The Proxy Pattern

Relevant Links

- OODesign[1]
- GoF Book[2], page 207

Notes

Classification

The Proxy is a Structural Pattern.

Intent

To provide a surrogate / placeholder for another object to control access to that object.

Motivation

The intent of the proxy pattern is to control access to an object. There are a number of reasons why one might want to do that.

If the object is expensive to create, we might use a proxy to delay that creation until it is actually needed.

Or we might want to control exactly how an object's methods are called.

Or perhaps simply to log calls to the object's methods for testing purposes.

Some times we also have "remote proxies" that stand for objects that live in a different space (like another computer).

Participants / Implementation

There are 3 participants in a proxy pattern:

Subject This is the interface with the functionality that our object is meant to support.

RealSubject This is the actual object that our proxy is supposed to control access to. It must implement the Subject interface (in essence defines that interface).

Proxy The proxy also implements the Subject interface, and relays any appropriate requests to the RealSubject. It needs to maintain a reference to RealSubject to achieve its goals.

[1] http://www.oodesign.com/proxy-pattern.html

[2] http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/

Javascript implementation In its simplest form, a proxy would be constructed by creating a new object that contains the original as a property, and it also contains one property/method for each property/method of the original object. In general you do not need to provide more than what is part of the interface, but you have to also respect any properties inherited through the prototype.

If you do not care about controlling access to the prototype methods, you can simply inherit from the prototype directly.

Returning the correct this can get tricky, as the proxy must return itself whenever the original object would have returned this.

In our implementation, we will introduce two new objects: One is the actual proxy, which is meant to look exactly like the original object. The other is an object that manages the proxy, and acts as a glue between the two objects.

Here is a possible implementation[3].

Example Recall the forceContract method in the iterator:

```
proto.forceContract = function() {
    var hasNextSuccessful = false,
        oldNext = this.next,
        oldHasNext = this.hasNext;
    this.next = function() {
        if (hasNextSuccessful) {
            hasNextSuccessful = false;
            return oldNext.call(this);
        }
        throw new Error("Should only call 'next' after a successful 'hasNext'");
    };
    this.hasNext = function() {
        if (hasNextSuccessful) {
           throw new Error("Should not follow up a successful 'hasNext' with another 'hasNext
        }
        hasNextSuccessful = oldHasNext.call(this);
        return hasNextSuccessful;
    };
    return this;
};
```

One of the things we talked about back then was that it would be best if this method did not replace the next/hasNext methods. We can instead serve a proxy:

```
proto.forceContract = function() {
    var hasNextSuccessful = false,
        that = this;
    return Iterator.new(
        function next() {
            if (hasNextSuccessful) {
                hasNextSuccessful = false;
                return that.next();
            }
            throw new Error("Should only call 'next' after a successful 'hasNext'");
```

[3] ../../testPages/proxy.js

```
        },
        function() {
            if (hasNextSuccessful) {
               throw new Error("Should not follow up a successful 'hasNext' with another 'has
            }
            hasNextSuccessful = that.hasNext();
            return hasNextSuccessful;
        }
    );
};
```

So we create a new iterator that restricts access to the old iterator to only appropriate calls: You must call hasNext and if it is successful follow it up with a next before you can use hasNext again. This would be called a **protection proxy**.