**Universidad de Antioquia**

Ingeniería de sistemas

Arquitectura del software

2554585-1

Laboratorio #4

Inyección de dependencias e Introducción a Hibernate

Yeison Quinto Bolaño

1045504772

Carlos Mauricio Duque Restrepo

Turbo ant.

8 de octubre. de 2018

**Introducción.**

Inicial mente iniciaremos con una recopilación e investigación respecto a Reflection, el cual es fundamental para realizar inyecciones de dependencia las cuales son necesarias en este laboratorio, posterior mente indagaremos con Hibernate el cual es es un middleware utilizado para la asignación relacional de objetos y para realizar una persistencia eficiente de objetos. Para comprender la arquitectura de Hibernate, primero echemos un vistazo a cómo Hibernate encaja en el desarrollo de una aplicación, entre una aplicación y el servidor de base de datos.

**Marco teórico.**

**Reflection.**

La Reflection en Java es un concepto muy poderoso y de poca utilidad en la programación normal, pero es la columna vertebral de la mayoría de los marcos Java, J2EE. Algunos de los marcos que utilizan la Reflection de Java son:

* Spring inyección de dependencia

Realizaremos una breve explicación de acuerdo a Sprin:

La inyección de dependencia es un patrón mediante el cual se implementa la IoC, donde el control que se invierte es la configuración de las dependencias del objeto.

El acto de conectar objetos con otros objetos, o de "inyectar" objetos en otros objetos, se realiza mediante un ensamblador en lugar de hacerlo por los propios objetos.

Así es como crearía una dependencia de objeto en la programación tradicional:

public class Store {

private Item item;

public Store() {

item = new ItemImpl1();

}

}

En el ejemplo anterior, necesitamos crear una instancia de la implementación de la interfaz del elemento dentro de la clase Store .

Al utilizar DI, podemos reescribir el ejemplo sin especificar la implementación del elemento que deseamos:

public class Store {

private Item item;

public Store(Item item) {

this.item = item;

}

}

En las siguientes secciones, veremos cómo podemos proporcionar la implementación del Elemento a través de metadatos.

Tanto IoC como DI son conceptos simples, pero tienen profundas implicaciones en la forma en que estructuramos nuestros sistemas, por lo que vale la pena entenderlos bien.

* JUnit : utiliza la Reflection para analizar la anotación @Test para obtener los métodos de prueba y luego invocarla.
* Tomcat web container para reenviar la solicitud al módulo correcto al analizar sus archivos web.xml y solicitar el URI.
* Eclipse auto finalización de nombres de métodos.
* Puntales
* Hibernar

La lista es interminable y todos utilizan la Reflection de Java porque todos estos marcos no tienen conocimiento ni acceso a clases definidas por el usuario, interfaces, sus métodos, etc.

No debemos utilizar la Reflection en la programación normal, donde ya tenemos acceso a las clases e interfaces debido a los siguientes inconvenientes.

**Rendimiento deficiente**: dado que la Reflection de java resuelve los tipos de forma dinámica, implica el procesamiento como la exploración de la ruta de clase para encontrar la clase a cargar, lo que provoca un rendimiento lento.

**Restricciones de seguridad:** Reflection requiere permisos de tiempo de ejecución que pueden no estar disponibles para el sistema que se ejecuta bajo el administrador de seguridad. Esto puede hacer que su aplicación falle en tiempo de ejecución debido al administrador de seguridad.

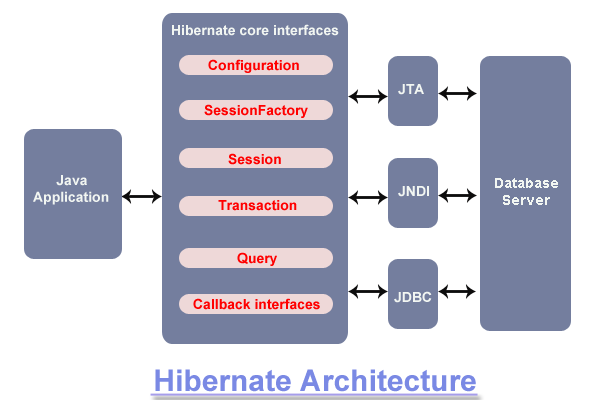
**Problemas de seguridad:** mediante la Reflection podemos acceder a una parte del código al que no debemos acceder, por ejemplo, podemos acceder a campos privados de una clase y cambiar su valor. Esto puede ser una amenaza grave para la seguridad y hacer que su aplicación se comporte de manera anormal.

**Alto mantenimiento**: el código de Reflection es difícil de entender y depurar, además, no se pueden encontrar problemas con el código en el momento de la compilación porque las clases pueden no estar disponibles, lo que hace que sea menos flexible y difícil de mantener.

**Hibernate**

Es fundamental tener encuanta los siguientes enunciados para iniciar:

* Un código de aplicación Java comprende todas las clases que definen la lógica empresarial de la aplicación. Estas clases se comunican con Hibernate.
* Luego viene Hibernate y sus interfaces centrales, mediante las cuales podríamos persistir (almacenar y recuperar) los objetos de nuestras clases de capa empresarial mediante la comunicación con el servidor de la base de datos.
* Hibernate utiliza Java Core API, Java Database Connectivity ( JDBC ), Java Transaction API ( JTA ), Java Naming and Directory Interface ( JNDI ) para comunicarse con la base de datos con el fin de mantener el estado del objeto al realizar crear, leer, actualizar, eliminar ( CRUD ) operaciones.



La arquitectura de Hibernate se compone de algunas interfaces importantes. Se explicará estas interfaces y para qué se utilizan exactamente.

**Configuración**

El objeto de configuración se utiliza para configurar de Hibernate. Este es el primer objeto que usamos cuando usamos Hibernate. Este objeto se utiliza para especificar la ubicación de los documentos de mapeo y algunas propiedades de Hibernate. Usando el objeto de configuración podemos crear un objeto SessionFactory, que finalmente se usa para crear un objeto Session para realizar las operaciones de persistencia del objeto.

Las interfaces de Hibernate que utilizan las aplicaciones para realizar las operaciones de crear, leer, actualizar, eliminar ( CRUD ) y consultar la base de datos son Session, Transaction y Query.

**Sesión:**

La interfaz de sesión es la interfaz principal de Hibernate utilizada por las aplicaciones. El objeto de sesión es liviano y puede crearse o destruirse fácilmente sin sobrecargar los recursos del sistema. El objeto de sesión no es seguro para subprocesos, por lo tanto, no debe ser utilizado por varios subprocesos a la vez. La interfaz de sesión se utiliza principalmente para realizar operaciones de persistencia como almacenar y recuperar los objetos dentro y fuera de la base de datos.

**SessionFactory:**

Para obtener el objeto Session, tenemos que usar la interfaz SessionFactory. El objeto SessionFactory es pesado y en cuanto a rendimiento, solo debe haber un objeto de SessionFactory utilizado por toda la aplicación porque se necesitan muchos recursos para crear. Un objeto SessionFactory puede ser compartido por los múltiples hilos.

**Transacción**

La interfaz de transacción le permite realizar todas las operaciones en el objeto persistente dentro de una sola transacción. Durante la ejecución de esta transacción única, cualquier error entre el inicio y el final de la transacción lleva a la cancelación de la transacción completa y a devolver al objeto persistente su estado original (antes del inicio de la transacción), lo que ayuda a mantener la integridad de la base de datos.

**Consulta**

La interfaz de consultas permite consultar la base de datos y también le permite controlar cómo se ejecutarán estas consultas. En Hibernate, las consultas se escriben utilizando Hibernate Query Language (HQL) o SQL. La interfaz de consulta vincula los parámetros enviados dentro de una consulta, controla el número de resultados devueltos por la ejecución de la consulta y ejecuta la consulta.

**Interfaces de devolución de llamada**

Interfaces de Hibernate que permiten que una aplicación responda a los eventos realizados por Hibernate, eventos como guardar, recuperar, actualizar, eliminar el objeto. Estas interfaces son Inteceptor, Lifecycle y Validatable y juntas también se conocen como interfaces de devolución de llamada.

**Conclusión**

Se pudo investigar, asimilar, aplicar y aprender respecto a Reflection e Hibernate su usabilidad características principales sus ventajas como implementar y aspectos a tener en cuenta a la hora de realizar aplicaciones que implemente esta tecnología, de igual forma fueron de gran utilidad en el desarrollo del Laboratorio #4.

Repositorio de laboratorio desarrollado

**https://github.com/Yeison-Quinto/laboratorio-4**

**Bibliografía**

* Oracle, 2018, **Using Hibernate in a Web Application - NetBeans IDE Tutorial** <https://netbeans.org/kb/docs/web/hibernate-webapp.html>
* Simplidigital, Publicado el 7 nov. 2017**, What is hibernate? | Hibernate tutorial** <https://www.youtube.com/watch?v=SjVFviJKpwM>
* Subrato Halder, 2015, **Software Engineer at Tech Mahindra** <https://www.quora.com/What-is-hibernate-in-java>
* Irene Puebla Hernández y Antonio García Dopico**, Arquitectura Y Diseño De Un Sistema De Gestión De Valores De Bolsa**, Universidad Politécnica De Madrid , España (Madrid), <http://oa.upm.es/1697/1/PFC_IRENE_PUEBLA_HERNANDEZ.pdf>
* journaldev, abril 4 2018 **Java Reflection Example** Tutorial <https://www.journaldev.com/1789/java-reflection-example-tutorial>
* <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>
* What is reflection and why is it useful?

https://stackoverflow.com/questions/37628/what-is-reflection-and-why-is-it-useful