Understanding Object Life Cycle

Object life cycle generally refers to how an object-oriented application (whether distributed or not) creates and destroys objects. For distributed applications, life cycle management presents particular challenges. For example, destruction of objects often can be surprisingly complex, especially in threaded applications. Before we go into the details of object creation and destruction, we need to have a closer look what we mean by the terms "life cycle" and "object" in this context.

Object life cycle refers to the act of creation and destruction of objects. For example, with our file system application, we may start out with an empty file system that only contains a root directory. Over time, clients (by as yet unspecified means) add new directories and files to the file system. For example, a client might create a new directory called `MyPoems` underneath the root directory. Some time later, the same or a different client might decide to remove this directory again, returning the file system to its previous empty state. This pattern of creation and destruction is known as object life cycle.

The life cycle of distributed objects raises a number of interesting and challenging questions. For example, what should happen if a client destroys a file while another client is reading or writing that file? And how do we prevent two files with the same name from existing in the same directory? Another interesting scenario is illustrated by the following sequence of events:

1. Client A creates a file called `DraftPoem` in the root directory and uses it for a while.
2. Some time later, client B destroys the `DraftPoem` file so it no longer exists.
3. Some time later still, client C creates a new `DraftPoem` file in the root directory, with different contents.
4. Finally, client A attempts to access the `DraftPoem` file it created earlier.

What should happen when, in the final step, client A tries to use the `DraftPoem` file? Should the client's attempt succeed and simply operate on the new contents of the file that were placed there by client C? Or should client A's attempt fail because, after all, the new `DraftPoem` file is, in a sense, a completely different file from the original one, even though it has the same name?

The answers to such questions cannot be made in general. Instead, meaningful answers depend on the semantics that each individual application attaches to object life cycle. In this chapter, we will explore the various possible interpretations and how to implement them correctly, particularly for threaded applications.

See Also

- Slice for a Simple File System