**Different between session.get() and session.load()**

Often times, you will notice Hibernate developers mix use of **session.get()** and **session load()**, do you wonder what’s the different and when you should use either of it?

**Different between session.get() and session.load()**

Actually, both functions are use to retrieve an object with different mechanism, of course.

**1. session.load()**

* It will always return a “**proxy**” (Hibernate term) without hitting the database. In Hibernate, proxy is an object with the given identifier value, its properties are not initialized yet, it just look like a temporary fake object.
* If no row found , it will throws an **ObjectNotFoundException**.

## 2. session.get()

* It always **hit the database** and return the real object, an object that represent the database row, not proxy.
* If no row found , it return **null**.

## It’s about performance

Hibernate create anything for some reasons, when you do the association, it’s normal to obtain retrieve an object (persistent instance) from database and assign it as a reference to another object, just to maintain the relationship. Let’s go through some examples to understand in what situation you should use **session.load()**.

## 1. session.get()

For example, in a Stock application , Stock and StockTransactions should have a “one-to-many” relationship, when you want to save a stock transaction, it’s common to declared something like below

Stock stock = (Stock)session.get(Stock.**class**, **new** Integer(2));

StockTransaction stockTransactions = **new** StockTransaction();

//set stockTransactions detail

stockTransactions.setStock(stock);

session.save(stockTransactions);

**Output**

Hibernate: select ... from stock stock0\_ where stock0\_.STOCK\_ID=?

Hibernate: insert into stock\_transaction (...) values (?, ?, ?, ?, ?, ?)

In session.get(), Hibernate will hit the database to retrieve the Stock object and put it as a reference to StockTransaction. However, this save process is extremely high demand, there may be thousand or million transactions per hour, do you think is this necessary to hit the database to retrieve the Stock object everything save a stock transaction record? After all you just need the Stock’s Id as a reference to StockTransaction.

**2. session.load()**

In above scenario, **session.load()** will be your good solution, let’s see the example,

Stock stock = (Stock)session.load(Stock.**class**, **new** Integer(2));

StockTransaction stockTransactions = **new** StockTransaction();

//set stockTransactions detail

stockTransactions.setStock(stock);

session.save(stockTransactions);

## Output

Hibernate: insert into stock\_transaction (...) values (?, ?, ?, ?, ?, ?)

In session.load(), Hibernate will not hit the database (no select statement in output) to retrieve the Stock object, it will return a Stock proxy object – a fake object with given identify value. In this scenario, a proxy object is enough for to save a stock transaction record.

**Exception**

In exception case, see the examples

**session.load()**

Java

Stock stock = (Stock)session.load(Stock.class, new Integer(100)); //proxy

//initialize proxy, no row for id 100, throw ObjectNotFoundException

System.out.println(stock.getStockCode());

It will always return a proxy object with the given identity value, even the identity value is not exists in database. However, when you try to initialize a proxy by retrieve it’s properties from database, it will hit the database with select statement. If no row is found, a **ObjectNotFoundException**will throw.

org.hibernate.ObjectNotFoundException: No row with the given identifier exists:

[Stock#100]

## session.get()

//return null if not found

Stock stock = (Stock)session.get(Stock.**class**, **new** Integer(100));

System.out.println(stock.getStockCode()); //java.lang.NullPointerException

It will always return null , if the identity value is not found in database.

**Conclusion**

There are no always correct solution, you have to understand the differential in between, and decide which method is best fix in your application.