**Difference bewteen session.get and session.load in Hibernate**

In Hibernate, to get the the information of a specific record, we have two methods **get()** and **load()**, both defined in**Session** interface. Then what is the difference between them and when to use which one. This tutorial throws some light over it.

The explanation is more linked to **eager** and **lazy intializations**.

**Object obj = s.get(Student.class, new Integer(100)); // eager initialization  
Object obj1 = s.load(Student.class, new Integer(101)); // lazy initialization**

The above statements show the usage of **get()**and **load()** methods. A complete program on these methods is available at [Hibernate – get, load, delete and Update Methods](http://way2java.com/hibernate/hibernate-get-load-delete-and-update-methods/).

The two methods are defined in**org.hibernate.Session** interface. First let us see the signature of these methods as defined in Hibernate API.

**Object get(Class class, Serializable id) throws HibernateException**

Return the persistent instance of the given entity class with the given identifier, or **null** if there is no such persistent instance. (If the instance, or a proxy for the instance, is already associated with the session, return that instance or proxy.)

**Object load(Class class, Serializable id) throws HibernateException**

Returns the persistent instance of the given entity class with the given identifier, assuming that the instance exists.

You should not use this method to determine if an instance exists (use **get()** instead). Use this only to retrieve an instance that you assume exists, where non-existence would be an actual error.

Both methods are used to retrieve only one Java object (record) from the database (for multiple records, we have, [list() and iterate()](http://way2java.com/hibernate/hibernate-list-and-iterate-lazy-and-eager-initialization/) methods).

**get()**

The **get()** method returns the Java object from the database as an object of**Object** class and is required to type cast as follows.

|  |
| --- |
| Object obj = s.get(Student.**class**, **new** Integer(1)); *// where sid is 1*  Student s1 = (Student) obj; |

We can club the above two statements as follows as one statement.

|  |
| --- |
| Student s1 = (Student) s.get(Student.**class**, **new** Integer(1)); |

If the record on the primary key **1**exists, the **s1** contains all the column values in the form of object **s1**. If the record on **id** **1** does not exist, the**s1** contains **null** value.

Now it is your job to retrieve the values from **s1** object as follows using getter methods.

|  |
| --- |
| **int** id = s1.getSid();  String name = s1.getSname();  **double** marks = s1.getSMarks();  Date joindate = s1.getSjoindate(); |

All the above is fine and actually what is happening when **get()** method is called. Hibernate creates a SQL statement as follows and executes.

|  |
| --- |
| select \* from school where stud\_id=1; |

This is called as **eager fetching**. That is, a call goes to the database to fetch the record.

**load()**

|  |
| --- |
| Student s1 = (Student) s.load(Student.**class**, **new** Integer(1)); |

In the above statement, **s1** contains a **proxy** of **Student** object and not the actual Student object.

**1. What is proxy?**

**Proxy** is a subclass of **Student** class, like **StudentImpl**, implemented internally by Hibernate. **s1** belongs to such proxy class.

**2. Why Hibernate returns a proxy and not actual object? How this is advantageous?**

Proxy contains a reference of the primary key **id** of the actual object. **load()** is best used to **delete** a record or **update** a record. When deletion is required, why the object **s1** should contain all the record data which may spread number of columns? To fetch the record, a call should go to the database. When a method call is avoided the performance increases.

**3. When the proxy object s1 is converted into a real object of Student?**

When you call **s1.getName()** or **s1.getMarks()** etc. the object **s1** becomes an actual **Student** object and now database hit is made. This is called as lazy fetching.

**4. Why all this round about?**

It is very important to know. The whole magic of Hibernate lies here only. **s1**proxy object is returned even when the student record by **id** 1 does not exist also. That is, **s1** is never null (incase of get(), **s1** can be null also).

Infact, the object **s1** returned by the load() method is used for deletion or updation of a record as follows.

**s.delete(s1); or s.update(s1); // if id 1 does not exist, exception is raised**

When the deletion operation is to be done there is no necessity that **s1** should contain the data of all the columns. **load()** increases the performance for delete and update operations as **load()** does not hit the database (and hits only when deletion or updation is done). Really, if data is required, **get()** is preferred.

**Special cases**

If **get(Student.class, new Integer(1))** is called first and later again **load(Student.class, new Integer(1))** is called and if the record with primary key **1** exists, the **s1** in the both the cases contains a real object. Why? We know **get()** method returns a Student object and **load()** returns **s** proxy object. With **get()**, a real object exists in the cache memory, the same object is returned to **load()** also.

If the reverse happens? If **load()** is called first and then **get()**? In both cases, **s1** contains a proxy object because first **load()** returns a proxy and remains in the cache memory.

To put it simple:

|  |  |  |
| --- | --- | --- |
|  | **get( )** | **load( )** |
| **1** | Return value null is possible | Never returns null |
| **2** | Fast if record exists | Slow if record exists |
| **3** | Used to retrieve object (record) | Used for delete etc. operations |
| **4** | Eager fetching | Lazy fetching |
| **5** | Always hits the database | Not always hits |
| **6** | Does not return proxy object | Always returns a proxy object |
| **7** | Performance-wise is slow as it may have to make number of rounds to database to get data | Better performance. If already in the cache, the record is available to much difference |
| **8** | As it returns null if no record exist, the execution continues | It throws ObjectNotFoundException, if record not found. Execution terminates if not handled successfully |

For the differences between **list()** and **iterate()**refer "[Difference between Query.list() and Query.iterate() Hibernate](http://way2java.com/hibernate/difference-between-query-list-and-query-iterate-hibernate/)".

**More about load() method**

**load()** method is overloaded five times in Session interface. One overloaded method is given down for extra discussion.

***Object load(Class theClass, Serializable id, LockMode lockmode) throws HibernateException***

Return the persistent instance of the given entity class with the given identifier, obtaining the specified lock mode, assuming the instance exists.

Example:

|  |
| --- |
| Course c1 = session.load(Course.**class**, **new** Integer(100), LockMode.UPGRADE); |

Now the object**c1**is locked so that other programmers cannot modify the object **c1** until all **c1** operations are done like calling **c1.setDuration(50)** method etc as follows.

|  |
| --- |
| c1.setDuration(50);  session.flush();  tx.commit(); |

Other lock modes: **WRITE**, **NONE**, **READ**, **UPGRADE\_NOWAIT**.