Using Dynamic Proxies

Hermod Opstvedt
Chief Architect
DnB NOR ITU
Dynamic Proxies

- What are Dynamic proxies?
Dynamic Proxies

Think of dynamic proxies as a chameleon
Dynamic Proxies

- They take on the appearance of the background.
  - In Java terms this means that they exhibit all interfaces of the object for which they are proxies.
Dynamic Proxies

- Proxy - from Webster:

  - Etymology: Middle English *procucie*, contraction of *procuracie*, from Anglo-French, from Medieval Latin *procuratia*, alteration of Latin *procuratio* procuration
  1: the agency, function, or office of a deputy who acts as a substitute for another
  2a: authority or power to act for another 2b: a document giving such authority; specifically: a power of attorney authorizing a specified person to vote corporate stock
  3: a person authorized to act for another: **PROCURATOR**
  - *proxy* adjective
Dynamic Proxies

- Are an implementation of the delegator pattern. Also can be associated with the Adaptor pattern and the Decorator pattern.
- First introduced in JDK 1.3
- Improvements done in JDK 1.4 & 1.5
  - First & foremost, speed
Dynamic Proxies

**Dynamic Proxy**

- A dynamic proxy class is a class that implements a list of interfaces specified at runtime on an instance.
  - Must implement java.lang.reflect.InvocationHandler
  - Proxy classes are public, final and non-abstract.
  - No two elements in the interfaces array may refer to identical Class objects.
Dynamic Proxies

- No two interfaces may each have a method with the same name and parameter signature but different return type.
- The unqualified name of a proxy class is unspecified.
  - This has security implications in J2EE environment: The java.securityProtectionDomain of a proxy class is the same as that of system classes loaded by the bootstrap class loader, such as java.lang.Object, because the code for a proxy class is generated by trusted system code. This protection domain will typically be granted java.security.AllPermission.

- Exception to the rule: If a proxy class implements a non-public interface, then it will be defined in the same package as that interface.
Dynamic Proxies

- All non-public interfaces must be in the same package.
- The public, non-final methods of `java.lang.Object` logically precede all of the proxy interfaces for the determination of which Method object to pass to the invocation handler.
Dynamic Proxies

- Limitation

- A dynamic proxy can only be created for instances that implement an interface. You cannot create a dynamic proxy for an instance whose static type is a class.
Dynamic Proxies

- So this class cannot be proxyfied

```java
public class NoGo
{
    public NoGo()
    {
    }

    public String doSomething()
    {
        return "Done something";
    }
}
```
Dynamic Proxies

- But this can

```java
public class WillWork implements IWillWork {
    public WillWork() {
    }

    public String doSomething() {
        return "Done something";
    }
}
```
Dynamic Proxies

- Ordering

The ordering of interfaces is important!

- If you have methods with the same name and signature in more than one interface, the method from the first interface in the list is the one that is used to create the proxy.
Dynamic Proxies

Why should you use dynamic proxies?

- You want to do things with a class without touching the class itself
  - Common functionality:
    - Timing
    - Logging
    - Validation
    - Authentication
    - Interruptible calls – *i.e.* running in separate threads
    - *etc.*
Dynamic Proxies

➢ For instance timing – How do you time how long it takes to do method calls on a class?
  • Do it from where you call your class(es)?
    ✓ Means you have to write a lot of code in your calling classes that you really do not want there.
    ✓ You have to find all the places that you call your class(es) and write the timing code.
    ✓ In short, a lot of work
Dynamic Proxies

- Create a superclass that they inherit from?
  - Not always possible – Single inheritance in Java.
  - Means refactoring/rewriting all your classes.

- Create proxy(decorator) classes?
  - Means writing a new class for each and every one of your classes that you want common functionality for.
  - You get into a maintenance nightmare.
Dynamic Proxies

Or create a dynamic proxy?

- Means writing one new class with the desired common functionality.
- Means extracting an interface out of your class(es).
- Means rewriting where you instantiate your class(es).
  or
  - Use some sort of Factory pattern to instantiate your class(es)
  or
  - Create a new Classloader that does it.
**Dynamic Proxies**

- **Standard method invocation**

```
Client -------------------------> Called class
      doSomething
```
Dynamic Proxies

- Proxyfied method invocation

![Diagram of proxy mechanism]

- Client
- Proxy
- Called class

Client calls `doSomething` through the proxy, which then calls `doSomething` on the called class.
Dynamic Proxies

- The key factor in dynamic proxies is the fact that they are invocation handlers.
  - The method that you as a dynamic proxies writer need to implement is the invoke() method.
  - This is where all the magic happens.
  - This is where you insert whatever you want to do prior to or after you call the method on the proxyfied object.
Dynamic Proxies

Client → Proxy → Object

do some stuff before

do some stuff after
Dynamic Proxies

- Instantiating a Proxy
  - Do a call to the static method `newProxyInstance` on `Proxy`
    - Takes three parameters
      - `loader` – the `ClassLoader`
      - `interfaces` – `Class[]`, an array of interface classes
      - `h` – `InvocationHandler`, your implementation of the `InvocationHandler` interface
Dynamic Proxies

- Implementing InvocationHandler.
  - Create a class that implements InvocationHandler
  - Create a newInstance method so you get the singleton behavior
  - Implement the invoke method
    - Takes three parameters
      - proxy – Object that you want to be proxified
      - method – Method that you want to call
      - args – Object[] parameters to the method
Dynamic Proxies

- Time for some samples:
  - Method tracker
  - A simple dynamic proxy that tracks all method invocations
Dynamic Proxies

public class SimpleProxy implements InvocationHandler
{
    private Object object;
    public static Object newInstance(Object object)
    {
        return Proxy.newProxyInstance(object.getClass().getClassLoader(),
                                        CSSUtil.getInterfaces(object.getClass()), new SimpleProxy(object));
    }

    private SimpleProxy( Object object )
    {
        this.object = object;
    }
}
Dynamic Proxies

```java
public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    Object retval=null;
    try {
        track("Start method " + m.getName());
        retval = m.invoke(object, args);
    } catch (InvocationTargetException ite) {
        throw ite.getTargetException();
    } catch (Exception e) {
        throw new RuntimeException("Error occurred during method invocation: " + e.getMessage());
    } finally {
        track("End method " + m.getName());
    }
    return retval;
}
```
private void track(String message) {
    System.out.println(System.currentTimeMillis() + " ": " + message);
}

Dynamic Proxies
Dynamic Proxies

- The object to call

```java
public interface IWillWork
{
    public String doSomething();
}

public class WillWork implements IWillWork
{
    public WillWork()
    {
    }

    public String doSomething()
    {
        return "Done something";
    }
}
```
Dynamic Proxies

The client

```java
public class WillWorkTester {
    public static void main(String[] args) {
        IWillWork willWork = (IWillWork) SimpleProxy.newInstance(new WillWork());
        System.out.println(willWork.doSomething());
    }
}
```
Dynamic Proxies

Output

1123943703953: Start method doSomething
1123943703953: End method doSomething
Done something
Dynamic Proxies

- Adding standardized toString Functionality

  > Alter our previous proxy by adding a check prior to invoking the method:

  ```java
  ... if(m.getDeclaringClass().getName().equals("java.lang.Object") && m.getName().equals("toString"))
     {
       return doToString(proxyObject);
     }
  ...
  ```
Dynamic Proxies

And adding the doToString method:

```java
private String doToString(Object object)
{
    return ReflectionToStringBuilder.toString(this.object,ToStringStyle.MULTI_LINE_STYLE);
}
```
Dynamic Proxies

- The class that we want to call toString on

```java
public class ToStringSample implements IWillWork, IToStringSample {

    private String aString=null;
    private Integer aInteger=null;
    private boolean aBoolean=false;
}
```

- The class has corresponding getters/setters
Dynamic Proxies

And our test class, extending on the previous sample

```java
IWillWork toStringSample = (IWillWork) ToStringProxy.newInstance(new ToStringSample());
System.out.println(toStringSample.doSomething());
((IToStringSample) toStringSample).setAString("A string value");
((IToStringSample) toStringSample).setAInteger(new Integer(5));
((IToStringSample) toStringSample).setABoolean(true);
System.out.println(toStringSample.toString());
...
```
Dynamic Proxies

And when we run it we get:

1124013300250: Start method doSomething
1124013300250: End method doSomething
Done something
1124013300250: Start method setAString
1124013300250: End method setAString
1124013300250: Start method setAInteger
1124013300250: End method setAInteger
1124013300250: Start method setABoolean
1124013300250: End method setABoolean
1124013300328: End method toString

no.dnbnor.css2005.dp.samples.ToStringSample@1f1fba0[
  aString=A string value
  aInteger=5
  aBoolean=true
]
Dynamic Proxies

- An unproxyfied call to toString on the object would give us:

  Done something
  no.dnbnor.css2005.dp.samples.ToStringSample@194df86
Dynamic Proxies

- Dynamic proxies classes are singleton with respect to the interface they cover.
  - This means that for a given interface, the same proxy class will be used.
  - This feature comes in handy when it comes to, for instance, doing statistics.
  - Let’s implement a call counter.
Dynamic Proxies

Let’s create CallCounterProxy just as we did SimpleProxy.

- We need to add a new variable: int counter=1;
- And then we insert a call to the track method just prior to the previous call we made:

  track("Call number:" + counter++ + " on object: " + object.getClass().getName());

- For now we won’t worry about synchronization issues.
Dynamic Proxies

- Then we have our test client: CounterTester

...  
IWillWork willWork = (IWillWork) CallCounterProxy.newInstance(new WillWork());
System.out.println(willWork.doSomething());
System.out.println(willWork.doSomething());
System.out.println(willWork.doSomething());

// And now with a different Interface
IDifferent different = (IDifferent) CallCounterProxy.newInstance(new Different());
System.out.println(different.sayHello());
System.out.println(different.sayHello());
System.out.println(different.sayHello());

...
Dynamic Proxies

A test run will give us:

1130249584843: Call number:1 on object: no.dnbnor.css2005.dp.samples.WillWork
1130249584843: Start method doSomething
1130249584843: End method doSomething
Done something
1130249584843: Call number:2 on object: no.dnbnor.css2005.dp.samples.WillWork
1130249584843: Start method doSomething
1130249584843: End method doSomething
Done something
1130249584843: Call number:3 on object: no.dnbnor.css2005.dp.samples.WillWork
1130249584843: Start method doSomething
1130249584843: End method doSomething
Done something
1130249584843: Call number:1 on object: no.dnbnor.css2005.dp.samples.Different
1130249584843: Start method sayHello
1130249584843: End method sayHello
Hi there
1130249584843: Call number:2 on object: no.dnbnor.css2005.dp.samples.Different
1130249584843: Start method sayHello
1130249584843: End method sayHello
Hi there
1130249584843: Call number:3 on object: no.dnbnor.css2005.dp.samples.Different
1130249584843: Start method sayHello
1130249584843: End method sayHello
Hi there
Dynamic Proxies

- Validation
  - This is a concern that you run into many times
  - How do you implement validation in a flexible way
    - Write it directly into your business objects?
      - Not reusable – same logic often needed more than one place
      - Maintenance issues
    - Create a static validation Service object
      - Looser coupling
      - Business object still needs to be “validation aware”
Dynamic Proxies

- Using a dynamic proxy
  - Means full flexibility
    - Business objects are not aware of what goes on
    - Validation can be switched on/off at runtime
    - Validation logic can be altered at runtime through the use of a different invocation handler

- Lets do a simple sample
  - Implement a dynamic proxy that checks that all parameters to a method are not null
Dynamic Proxies

- We Create a Validation proxy just like our SimpleProxy and add a method checkForNulls, which we call just prior to invoking our method.

... checkForNulls(args);
retval = m.invoke(object, args);
...


private void checkForNulls(Object[] args)
{
    if (args != null && args.length > 0)
    {
        for (int i = 0; i < args.length; i++)
        {
            if (args[i] == null)
            {
                throw new RuntimeException("Parameter can not be null");
            }
        }
    }
}
Dynamic Proxies

We create a new client ValidationClient just like our ToStringTester, only this time we will set the AString property to null.

```java
IWillWork toStringSample = (IWillWork) ValidationProxy.newInstance(new ToStringSample());
System.out.println(toStringSample.doSomething());
((IToStringSample) toStringSample).setAInteger(new Integer(5));
((IToStringSample) toStringSample).setABoolean(true);
System.out.println(toStringSample.toString());
```
Dynamic Proxies

Running our test client yields:

1124017924234: Start method doSomething
1124017924234: End method doSomething
Done something
1124017924234: Start method setAString
1124017924234: End method setAString
java.lang.RuntimeException: Parameter can not be null

at no.dnbnor.css2005.dp.samples.ValidationProxy.checkForNulls(ValidationProxy.java:69)
at no.dnbnor.css2005.dp.samples.ValidationProxy.invoke(ValidationProxy.java:34)
at $Proxy0.setAString(Unknown Source)
at no.dnbnor.css2005.dp.samples.ValidationTester.main(ValidationTester.java:14)

Exception in thread "main"
Dynamic Proxies

- How we have made a better site by using dynamic proxies
  - We have a rather complex infrastructure
    - Several layers of application servers
    - Many legacy backend systems
      - Databases – JDBC
      - CICS, IMS – JMS/MQ
      - Partners – Corba, JMS, WebServices
Dynamic Proxies

Presentation layer  Business/Integration layer  Backends

RMI/IIOP
Dynamic Proxies

- **RMI/IIOP**
  - It is only possible to set a single system-wide timeout parameter.
    - Has to be at least the length of the longest running call
  - RMI/IIOP does not always honor the timeout, meaning calls may hang forever
Dynamic Proxies

Presentation layer

Dynamic Proxies

Business/Integration layer

RMI/IIOP

Backends
Dynamic Proxies

Introducing dynamic proxies meant:

- Calls are now run in separate threads
  - Gives the opportunity to individually set timeouts for a whole server, a remote object or a remote method.
  - Will throw a busy exception if it times out.
Dynamic Proxies

- Gives the opportunity to disable a whole server, a remote object or a remote method.
  - We can now, based on statistics, automatically disable at any level for a pre-configured period.
  - Implemented an administration application to manually disable calls.
  - Will throw a service unavailable exception.
Dynamic Proxies

- We get standardized logging
- We get standardized statistics
- But not least: No more hangs
Dynamic Proxies

- Performance (ms) – 1 million method calls:

![Graph showing performance comparison between JDK 1.3, JDK 1.4, and JDK 1.5. The graph compares unproxyfied and proxified performance.]
Dynamic Proxies

- References

  - http://java.sun.com/j2se/1.3/docs/api/java/lang/reflect/Proxy.html
Dynamic Proxies

- Questions