The Orderly Shutdown Pattern

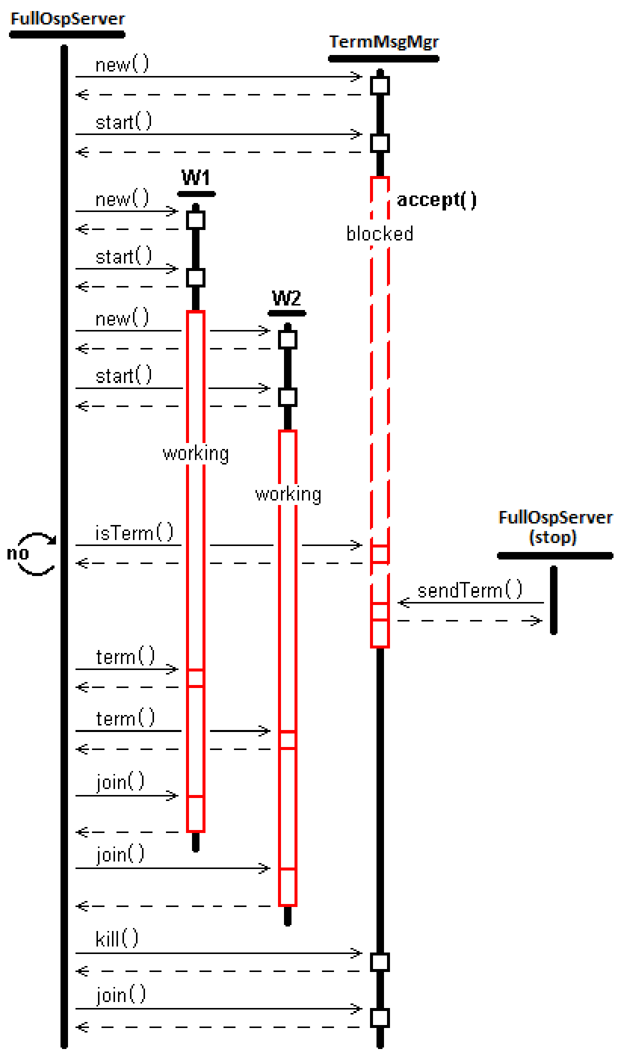
Orderly Shutdown Pattern Repository Overview

This repository contains materials to introduce and demonstrate **The Orderly Shutdown Pattern** (OSP).

The Orderly Shutdown Pattern (OSP) allows a running service of arbitrary complexity to monitor for, and respond to *at a time of its own choosing*, a request to shutdown. If/when such request is received, the service will then be able to complete any critical work it has queued or initiated (at the discretion of the service) prior to entering a shutdown process leading ultimately to termination.

The advantage of this Pattern is that the running service can determine when *and* how to handle the shutdown process. This may include re-enqueueing with a message broker work tasks that are not complete. Alternatively, it may include completing those tasks prior to termination. Any pending I/O operations can be properly flushed and closed, and any related services can likewise be notified of its pending termination.

How is this different from Unix signal handling? To begin with, this pattern allows control of the service even if that service's host does not allow console access. No access? No problem; rather, network to the target host from any (every?) machine inside the firewall. Secondly, the shutdown message *could* include any level of nuance (beyond simply "Shutdown as soon as convenient" which is implemented here). Any port can be used (accommodating firewall and tunneling factors); requires no higher-level protocol (HTTP, etc); very light weight (a single socket on a single thread, vs embedded http server, etc.); extendable to SSL if desired/required.



Example: FullOspServer
(red indicates OS thread)

Elements of the OSP

The Orderly Shutdown Pattern relies upon the following basic concepts -

- Multi-Threading
- Socket Communication

To understand the operation and use of the OSP implementations (basic and full - see below) it is required to be familiar with the above two concepts.

Concepts Background and Introduction

As the ajoining diagram illustrates, the OSP wraps additional logic and processing around the above concepts, leveraging their potential. This perhaps could make it difficult for some to modify, enhance or refactor the OSP architecture.

Therefore, the most basic of examples of these two foundational concepts - multi-threading and sockets communication - is included herein as an introduction where desired. Please see -

- Basic multi-threading example: mt_basic
- Basic server-client socket communnication example: socket_basic

The Orderly Shutdown Pattern Implementations - Basic and Full

Two fully operational implementations are offered - basic and full. Both of these implementations use the following devices to generically demonstrate the OSP -

- A thread sleep behavior to simulate the passage of time spent on 'real, uninterruptible work.'
- An internal counter that, should no termination message be sent, will eventually terminate the service (to illustrate the service shutdown behavior in any case).

The OSP Basic Implementation

The Basic implementation includes these features -

- Use of a separate thread to listen for and manage termination message receipt. This is encapsulated in a separate Class derived from Thread.
- Use of a listener socket in the service process which remains open for the duration awaiting a termination message.
- An 'uninterruptible work' loop that runs in the service, periodically checking for termination message arrival.

The OSP Full Implementation

The Full implementation employs all of the elements of the Basic implementation, and adds the following -

- Two instances of a separate threaded worker class that does the 'uninterruptible work' of the service, enabled to check for the presence of the termination condition (within process, not across the network) at appropriate times.
- An enhancement to the termination listener socket management to *fend off* poorly formed, or completely random, messages received on that socket.
- An enhancement to the termination sender logic to allow the termination message to be sent from a host other than the host on which the target service is running. For testing, this logic still supports 'localhost' as the target of the termination message.

Building and Running the OSP Examples

All the examples are command-line only -- no GUI stuff involved. That is both intentional and practical - these are server-based concepts, not desktop-based (although they *can* be employed there). To run some of the examples, only a single console window is required; for others, a second console window must be used (for example, to execute a stop script on a service running in another window).

All the examples have been built and tested on Ubuntu (specifically Ubuntu-server 17.10) though no features specific to that distro have been employeed. For the Java pieces, even older JDK releases will work but testing has been done using Java 1.7 thru Java 9.

All the elements of the OSP repository contain working versions (mt, socket, basic, full). There are build scripts or make files for each. Details on installing the proper software for your environment is outside the scope of this effort but there are innumerable sources for this and I've added in below some of the install steps taken from my cmd-line history.

In brief, you will need C++ or Java development tools on your machine to run these examples. You can download a tarfile from github, or clone the repository (preferred, and easy as pie). As noted throughout, an MIT Open Source License has been attached to everything - it's short, direct, complete, and quite permissive, but PLEASE READ THE LICENSE (appearing below) if you are unfamiliar with its permission grants and restrictions.

Comments (and brief questions) are welcome at jt [at] iwaytechnology [dot] com.

Supporting Software Installation Hints

This is not intended to be a tutorial on build-machine setup, but the following steps were taken to prepare an Ubuntu-server for creating these works. Note that not all of these tools were/are required (in particular, the ubuntu-desktop), but many may have been used tangentially in the process.

Some Ubuntu Tools

The Ubuntu Desktop install (see below) I've found to be lightweight and quick to install because it will avoid dragging down 'office-like' software and other things superfluous for a development-only machine. I'm not sure the gnome-terminal install is required, but there you go; Firefox I believe is absent initially; openssh-server allows you to ssh to the machine (some simple configuration is required - see the web for hints - you're looking for /etc/ssh/sshd_config ... if that file is missing, it suggests the package is not installed yet.

```
$ sudo apt-get install --no-install-recommends ubuntu-desktop
$ sudo apt-get install gnome-terminal
$ sudo apt-get install firefox
$ sudo apt-get install openssh-server
$ sudo apt-get install git-core
```

C++

Many dev package installs on my machines are probably not actually used in the OSP code. Start with this and see what more you need to build and run the example code. This should include gcc/g++, make, libs and headers.

```
$ sudo apt-get install build-essential
```

Java

```
$ sudo apt-get install openjdk-8-jdk
- or -
$ sudo apt-get install oracle-java9-installer
```