Java-Success.com

Industrial strength Java Career Companion

Register | Login | Logout | Contact Us

search here … Go

Home | Java FAQs | 600+ Java Q&As | Career | Tutorials | Member | Why?

Can u Debug? | Java 8 ready? | Top X | Productivity Tools | Judging Experience?

Home › Interview › Spring, Hibernate, & Maven Interview Q&A › Hibernate › 02: Understanding Hibernate proxy objects and avoiding potential pitfalls

02: Understanding Hibernate proxy objects and avoiding potential pitfalls

Posted on December 24, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q. How does hibernate support lazy loading?
A. Hibernate uses a proxy object to support lazy loading. Basically as soon as you reference a child or lookup object via the accessor/getter methods, if the linked entity is not in the session cache (i.e. the first-level cache), then the proxy code will go off to the database and load the linked object. It uses javassist (or CGLIB) to effectively and dynamically generate sub-classed implementations of your objects.

Let's look at an example. An employee hierarchy table can be represented in a database table as shown below

```
public class Employee {

   private Long id;
```

```
    private String name;
    private String title;
    private Employee superior;
    private Set<Employee> subordinates;

    //getters and setters are omitted

}


```

In the above example, if you use lazy loading then the "superior" and "subordinates" will be proxied (i.e. not the actual object, but the stub object that knows how to load the actual object) when the main "Employee" object is loaded. So, if you need to get the "subordinates" or "superior" object, you invoke the getter method on the employee like *employee.getSuperior( )* and the actual object will be loaded.

Q. What are some of the pitfalls of using a proxy object?
A. The typical pitfall is in how you implement your ***equals( )*** method that gets invoked when comparing objects.

Pitfall 1: As explained before, **the proxy objects are dynamically created by sub-classing your object** at runtime. The subclass will have all the methods of the parent, but the fields (e.g. name, etc) in the proxy object will remain null, and when any of the methods are accessed via getter/setter method, the proxy loads up the real object from the database.

A typical *equals( )* method implementation will look like

```
   @Override
   public boolean equals(Object obj) {
     if (obj == null) {
       return false;
     }
     if (obj == this) {
       return true;
     }
     if (!(obj instanceof Employee)) {           //
       return false;
     }
     return name.equals((Employee)obj).name);   //
   }


```

16 Technical Key Areas

80+ step by step Java Tutorials

As discussed before, the supplied object is a proxy object and the supplied name will be null. This can be fixed by using the getter method in **Line Y** instead of using the field directly. Using the getter method will tell the proxy object to load the actual object as shown below.

```
@Override
public boolean equals(Object obj) {
  if (obj == null) {
    return false;
  }
  if (obj == this) {
    return true;
  }
  if (!(obj instanceof Employee)) {
    return false;
  }
  return name.equals((Employee)obj).getName())
}


```

Pitfall 2: We saw earlier that the **the proxy objects are dynamically created by sub-classing your object.** In a simple scenario where you only have the "*Employee*" object the **typecasting** in Line Y and "**instanceof**" operator in Line X will work. But, what will happen if you have a type hierarchy for The class *Employee* as shown below

```
public class PermanentEmployee extends Employe
    .......
}

public class CasualEmployee extends Employee {
    .......
}


```

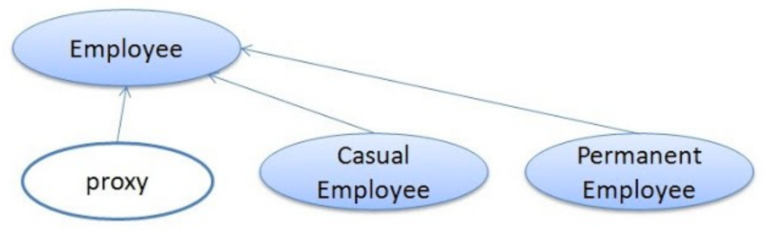
When you have a type hierarchy as shown above, the **type casts** and **instanceof** operators will not work with the proxy objects.

A proxy class is a subclass of a field type that is required. Hibernate creates a dynamic subclass by looking at the type of field. This means that if the field type is not the actual implementation (e.g. CasualEmployee) type, but an interface

100+ Java pre-interview coding tests

How good are your .....?

or superclass (e.g. Employee), the type of the proxy will be different than the type of the actual object. So, as shown in the diagram below, If the field type is the superclass (e.g. Employee) of the actual implementation (i.e. CasualEmployee), the proxy type (i.e. proxy) and the implementation-type (e.g. CasualEmployee) will be siblings in the type hierarchy, both extending the superclass. This means the proxy object will not be an instance of the implementing type (i.e. CasualEmployee), and the application code depending on this check will fail.



Hibernate proxy

To prevent this issue, you have two approaches.

Approach 1: Switch off proxying on the top level class by setting lazy=”false”, which will turn proxying off for the hierarchy.

Approach 2: Use the “Gang of Four” (i.e. GoF) **visitor design pattern** that allows you to adapt a single object or manipulate a collection of polymorphic objects without all the messy typecasts and instanceof operations.

Pitfall 3: As per the above example, if you have an “Employee” class, that contains a “name” property, when you invoke do “employee.getName()”, the proxies will get the “name” from Hibernate caches (either 1st or 2nd levels) or the database when requested. But if this call happens in the presentation layer like in the Struts action class, you will get the org.hibernate.LazyInitializationException because the **Hibernate Session is closed and this lazy attribute does not have the session attached**, hence can't load their lazy references.

The solution is to de-proxy the employee class as shown below:

Step 1: Write a generic utility class to **de-proxy** a given object

```
public class HibernateUtil {

    public static <T> T unproxy(T entity) {
        if (entity == null) {
            return null;
        }

        if (entity instanceof HibernateProxy) {
            Hibernate.initialize(entity);
            entity = (T) ((HibernateProxy) entit
        }

        return entity;
    }
}

```

Step 2: Use the above utility class

```
public Employee getSuperior() {
    superior = HibernateUtils.unproxy(employee);
    return superior;
}

```

These types of issues are hard to debug, and being aware of these pitfalls can save you lots of time in debugging and fixing the issues.

Popular Member Posts

- ♦ 11 Spring boot interview questions & answers

  904 views
- ♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

  816 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

427 views

18 Java scenarios based interview Questions and Answers

408 views

♦ 7 Java debugging interview questions & answers

323 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

311 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

303 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

301 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

251 views

001B: ♦ Java architecture & design concepts interview questions & answers

209 views

0 Like Share Tweet submit reddit 0 G+1 Share

Bio **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Are you ready for digital recruitment? Can you be found in Google search?

Working with .properties config files in Java ›

Posted in Hibernate, member-paid

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from "Blah blah" to "Wow"? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer