**HIBERNATE LAZY LOADING** or **HIBERNATE LAZY FETCHING**

* HIBERNATE LAZY FETCHING means to decide whether to load child objects while loading the Parent Object.
* A setting need to be done in respective HIBERNATE mapping file of the parent class.
* **Lazy = true (means not to load child)**
* By default, the LAZY LOADING of the child objects is true.
* This make sure that the child objects are not loaded unless they are explicitly invoked by calling getChild() method on parent.

Example :

Parent Class : Employee class

Child class : Address Class

public class Employee {  
                        private Set address = new HashSet();  
                        public Set getAddress () {  
                                    return address;  
                        }  
                        public vod setAddresss(Set address) {  
                                    this. address = address;  
                        }

}

**Employee.hbm.xml**:

<set name="address" inverse="true" cascade="delete" lazy="false">  
                        <key column="a\_id" />  
                        <one-to-many class="beans Address"/>

</set>

* In the above configuration, If lazy="false":
* When you load the Employee object, the child object Address is also loaded and set to setAddresss() method.
* If you call getAddresss(), then the Address object is loaded without fresh database call.

* In the above configuration, If lazy=“true" :
* When you load the Employee object, the child object Address is not.
* If you call getAddresss(), then the Address object is loaded from Database.

***lazyinitializationexception***

The *org.hibernate.LazyInitializationException* is the sub class of the *org.hibernate.HibernateException.*

The *lazyinitializationexception*exception occurs when you try to access unfetched data or non-initialized association after the session is closed. For example, if session is closed and uninitialized collection is accessed, *lazyinitializationexception*occurs .

Why does LazyInitializationException happen?

The Person class has a Dog list. The easier and fattest way to display a person data would be, to use the entityManager.find() method and iterate over the collection in the page (xhtml).

Notice that in the code above, all we want to do is to find a person in the database and display its dogs to an user. If you try to access the page with the code above you will see the exception bellow:

**To understand better this error let us see how the JPA/Hibernate handles the relationship.**

**Every time we do a query in the database the JPA will bring to all information of that class. The exception to this rule is when we talk about list (collection). Image that we have an announcement object with a list of 70,000 of emails that will receive this announcement. If you want just to display the announcement name to the user in the screen, imagine the work that the JPA would have if the 70,000 emails were loaded with the name.**

**The JPA created a technology named Lazy Loading to the classes attributes. We could define Lazy Loading by: “the desired information will be loaded (from database) only when it is needed”.**

**Notice in the above code, that the database query will return a Person object. When you access the lazyDogs collection, the container will notice that the lazyDogs collection is a lazy attribute and it will “ask” the JPA to load this collection from the database.**

**In the moment of the query (*that will bring the lazyDogs collection*) execution, an exception will happen. When the JPA/Hibernate tries to access the database to get this lazy information, the JPA will notice that there is no opened collection. That is why the exception happens, the lack of an opened database connection.**

**Every relationship that finishes with @Many will be lazy loaded by default: @OneToMany and @ManyToMany. Every relationship that finishes with @One will be eagerly loaded by default: @ManyToOne and @OneToOne. If you want to set a basic field (E.g. String name) with lazy loading just do: @Basic(fetch=FetchType.LAZY).**

**Every basic field (E.g. String, int, double) that we can find inside a class will be eagerly loaded if the developer do not set it as lazy.**

**A curious subject about default values is that you may find each JPA implementation (EclipseLink, Hibernate, OpenJPA) with a different behavior for the same annotation. We will talk about this later on.**

**Load collection by annotation**

**The easier and the fattest way to bring a lazy list when the object is loaded is by annotation. But *this will not be the best approach always*.**

**In the code bellow we will se how to eagerly load a collection by annotation:**

|  |  |
| --- | --- |
| 1 | <b>@OneToMany(fetch = FetchType.EAGER) |
| 2 | @JoinTable(name = 'person\_has\_eager\_dogs') | |

|  |  |
| --- | --- |
| 3 | private List<Dog> eagerDogs;</b> |
| 1 | <b><h:dataTable var='dog' value='#{dataMB.person.eagerDogs}'> | |

|  |  |
| --- | --- |
| 2 | <h:column> |
| 3 | <f:facet name='header'> | |

|  |  |
| --- | --- |
| 4 | Dog name |
| 5 | </f:facet> | |

|  |  |  |
| --- | --- | --- |
| 6 | #{dog.name} | |
| 7 | </h:column> |

|  |  |
| --- | --- |
| 8 | </h:dataTable></b> |

**Pros and Cons of this approach:**

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Easy to set up | If the class has several collections this will not be good to the server performance |
| The list will always come with the loaded object | If you want to display only a basic class attribute like name or age, all collections configured as EAGER will be loaded with the name and the age |

**This approach will be a good alternative if the EAGER collection have only a few items. If the Person will only have 2, 3 dogs your system will be able to handle it very easily. If later the Persons dogs collection starts do grow a lot, it will not be good to the server performance.**