Hibernate – Fetching Strategies Examples

Posted on February 21, 2010 , Last modified : January 29, 2010
By mkyong

Hibernate has few fetching strategies to optimize the Hibernate generated select statement, so that it can be as efficient as possible. The fetching strategy is declared in the mapping relationship to define how Hibernate fetch its related collections and entities.

Fetching Strategies

There are four fetching strategies

1. fetch-"join" = Disable the lazy loading, always load all the collections and entities.
2. fetch-"select" (default) = Lazy load all the collections and entities.
3. batch-size="N" = Fetching up to 'N' collections or entities, *Not record*.
4. fetch-"subselect" = Group its collection into a sub select statement.

For detail explanation, you can check on the Hibernate documentation.

Fetching strategies examples

Here's a "one-to-many relationship" example for the fetching strategies demonstration. A stock is belong to many stock daily records.

Example to declare fetch strategies in XML file

```
...
<hibernate-mapping>
    <class name="com.mkyong.common.Stock" table="stock">
        <set name="stockDailyRecords"  cascade="all" inverse="true"
            table="stock_daily_record" batch-size="10" fetch="select">
            <key>
                <column name="STOCK_ID" not-null="true" />
            </key>
            <one-to-many class="com.mkyong.common.StockDailyRecord" />
        </set>
    </class>
</hibernate-mapping>
```

Example to declare fetch strategies in annotation

```
...
@Entity
@Table(name = "stock", catalog = "mkyong")
public class Stock implements Serializable{
...
        @OneToMany(fetch = FetchType.LAZY, mappedBy = "stock")
        @Cascade(CascadeType.ALL)
        @Fetch(FetchMode.SELECT)
        @BatchSize(size = 10)
        public Set<StockDailyRecord> getStockDailyRecords() {
                return this.stockDailyRecords;
        }
...
}
```

Let explore how fetch strategies affect the Hibernate generated SQL statement.

1. fetch="select" or @Fetch(FetchMode.SELECT)

Recent Posts

Java - Cron job to run a jar file
Gradle - How to skip unit test
Gradle - Spring MVC Web Project Example
How to configure hot deploy in Eclips
Gradle - Create a Jar file with dependencies

Popular Tutorials

Android Tutorial
JSF 2.0 Tutorial
Spring Tutorial
Maven Tutorial
Hibernate Tutorial

This is the default fetching strategy. it enabled the lazy loading of all it's related collections. Let see the example…

```
//call select from stock
Stock stock = (Stock)session.get(Stock.class, 114);
Set sets = stock.getStockDailyRecords();

//call select from stock_daily_record
for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
      StockDailyRecord sdr = (StockDailyRecord) iter.next();
      System.out.println(sdr.getDailyRecordId());
      System.out.println(sdr.getDate());
}
```

Output

```
Hibernate:
    select ...from mkyong.stock
    where stock0_.STOCK_ID=?

Hibernate:
    select ...from mkyong.stock_daily_record
    where stockdaily0_.STOCK_ID=?
```

Hibernate generated two select statements

1. Select statement to retrieve the Stock records -**session.get(Stock.class, 114)**
2. Select its related collections – **sets.iterator()**

2. fetch="join" or @Fetch(FetchMode.JOIN)

The "join" fetching strategy will disabled the lazy loading of all it's related collections. Let see the example…

```
//call select from stock and stock_daily_record
Stock stock = (Stock)session.get(Stock.class, 114);
Set sets = stock.getStockDailyRecords();

//no extra select
for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
      StockDailyRecord sdr = (StockDailyRecord) iter.next();
      System.out.println(sdr.getDailyRecordId());
      System.out.println(sdr.getDate());
}
```

Output

```
Hibernate:
    select ...
    from
        mkyong.stock stock0_
    left outer join
        mkyong.stock_daily_record stockdaily1_
            on stock0_.STOCK_ID=stockdaily1_.STOCK_ID
    where
        stock0_.STOCK_ID=?
```

Hibernate generated only one select statement, it retrieve all its related collections when the Stock is initialized. -**session.get(Stock.class, 114)**

1. Select statement to retrieve the Stock records and outer join its related collections.

3. batch-size="10" or @BatchSize(size = 10)

This 'batch size' fetching strategy is always misunderstanding by many Hibernate developers. Let see the *misunderstand* concept here…

```
Stock stock = (Stock)session.get(Stock.class, 114);
Set sets = stock.getStockDailyRecords();

for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
      StockDailyRecord sdr = (StockDailyRecord) iter.next();
      System.out.println(sdr.getDailyRecordId());
```

```
        System.out.println(sdr.getDate());
}
```

What is your expected result, is this per-fetch 10 records from collection? See the output
Output

```
Hibernate:
    select ...from mkyong.stock
    where stock0_.STOCK_ID=?

Hibernate:
    select ...from mkyong.stock_daily_record
    where stockdaily0_.STOCK_ID=?
```

The batch-size did nothing here, it is not how batch-size work. See this statement.

> The batch-size fetching strategy is not define how many records inside in the collections are loaded. Instead, it defines how many collections should be loaded.
> — Repeat N times until you remember this statement —

Another example
Let see another example, you want to print out all the stock records and its related stock daily records (collections) one by one.

```
List<Stock> list = session.createQuery("from Stock").list();

for(Stock stock : list){

    Set sets = stock.getStockDailyRecords();

    for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
            StockDailyRecord sdr = (StockDailyRecord) iter.next();
            System.out.println(sdr.getDailyRecordId());
            System.out.println(sdr.getDate());
    }
}
```

No batch-size fetching strategy
Output

```
Hibernate:
    select ...
    from mkyong.stock stock0_

Hibernate:
    select ...
    from mkyong.stock_daily_record stockdaily0_
    where stockdaily0_.STOCK_ID=?

Hibernate:
    select ...
    from mkyong.stock_daily_record stockdaily0_
    where stockdaily0_.STOCK_ID=?

Keep repeat the select statements....depend how many stock records in your table.
```

If you have 20 stock records in the database, the Hibernate's default fetching strategies will generate 20+1 select statements and hit the database.

1. Select statement to retrieve all the Stock records.
2. Select its related collection
3. Select its related collection
4. Select its related collection
….
21. Select its related collection

The generated queries are not efficient and caused a serious performance issue.

Enabled the batch-size='10' fetching strategy
Let see another example with batch-size='10' is enabled.
Output

```
Hibernate:
```

```
    select ...
    from mkyong.stock stock0_

Hibernate:
    select ...
    from mkyong.stock_daily_record stockdaily0_
    where
        stockdaily0_.STOCK_ID in (
            ?, ?, ?, ?, ?, ?, ?, ?, ?, ?
        )
```

Now, Hibernate will per-fetch the collections, with a select *in* statement. If you have 20 stock records, it will generate 3 select statements.

1. Select statement to retrieve all the Stock records.
2. Select In statement to per-fetch its related collections (10 collections a time)
3. Select In statement to per-fetch its related collections (next 10 collections a time)

With batch-size enabled, it simplify the select statements from 21 select statements to 3 select statements.

4. fetch="subselect" or @Fetch(FetchMode.SUBSELECT)

This fetching strategy is enable all its related collection in a sub select statement. Let see the same query again..

```
List<Stock> list = session.createQuery("from Stock").list();

for(Stock stock : list){

    Set sets = stock.getStockDailyRecords();

    for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
            StockDailyRecord sdr = (StockDailyRecord) iter.next();
            System.out.println(sdr.getDailyRecordId());
            System.out.println(sdr.getDate());
    }
}
```

Output

```
Hibernate:
    select ...
    from mkyong.stock stock0_

Hibernate:
    select ...
    from
        mkyong.stock_daily_record stockdaily0_
    where
        stockdaily0_.STOCK_ID in (
            select
                stock0_.STOCK_ID
            from
                mkyong.stock stock0_
        )
```

With "subselect" enabled, it will create two select statements.

1. Select statement to retrieve all the Stock records.
2. Select all its related collections in a sub select query.

Conclusion

The fetching strategies are highly flexible and a very important tweak to optimize the Hibernate query, but if you used it in a wrong place, it will be a total disaster.

Reference

1. http://docs.jboss.org/hibernate/core/3.3/reference/en/html/performance.html
2. https://www.hibernate.org/315.html

Tags : hibernate