UsingJNDIToObtainHiveMindServices

Q. Using JNDI To Obtain HiveMind Services

(DanielFeist 24/06/2004)

I have been experimenting with the use of JNDI as a facade to HiveMind by implementing a simple JNDI SPI. The lookup method of a JNDI Context takes just a Name, yet to obtain a HiveMind service i need to be able to specifiy the interface expected. What i have found is that i can, in place of the interface expected put Object.class and everything works ok. Object is not an interface so i'm not quite sure why this works. Is this intentional? Will HiveMind continue to support this?

A.

HowardLewisShip: Yes, this will continue work. The idea of passing in the expected (assignable) type is to allow HiveMind to do a check that the service object or proxy returned is assignable. Better a good message from inside HiveMind than a bad ClassCastException. Using `java.lang.Object` is acceptible if you don't care about that test.

I think it's a very good idea, where practical, to do this extra step ... it supports the *Feedback* principle. This is insurance against a change to a 3rd party library where an existing service's interface is changed (a bad idea!). Your existing code will fail ... but fail with a more sophisticated message. This is why I would object to a convienience method that takes just a service name.

DanielFeist: I agree completly and don't think the addition of another method which takes just the service-id is a good idea. I just wanted to know if the passing of `Object.class` is acceptable if there is a situation, like in the example I gave or similair, where it is not possible to pass the interface expected.

Discussion

I believe it is possible to infer the class name from the lookup request. Here is an example of using JNDI to discover the ThreadLocalStorage service.

```
Context c = new InitialContext();
ThreadLocalStorage tls = (ThreadLocalStorage) c.lookup("service:" + ThreadLocalStorage.class.getName());
```

If there was a distinct service id, then that could be managed as well.

```
Context c = new InitialContext();
ThreadLocalStorage tls = (ThreadLocalStorage) c.lookup("service:" + ThreadLocalStorage.class.getName() + "!some.
service.id");
```

The internals of the lookup method could tokenize the string and do a forName on the class portion of the string.

```
public class serviceURLContext implements Context {
  private Registry _registry;

  public Object lookup(String name) throws NamingException {
    name = name.substring(8).trim();

    String serviceClassName = null;
    String serviceId = null;

    int i = name.indexOf("!");
    if (i > 0) {
      serviceClassName = name.substring(0, i);
      serviceId = name.substring(i + 1);
    } else {
      serviceClassName = name;
      serviceId = name;
    }

    Class serviceClass = Thread.currentThread().getContextClassLoader().
      loadClass(serviceClassName);

    return _registry.getService(serviceId, serviceClass);
  }
}
```

– NareshSikha - 30 Jun 04