A Simple Pool Client

Suppose you're writing a set of `java.io.Reader` utilities, and would like to provide a method for dumping the contents of a `Reader` to a `String`. Here's the code for the `ReaderUtil`, implemented without an `ObjectPool`:

假设：写一个Reader工具类，提供一个将Reader里的内容转储为String。

```
import java.io.Reader;
import java.io.IOException;

public class ReaderUtil {
    public ReaderUtil() {
    }

    /**
     * Dumps the contents of the {@link Reader} to a
     * String, closing the {@link Reader} when done.
     */
    public String readToString(Reader in) throws IOException {
        StringBuffer buf = new StringBuffer();
        try {
            for(int c = in.read(); c != -1; c = in.read()) {
                buf.append((char)c);
            }
            return buf.toString();
        } catch(IOException e) {
            throw e;
        } finally {
            try {
                in.close();
            } catch(Exception e) {
                // ignored
            }
        }
    }
}
```

For the sake of this example, let's assume we want to to pool the `StringBuffer`s used to buffer the `Reader`'s contents. (A pool of `StringBuffer`s may or may not be useful in practice. We're just using it as a simple example here.)

Let's further assume that a complete pool implementation will be provided via a constructor. (We'll show you how to create such an implementation in just a moment.) Then to use the pool we simply call `borrowObject` to obtain the buffer, and then call `returnObject` when we're done with it. Then a `ReaderUtil` implementation using a pool of `StringBuffer`s might look like this:

```
import java.io.IOException;
import java.io.Reader;
import org.apache.commons.pool2.ObjectPool;

public class ReaderUtil {
```

```
    private ObjectPool<StringBuffer> pool;

    public ReaderUtil(ObjectPool<StringBuffer> pool) {
        this.pool = pool;
    }

    /**
     * Dumps the contents of the {@link Reader} to a String, closing the {@link Reader} when done.
     */
    public String readToString(Reader in)
        throws IOException {
        StringBuffer buf = null;
        try {
            buf = pool.borrowObject();
            for (int c = in.read(); c != -1; c = in.read()) {
                buf.append((char) c);
            }
            return buf.toString();
        } catch (IOException e) {
            throw e;
        } catch (Exception e) {
            throw new RuntimeException("Unable to borrow buffer from pool" + e.toString());
        } finally {
            try {
                in.close();
            } catch (Exception e) {
                // ignored
            }
            try {
                if (null != buf) {
                    pool.returnObject(buf);
                }
            } catch (Exception e) {
                // ignored
            }
        }
    }
}
```

Since we've constrained ourselves to the `ObjectPool` interface, an arbitrary pool implementation (returning, in our case, `StringBuffer`s) can be used. When a different or "better" pool implementation comes along, we can simply drop it into our `ReaderUtil` without changing a line of code.

A PooledObjectFactory

The implementations provided in pool2 wrap pooled objects in `PooledObject` wrappers for internal use by the pool and object factories. The `PooledObjectFactory` interface defines lifecycle methods for pooled objects. The simplest way to implement a `PoolableObjectFactory` is to extend `BasePooledObjectFactory`.

Here's a `PooledObjectFactory` implementation that creates `StringBuffer`s as used above.

```
import org.apache.commons.pool2.BasePooledObjectFactory;
import org.apache.commons.pool2.PooledObject;
import org.apache.commons.pool2.impl.DefaultPooledObject;

public class StringBufferFactory
    extends BasePooledObjectFactory<StringBuffer> {

    @Override
    public StringBuffer create() {
        return new StringBuffer();
    }

    /**
     * Use the default PooledObject implementation.
     */
    @Override
    public PooledObject<StringBuffer> wrap(StringBuffer buffer) {
        return new DefaultPooledObject<StringBuffer>(buffer);
    }

    /**
     * When an object is returned to the pool, clear the buffer.
     */
    @Override
    public void passivateObject(PooledObject<StringBuffer> pooledObject) {
        pooledObject.getObject().setLength(0);
    }

    // for all other methods, the no-op implementation
    // in BasePooledObjectFactory will suffice
}
```

We can, for example, use this factory with the `GenericObjectPool` to instantiate our `ReaderUtil` as follows:

```
ReaderUtil readerUtil = new ReaderUtil(new GenericObjectPool<StringBuffer>(new StringBufferFactory()
));
```