferente (un nuevo objeto con diferente referencia), pero seguiré siendo â€œyoâ€□.

Estos métodos van a permitir encontrar un objeto en una colección (`equals`) o localizar el objeto en un mapa (`hashCode`).

Se debe cumplir que, si dos objetos son iguales según el método `equals`, deben generar el mismo entero al llamar a `hashCode` (si no son iguales pueden producir el mismo entero, pero es aconsejable que generen enteros diferentes para aumentar el rendimiento de los mapas).

De hecho muchos puristas consideran que una clase no está correctamente implementada si no tiene debidamente sobreescritos estos métodos. Con esto os podéis hacer una idea de la importancia de estos métodos.

En este tutorial vamos a ver algunas importantes consideraciones a tener en cuenta cuando sobreescribimos estos métodos, especialmente si vamos a trabajar con Hibernate o con cualquier otro sistema que genere proxys de nuestras clases de forma dinámica en tiempo de ejecución (por ejemplo con el uso de aspectos).

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 2.6.23, KDE 3.5
- JDK 1.5.0_14 (instalada con el paquete sun-java5-jdk de Debian)
- Eclipse Europa

3. Ojo con el método equals

Al reescribir el método `equals`, tenemos que fijarnos bien en que el parámetro de entrada es de tipo `Object`. Es decir el método siempre debe ser: `public boolean equals(Object obj)`.

Es un descuido muy común usar como parámetro de entrada una instancia de la misma clase donde se está reescribiendo el método, por ejemplo, para la clase `Person`: `public boolean equals(Person obj)`. Esto es un error ya que no estamos sobreescribiendo el método `equals` de nuestro padre, sino que estamos creando un nuevo método, con el mismo nombre, pero con parámetros diferentes. Esto provoca que cuando se invoque el método `equals` intentando hacer uso del polimorfismo (por ejemplo en las colecciones) no se llame al método que hemos escrito nosotros, sino a la implementación por defecto proporcionada por la clase `Object`.

4. Hagámoslo fácil, usemos el IDE

Actualmente, prácticamente todos los IDEs (entornos integrados de desarrollo) tienen alguna manera de generar de forma automática estos métodos. Por ejemplo en Eclipse podemos hacer: *Source --> Generate hashCode() and equals()...* --> marcamos los atributos que queremos que se tengan en cuenta en estos métodos.

> **Importante**: Siempre deberíamos tener en cuenta los mismos atributos para calcular el `equals` y para calcular el `hashCode`. Esta regla es válida independientemente de si generamos estos métodos de forma automática o los escribimos a mano.

Por ejemplo para la siguiente clase:

```
public class Person implements Serializable {
    private String name;
    private String surname;
    private int age;

    ...
}
```

Deberíamos obtener la siguiente implementación si elegimos todos los atributos:

```
    @Override
    public int hashCode() {
        final int prime = 31;
```

```
        int result = 1;
        result = prime * result + age;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        result = prime * result + ((surname == null) ? 0 : surname.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        final Person other = (Person)obj;
        if (age != other.age) return false;
        if (name == null) {
            if (other.name != null) return false;
        } else if (!name.equals(other.name)) return false;
        if (surname == null) {
            if (other.surname != null) return false;
        } else if (!surname.equals(other.surname)) return false;
        return true;
    }
```

Estas implementaciones tiene el inconveniente de que cada vez que añadimos o quitamos un atributo hay que volver a generar el método o modificarlo a mano.

5. Hagámoslo más fácil, usemos commons-lang

Apache nos proporciona la librería `commons-lang` (http://commons.apache.org/lang/). En esta librería podemos encontrar multitud de ayudas de uso muy común en cualquier aplicación (os recomiendo que le echéis un buen vistazo). Entre estas ayudas vamos a encontrar métodos para implementar `equals` y `hashCode` por reflexión.

De esta forma nuestros métodos quedarían de la siguiente forma:

```
    @Override
    public int hashCode() {
        return HashCodeBuilder.reflectionHashCode(this);
    }

    @Override
    public boolean equals(Object obj) {
        return EqualsBuilder.reflectionEquals(this, obj);
    }
```

De esta forma no sólo escribimos menos, sino que podemos cambiar los atributos de la clase sin preocuparnos de estos métodos.

> **Importante**: Podemos tener inconvenientes si usamos un SecurityManager, ya que esto podría causar una excepción al intentar acceder por reflexión a los atributos privados de la clase.

6. Usando Hibernate, comienzan los problemas

Con las dos implementaciones que hemos visto vamos a encontrarnos con problemas si usamos Hibernate (o EJB 3), si nuestra clase es una clase persistente. Prácticamente todos los problemas que se van a dar vienen provocados porque en varias ocasiones Hibernate no trabaja con nuestra clase directamente, sino con un proxy que genera dinámicamente en tiempo de ejecución. Por ejemplo: si usamos el método `load` para recuperar un objeto, en el caso de relaciones con inicilización â€œlazyâ€□, ...

6.1. Identificación del tipo del objeto con el que comparamos

Una de las primeras cosas que siempre se hace en el `equals` es mirar si el tipo del objeto que nos pasan como parámetro es igual que nuestro tipo. Lo veíamos con el código:

```
    if (getClass() != obj.getClass()) return false;
    final Person other = (Person)obj;
```

Esto nos va a dar problemas con Hibernate, ya que como crea un proxy, las clases no van a ser realmente las mismas. Por esto debemos usar la sentencia `instanceof`, de esta forma:

```
    if (!(obj instanceof Person)) return false;
    final Person other = (Person)obj;
```

Aquí no habrá problema porque el proxy que crea Hibernate extiende de nuestra clase `Person`, así que si preguntamos si una instancia del tipo del proxy creado por Hibernate es â€œ`instanceof`â€□ `Person`, la respuesta será `true` (el tipo también se hereda en un relación de herencia).

6.2. Accediendo directamente a los atributos

Tanto en el código del estilo:

```
if (name == null) {
    if (other.name != null) return false;
} else if (!name.equals(other.name)) return false;
```

como en lo que hace internamente la clase `EqualsBuilder`, se accede directamente al valor de los atributos; es decir, no se usan los getter correspondientes.

Esto también nos puede dar problemas si hemos recuperado el objeto mediante el método `load` de la clase `Session`. El problema viene provocado porque al cargar el objeto con `load`, Hibernate lo que devuelve es un proxy. Este proxy sirve para que no se acceda al la base de datos hasta que realmente se acceda al objeto; es decir, hasta que no se llame a alguno de los getters no se recuperan los valores de la base de datos, por lo que, en nuestro ejemplo, `name` y `surname` valdrán `null`.

De esto se deduce que si comparamos un objeto con otro que acabamos de recuperar con `load`, el resultado del método `equals` será `false`, aunque los objetos sean iguales, ya que estaremos comparando un objetos con otro cuyos atributos están a `null` porque todavía no se han recuperado de la base de datos.

Este mismo problema lo vamos a tener con las relaciones entre objetos con inicialización â€œlazyâ€□, es decir, hasta que no se llama al getter no se recupera la colección.

Para solucionar esto deberíamos garantizar que el acceso a los atributos se haga mediante los getters (veremos la solución un poco más adelante).

6.3. Otros problemas asociados a las relaciones entre objetos persistentes

Imaginemos que la clase `Person` tiene un atributo que es una lista de Telefonos: `private List<Phone> phones;`

La clase `Phone` también se persiste mediante Hibernate, así que tenemos una relación de 1 a n entre `Person` y `Phone`.

Si en el `equals` de `Person` hacemos lo siguiente:

```
if (!this.getPhones().equals(other.getPhones())) return false;
```

nos vamos a encontrar con un problema: el resultado del `equals` va a ser `false`, a pesar de estar usando los getters como dijimos en el caso anterior.

Esto se debe a que cuando Hibernate recupera una relación de este tipo usa sus propias implementaciones de las colecciones. En este caso Hibernate nos puede estar devolviendo un `PersistenBag` que tiene atributos diferentes a los de un `ArrayList` normal (por ejemplo tiene un atributo que identifica que extremo de la relación es el propietario de la misma), por lo que el la comparación fallará.

Para evitar esto, lo mejor es que en la implementación del método `equals` de `Person`, recorramos uno a uno los elementos de la lista y los vallamos comparando (veremos la solución un poco más adelante).

6.4. ¿Que pasa con el id de persistencia?

Hibernate nos recomienda que usemos un identificador â€œartificialâ€□ para los objetos persistentes, es decir un id que no tenga nada que ver con los datos de negocio (si luego queremos hacer accesos por los campos de negocio basta con hacer índices sobre ellos). De esta manera en la clase `Person` nos aparecerá un atributo del estilo:

```
private Integer id;
```

Este atributo nunca habrá que tenerlo en cuenta en el `equals` (y por lo tanto tampoco en el `hashCode`).

En un principio podemos estar tentados a usarlo ya que si dos objetos tienen el mismo id, es que son el mismo. Pero debemos recordar que un objeto sólo tiene id si ya ha sido persistido. Es decir podríamos estar comparando un objeto ya persistido con otro que todavía no ha sido persistido; estos dos objetos podrían ser iguales, pero el segundo todavía no tiene id, por lo que la comparación devolvería `false`.

6.5. Una solución a todos los problemas

Teniendo en cuenta todos los casos comentados anteriormente, la clase `Person` nos podrían quedar de la siguiente manera:

```
@Entity
public class Person {

    private Integer id = null;
    private String name = null;

    // ... otros atributos

    private List<Phone> phone = new ArrayList<Phone>();

    @Id
    @GeneratedValue
```

```
        @Column(unique = true, nullable = false)
        public Integer getId() {
            return id;
        }

        /** Para uso de Hibernate. */
        @SuppressWarnings("unused")
        private void setId(Integer id) {
            this.id = id;
        }

        @OneToMany(fetch = FetchType.LAZY, cascade = { CascadeType.ALL })
        @JoinColumn(name = "phone_id", referencedColumnName = "id")
        public List<Phone> getPhones() {
            return phones;
        }

        /** Para uso exclusivo de Hibernate. */
        @SuppressWarnings("unused")
        private void setRecordFiles(List<RecordFile> recordFiles) {
            this.recordFiles = recordFiles;
        }

        // ... otros getter y setter

        @Override
        public int hashCode() {
            final HashCodeBuilder hcb = new HashCodeBuilder();
            hcb.append(getName());

            // ... otros atributos, pero no el id

            for (Phone phone : phones) {
                hcb.append(phone);
            }
            return hcb.toHashCode();
        }

        @Override
        public boolean equals(Object obj) {
            boolean isEquals = false;
            try {
                final Person other = (Person)obj;
                final EqualsBuilder eqb = new EqualsBuilder();
                eqb.append(getName(), other.getName());

                // ... otros atributos, pero no el id

                if (getPhones().size() != other.getPhones().size()) {
                    return false;
                }
                for (int i = 0; i < phones.size(); i++) {
                    eqb.append(phones.get(i), other.getPhones().get(i));
                }
                isEquals = eqb.isEquals();

            } catch (Exception e) {
                // Sobre todo si no se puede hacer la conversión de tipos,
                // y en general si se produce cualquier error.
                isEquals = false;
            }

            return isEquals;
        }
    }
```

7. Conclusiones

Hemos visto como algo aparentemente inocente se puede convertir en un quebradero de cabeza. Por eso desde Autentia (http://www.autentia.com) siempre os animamos a que profundicéis en las tecnologías que usáis, y que implicaciones tienen. Y os intentamos facilitar las cosas con este tipo de tutoriales.

Espero que os haya servido de ayuda y que aclare un poco las cosas.

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. - â€œSoporte a Desarrolloâ€□

http://www.autentia.com

- Puedes opinar sobre este tutorial haciendo clic aquí.
- Puedes firmar en nuestro libro de visitas haciendo clic aquí.
- Puedes asociarte al grupo AdictosAlTrabajo en XING haciendo clic aquí.
- Añadir a favoritos Technorati. ADD THIS BLOG TO MY Technorati FAVORITES

Servicio de notificaciones:

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales.

Formulario de subcripción a novedades:

E-mail [ ] Aceptar

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	pdf
Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: Hibernate (segunda parte)	En este artículo se va a continuar con el desarrollo de la aplicación Myfaces JSF con Maven multimódulo que comenzamos en un tutorial anterior. Además también se tratará de la integración de Hibernate con las aplicaciones.	2007-07-31	2633	pdf
Hibernate y el mapeo de la herencia	En este tercer tutorial de la saga vamos a ver en este tutorial como implementar las relaciones de herencia con las anotaciones de JPA	2007-06-27	2561	pdf
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.	2007-06-13	5398	pdf
Manejar dos bases de datos distintas con Hibernate	Alejandro Pérez nos enseña como manejar dos bases de datos distintas con Hibernate	2005-07-13	13875	pdf
Hibernate y Joins con la clase Criteria	En este tutorial vamos a ver como hacer "joins" entre entidades relacionadas, y las implicaciones que esto puede tener, usando Hibernate	2007-06-25	2672	pdf
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	2004-08-26	24019	pdf
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	2006-04-17	13037	pdf
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	2007-03-23	5813	pdf
Comparativa entre Hibernate y EJB3 en la Capa de Persistencia	El presente documento pretende dar algunas luces a la comparativa entre la opción de usar Hibernate y/ó EJB3 para la capa de persistencia	2007-08-16	3467	pdf
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	2007-06-25	3574	pdf

Nota:

Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento. Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores. En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo. Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador `rcanales@adictosaltrabajo.com` para su resolución.