Trabajo Práctico Especial
Protocolos de Comunicación 2013b

Proxy HTTP

Alderete, Facundo
Domingues, Matías
Martínez Correa, Facundo

Índice

Introducción

En este escrito presentamos el proxy HTTP hyttrop. Dicha aplicación fue concebida en dos partes. La primera, un framework basado en el patrón reactor con el cual se establece la forma de trabajo en las conexiones. La segunda, es la lógica de negocio del proxy, que indica cómo se manejan las conexiones de los clientes y paso de los mensajes, además de las transformaciones. A su vez, cuenta con una tercera parte ad-hoc, que es el sistema de administración y configuración, con el cual se puede establecer propiedades, cambiar puertos para la exposición del servicio y recolectar estadísticas.

Framework Thorium

Introducción

En la realización del proyecto nos vimos con la necesidad de implementar un framework que nos separe la lógica de negocio del proxy, en este caso el manejo del protocolo HTTP, respecto del manejo de conexiones. Esto se nos hizo evidente, puesto que HTTP es un protocolo de aplicación, y su naturaleza es ser interpretado desde una forma lógica en la cual se abstrae de conexiones y paquetes.

Con este motivo, es que nació Thorium. Un framework basado en la implementación de Ron Hitchens mostrada en el libro Java NIO de editorial O'Reilly. En este libro se utiliza la librería de Java para crear un framework basado en el patrón Reactor mediante el uso de canales no bloqueantes provistos por la mencionada API.

Otra razón importante para esto, que la importante separación que implica la creación de un framework hace que el resolver problemas y bugs sea altamente eficiente, puesto que se puede diferenciar fácilmente si es que se trata de un problema en el cliente o no. Además mejora la implementación, ya que hace difícil el derrame entre capas.

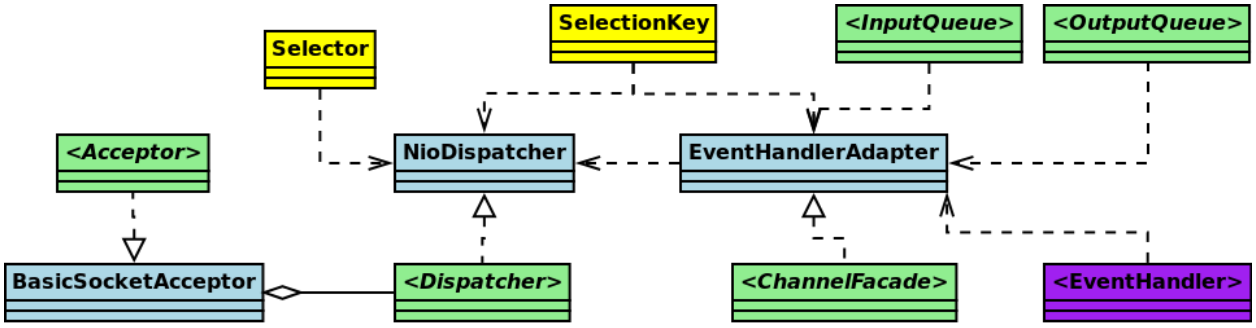
La razón para basarnos en la implementación de Hitchens fue que nos pareció simple y sencilla, proveyendo al usuario de la plataforma sólida e interfaces intuitivas.

Arquitectura General

El framework consta de dos threads principales y un pool de threads para el procesamiento de eventos. Los otros dos threads corresponden al de aceptación de nuevos clientes y al de despacho de eventos.

Diseñamos el framework teniendo en cuenta que muchos de los motivos por los que se critica al patrón reactor en una ganancia marginal de performance respecto a un patrón no bloqueante se debe a la inclusión de la capa de aceptación de conexiones (de ahora en más *Acceptor*) en el mismo thread que en la capa de despacho de eventos (de ahora en más *Dispatcher*).

El Dispatcher trabajará con manejadores de eventos, como lo indica el patrón Reactor. Debido a que esto debe ser genérico, es que se utiliza el patrón Adapter para crear un objeto que envuelve y adapta el manejador de eventos, además de proveerle al cliente una serie de facilidades para la comunicación.



El cliente deberá implementar tres interfaces, para hacer uso del framework de manera completa. Dichas interfaces son:

- ***EventHandler***: Esta interfaz es la principal que debe crear el usuario y es la que será llamada al recibir una nueva conexión, al recibir un nuevo mensaje, para el preprocesamiento de mensajes a enviar y al finalizar el la conexión del cliente. Habrá uno de estos por conexión.
- ***Message***: Esta interfaz funciona como un marcador, sin la necesidad de implementar ningún método. Sirve meramente para abstraer al framework del mensaje que el cliente espera. Puede haber más de uno por conexión.
- ***MessageValidator***: Esta interfaz se encarga de, como dice su nombre, validar que los bytes recibidos sean un mensaje apropiado para el cliente. En el momento que este valide el mensaje, se invocará al cliente para procesar dicho mensaje. Habrá uno por conexión.

Una cuarta interfaz que debiera implementar es la de ***EventHandlerFactory,*** la cual le sirve al framework para crear nuevos EventHandlers a medida que se suscriben conexiones.
A su vez, el cliente verá y se comunicará por medio de la interfaz ChannelFacade, InputQueue y OutputQueue.

Dispatcher y el manejo de Adapters

El dispatcher maneja todos los adapters en un solo loop dentro de su thread. En dicho thread no se realiza el proceso de lectura y escritura en los canales, puesto que esto tiene que llamar a código del cliente, y como regla principal del framework para mejorar su seguridad y estabilidad, es hacer la menor cantidad de llamadas al cliente posible en threads que no sean los de trabajo.

A su vez, cada adapter se anota en el dispatcher avisando cambios que no necesariamente pueden verse reflejados en las keys que contienen, como ser su finalización.

Lo que el cliente ve: ChannelFacade y Queues

El cliente, como se comentó antes, sólo puede ver la interfaz ChannelFacade, InputQueue y OutputQueue. Estas interfaces fueron necesarias de implementar por ser una consecuencia de la asincronía que implica el uso de canales no bloqueantes y la posibilidad de tener un mensaje recibido de forma incompleta.

El ChannelFacade, como indica su nombre, abstrae el canal y expone las funcionalidades básicas que se pueden realizar con él. En particular, en esta versión del framework, solamente se puede obtener las colas de salida y entrada.

Las interfaces de InputQueue y OutputQueue expresan de manera simple cómo es el funcionamiento general del proxy al momento de comunicarse con el exterior: como una cola que bufferea. Esto es necesario por la necesidad de ser flexibles al momento de transmitir el mensaje. Ya que es posible que el mensaje no pueda ser transmitido de una sola pasada y se requieran más de una. Estos objetos son los responsables de volver a suscribir al cliente para estos eventos, liberando al cliente de dicha tarea.

Proxy HTTP: hyttrop

La idea básica en la realización fue siempre tener en cuenta la optimización de tiempos y la performance. Es por eso que decidimos, dados los requerimientos impuestos por la cátedra, que no era necesario en ningún momento el uso de espacio en disco y que todo podía ser bufferizado en memoria y pasado «on the fly». Incluso al momento de realizar transformaciones.

Como lo indica la filosofía del framework que utiliza, la idea es ser liviano y rápido, minimizando el footprint en procesador y el context-switching tratando de no atentar contra la memoria ocupada.

Debido al uso de un framework, su implementación fue sumamente sencilla y de relativa facilidad. Únicamente, como parte de la extensión de thorium, debimos implementar un validador de mensajes http, él cual se encargará de que él mismo llegue completo, y dé que no pase de manos hasta que cumpla con todas las validaciones correspondientes. Por otra parte, implementamos objetos representativos tanto de un request como de un response, para establecer más claramente la diferencia entre uno y otro. Sin embargo contemplamos que las partes básicas que tienen en común esten contenidas en una clase abstracta, llamada HttpMessage, la cual manejara únicamente él body del mensaje.

Problemas encontrados

Para poder realizar el tratamiento del gzip, se tuvo que remover los headers particulares que definian el encoding, tamaño y compresión del archivo, ya que se busco que la descompresión ocurriera sin bufferizar los paquetes y se enviara instantaneamente.

En conjunto con el punto anterior, se tuvo que "des-chunkear" el mensaje para que pudiera ser procesado y luego convertido sin problemas.

Tuvimos que una inputqueue especial para el tratamiento del formato gzip, ya que él método que realiza la descompresión se bloquea en el caso de que se consuma todo el buffer y no haya encontrado el EOF. Nuestra inputqueue especial se encargará entonces de que siempre haya al menos 512 bytes en la cola, hasta llegar al EOF.

Protocolo de administración

Para acceder a las configuraciones del sistema y a la información de estadísticas implementamos un protocolo orientado a línea. El sistema de administración se accede mediante a un cliente (netcat por ejemplo) apuntando a la dirección ip donde este corre, y al puerto que se configura en el archivo *hyttrop.conf*. Al acceder al administrador se muestra un mensaje de bienvenida, y se pone a disposición del usuario la entrada estándar, para comenzar a ingresar los comandos. Este protocolo se basa en POP3 en cuanto a la forma de ingresar comandos y recibir respuestas. En el caso de que el comando y los parametros ingresados esten dentro de los soportado por el administrador se devolverá una respuesta positiva, además de un mensaje indicando los cambios que la ejecución del mismo género en el sistema. En caso de que los parámetros pasados al comando no sean los correctos se emitirá una respuesta positiva, pero se informará del correcto uso del mismo. En caso de error simplemente se emitirá una respuesta negativa. Toda respuesta será finalizada con una línea consistente únicamente con el caracter punto [‘.’].

Además, del mismo modo que Hyttrop, se montará sobre la base de Thorium. Pero, en contraste con nuestro proxy, actuará como servidor. Cabe destacar que no se requerirá ningún sistema de autenticación para acceder a este servicio.

```
facundo@facundo-Inspiron-1545:~/hyttrop$ nc localhost 2345
+OK Bienvenido al sistema de administracion de Hyttrop. Ingrese un comando para comenzar...

.
set l33t on
+OK Transformacion l33t encendida.

.
set l33t off
+OK Transformacion l33t apagada.

.
set statistics on
+OK Calculo de estadisticas encendido.

.
help statistics
Con este comando se podra activar, desactivar u obtener el calculo de estadisticas.
[set|get|help] statistics [on|off|reset]
help l33t
Con este comando se podra activar la transformacion l33t en los mensajes de tipo text/plain.
[set|help] l33t [on|off]
```

Un comando consiste en una clase que debe implementar la interfaz Command. En ella se define que un comando será capaz de tener un nombre identificatorio, proveer una ayuda breve y una extensa al usuario, describir que tipo de acciones aceptara (get, set, help) y finalmente implementar la lógica de su ejecución.

Finalmente, para que un comando este activo dentro del sistema, debe ser dado de alta frente al protocolo de administración. Esto se hace agregando el nuevo comando a la lista de comandos dentro del protocolo de administración (dentro de AdminHandlerFactory). En caso de que el nuevo comando no cumpla con las reglas requeridas por nuestro estándar, el sistema lanzará una excepción e informará del error mediante una entrada en el log.

Comandos de base:

- *get*: se utilizará para obtener información del administrador.
- *set*: con él se podrá generar un cambio en el estado interno del administrador.
- *help*: sirve para obtener ayuda acerca del comando solicitado.

Comandos implementados:

- l33t: como de él no se espera obtener una salida, únicamente aceptará como comandos base a *set* y a *help*. Además sus parámetros serán *on* y *off*, provocando que la conversión l33t se encienda o se apague, respectivamente.
- statistics: se lo usará para encender o apagar el cálculo de estadísticas mediante el comando base *set* y los parámetros *on*, *off*, y *reset* (vuelve las estadísticas a cero). Si se lo utiliza con el comando *get* se podrán obtener los datos de estadísticas, siempre que los cálculos se encuentren encendidos.
- proxy-port: servirá para obtener el puerto en el que se está proveyendo el servicio de proxy, con el comando base *get*. Si se lo ejecuta con el comando base *set* se podrá configurar este puerto a voluntad, pasándolo como parámetro.
- admin-port: idem al anterior, pero referido al puerto donde se ejecuta el administrador.
- default-origin-server: podrá configurarse u obtenerse él host y puerto default al que se mandaran todas las conexiones en caso de que quiera encadenarse nuestro proxy con otro.

Las estadísticas que serán provistas por nuestro administrador serán recolectadas por un objeto singleton, él cual podrá ser accedido cuando se lo necesite, ya que los datos para elaborar las estadísticas provendrán de distintas partes de nuestro proxy. Los datos que proveeremos serán:

- Cantidad de bytes transmitidos.
- Cantidad de conexiones establecidas.
- Histograma de relación entre un status code y la cantidad de su tipo que pasaron por nuestras manos.

Estos datos serán recolectados mientras el cálculo de este encendido. En el caso de que se decida detenerlo, los datos recolectados hasta el momento no se perderán, ya que se encuentran almacenados dentro del singleton, y podrán formar parte de la estadística nuevamente en caso de que la captura vuelva a activarse. Él único caso en que se perderán estos datos es cuando se dé de baja el servicio del proxy, ya que no se ofrece persistencia mediante archivos ni de ningún otro tipo.

Manual de usuario

A continuación especificaremos la forma correcta de ingresar un comando y la acción consecuente que él mismo generara en el sistema.

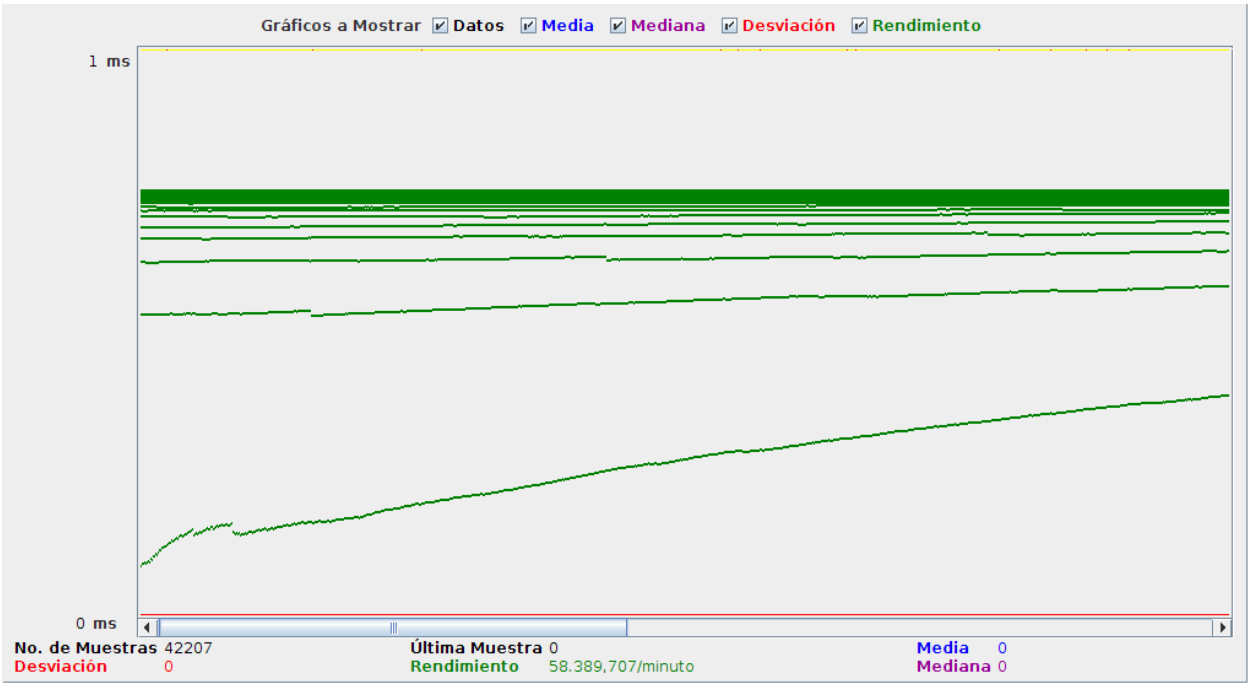
- set l33t [on|off] : enciende o apaga la conversión “l33t”.
- set statistics [on|off|reset] : enciende, apaga o reinicia las estadísticas.
- get statistics : obtiene el informe de estadísticas.
- set proxy-port <port> : cambia el puerto donde se ofrece el servicio de proxy.
- get proxy-port : para obtener el puerto donde se ofrece el servicio de proxy.
- set admin-port <port> : cambia el puerto donde se ofrece el servicio de administración.
- get admin-port : para obtener el puerto donde se ofrece el servicio de administración.
- set default-origin-server <host> <port> : para encadenar con otro proxy.
- get default-origin-server : para saber con que proxy se está encadenando.

Benchmarking

Para las pruebas de estrés utilizamos JMeter, con la siguiente configuración:

- Número de threads: 1000
- Periodo de subida: 100 segundos
- Contador del bucle: 100
- Hacemos pedidos a www.youtube.com:80, pidiendo la página principal.

El gráfico que obtenemos a la salida es el siguiente:



Como se puede ver aquí, el rendimiento empieza a disminuir a medida que se van sumando usuarios al proxy, pero llega un momento en que se estabiliza el tiempo de latencia.

Esto es consecuencia de la implementación en el manejo de conexiones que elegimos. Al utilizar el patrón reactor, la cantidad de threads por conexión no necesariamente crece de manera lineal, lo cual implica un menor consumo de recursos y una menor cantidad dé context switching.