search...

Main Menu

Home
Design Principles
Open Close Principle
Dependency Inversion Principle
Interface Segregation Principle
Single Responsibility Principle
Liskov's Substitution Principle
Creational Patterns
Singleton
Factory
Factory Method
Abstract Factory
Builder
Prototype
Object Pool
Behavioral Patterns
Chain of Responsibility
Command
Interpreter
Iterator
Mediator
Memento
Observer
Strategy
Template Method
Visitor
Null Object
Structural Patterns
Adapter
Bridge
Composite
Decorator
Flyweight
Proxy
Design Pattern Books

Forum

What Design Pattern To Choose?
Design Principles and Patterns
Enterprise Patterns
Books

Home » Creational Patterns » Object Pool
Bookmark this on Delicious

Object Pool

Motivation

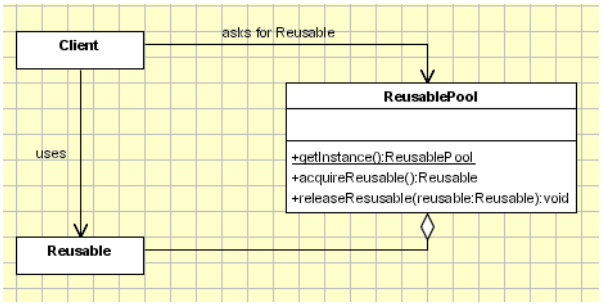
Performance can be sometimes the key issue during the software development and the object creation(class instantiation) is a costly step. While the Prototype pattern helps in improving the performance by cloning the objects, the Object Pool pattern offer a mechanism to reuse objects that are expensive to create.

Clients of an object pull "feel" like they are owners of a service although the service is shared among many other clients.

Intent

-reuse and share objects that are expensive to create.

Implementation



Implementation involves the following objects:
Reusable - Wraps the limited resource, will be shared by several clients for a limited amount of time.
Client - uses an instance of type Reusable.
ReusablePool - manage the reusable objects for use by Clients, creating and managing a pool of objects.

When a client asks for a Reusable object, the pool performs the following actions:
- Search for an available Reusable object and if it was found it will be returned to the client.
- If no Reusable object was found then it tries to create a new one. If this actions succeds the new Reusable object will be returned to the client.
- If the pool was unable to create a new Reusable, the pool will wait until a reusable object will be released.

The Client is responsible to request the Reusable object as well to release it to the pool. If this action will not be performed the Reusable object will be lost, being considered unavailable by the ResourcePool.

The clients are not aware that they are sharing the Reusable object. From the client poinf of view they are the owners of a new object which comes from the Resource pool in the same way that it comes from a factory or another creational design pattern. The only difference is that the Client should mark the Reusable object as available, after it finishes to use it. It's not about releasing the objects; for example if we work with databases, when a connection is closed it's not necesarely distroyed, it means that it can be reused by another client.

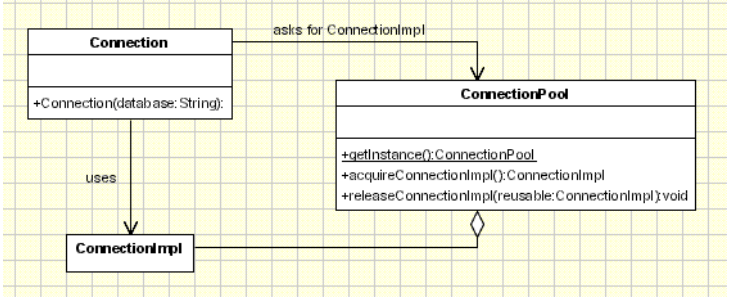
Why to use it?

Basically, we'll use an object pool whenever there are several clients who needs the same stateless resource which is expensive to create.

Applicability & Examples

Lets' take the example of the database connections. It's obviosly that opening too many connections might affect the performance for several reasons:
Creating a connection is an expensive operation.
When there are too many connections opened it takes longer to create a new one and the database server will become overloaded.

Here the object pool comes in to picture to manage the connections and provide a way to reuse and share them. It can also limit the maximum number of objects that can be created.



This pattern provide the following mechaninsm:

Connection - represent the object which is instantiated by the client. From the client perspective this object is instantiated and it handles the database operations and it is the only object visible to the client. The client is not aware that it uses some shared connections. Internally this class does not contain any code for connecting to the database and calls ConnectionPool.aquireImpl to get a ConnectionImpl object and then delegates the request to ConnectionImpl.

ConnectionImpl is the object which implements the database operations which are exposed by Connection for the client.

ConnectionPool is the main actor to manage the connections to the database. It keeps a list of ConnectionImpl objects and instantiates new objects if this is required.

When the client needs to query the database it instantiate a new Connection object specifing the database name and the call the query method which returns a set of records. From the client point of view this is all.

When the Connection.Query methd is called it asks for a ConnectionImpl object from the ConnectionPool. The ConnectionPool tries to find and return an unused object and if it doesn't find it creates one. At this point the maximum number of connections can be limited and if it was reached the pool cand wait until one will be available or return null. In the query method the request is delegated to the ConnectionImpl object returned by the object pool. Since the request is just delegated it's recomended to have the same method signature in Connection and ConnectionImpl.

Specific problems and implementation

Singleton reusable pool - The reusable pool is implemented as a singleton. The reusable pool should be accesible only to the Connection object.

1. Limited number of resources in the pool

The connection pool is responsable for sharing and reusing the resources. Sometimes the resources have to be well managed only because they affects the performace, but there are situations when the number of resources can not exceed a specific limit. In this case the Resource pool check the number of instantiated resources and of the limit is reach it will wait for a resource to be released, it will throw an exception or it will return a null value. In any of the last 2 situations the Client should be notified that the action failed because there are no available resources.

2. Handling situations when creating a new resource fails

There are many reasons when the ResourcePool.acquireConnectionImpl method fails to return a resource. It might happens because there are not available resources or some exception occured. Either way the client should be notified about his.

3. Syncronization

In order to work in a multithreading environment the methods that are used by differnt threads should be synchronized. There are only three methonds in the ResourcePool object that have to be synchronized:
- getInstance should be synchronized or should contain a synchronized block. For details check the singleton multithreading implementation.
- acquireConnectionImpl - this menthod returns a resource and should be synchronized not to return the same resource to two different clients running tin different threads.
- releaseConnectionImpl - this method release a resource. Ussually it doesn't have to be synchronized a resource is allocated only by one client. Internally some blocks might need to be synchronized(depending on the method implementation and the internal structures used to keep the pool.).

4. Expired resources(unused but still reserved)

The main problem for the Object Pool Pattern is that the objects should be released by the client when it finishes using them. There are plenty of examples when the client ”forget” to release the resources. Let's take the example the the database connections when connection are not closed/released after they are used. This seems a minor problem but there are many applications crushing for this reason.

In object pool can be implemented a mechanism to check when a specific resource was used last time and if the time expired, to return it to the available resource pool.

Hot Points

- When the Object Pool pattern is used the objects should be marked as available(released) by the client after they are used, so the pool will be aware about this. This is the main drawback because the client should do this and it's a common situation when database connection are not released afer they are used. To overcome this a mechanism can be implemented to release resources if they are not used for a period of time.
- Creating the resources might fail and this case should be treated carefully. When there is no available resource(beacause the number is limited or creating a new one failed) the client should be notified about it.

Conclusion

Althrough the object pool is handling the object instantiation it's main purpose is to provide a way for the clients to reuse the objects like they are new objects, without being shared and reused.

< Prev

[Back]