powered by: thorium

hyttrop

An http Proxy

Alderete, Facundo
Domingues, Matias
Martinez Correa, Facundo

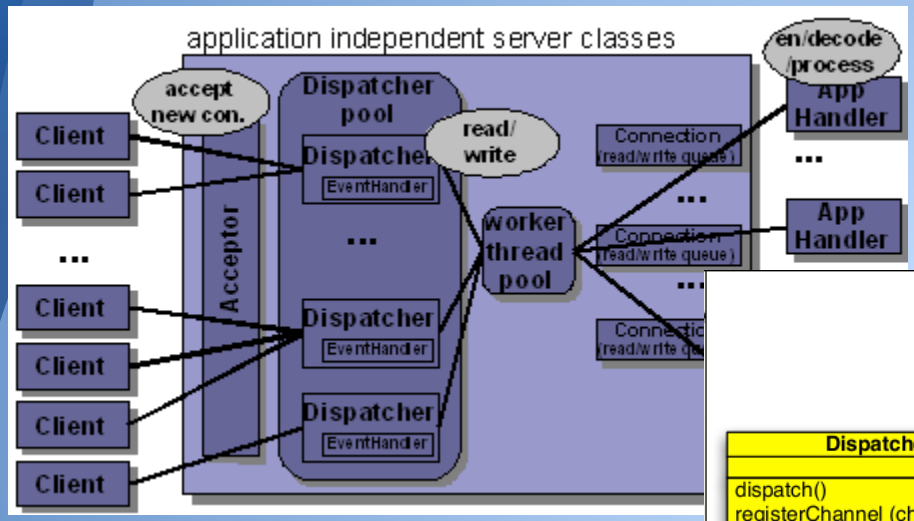
HYTTROP

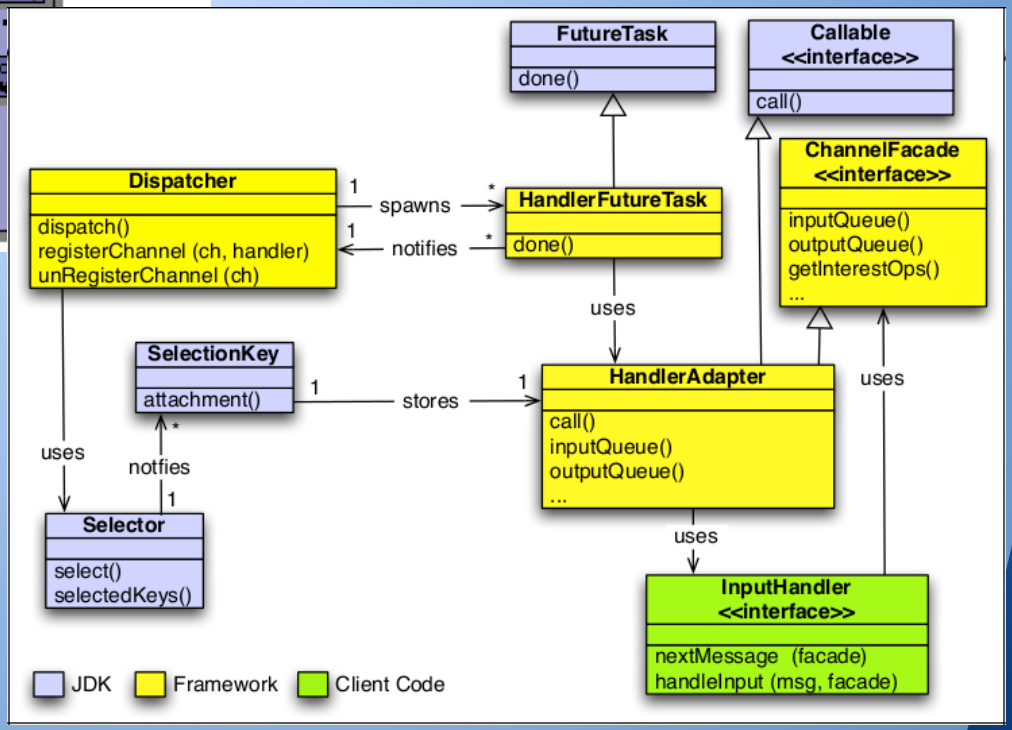
- Diseño
  - Bases y Diseño original
  - Nuestro diseño
  - ¿Por qué?
  - Acceptor y Dispatchers
  - HandlerAdapters e EventHandlers
  - ChannelFacade y Queues
  - FutureTasks
  - Servicios
- Funcionamiento
  - Comenzando la comunicación
  - Recibiendo y enviando mensajes
  - Administración
  - Posibles Mejoras y Extensiones

Diseño: Bases y Diseño original (Thorium)

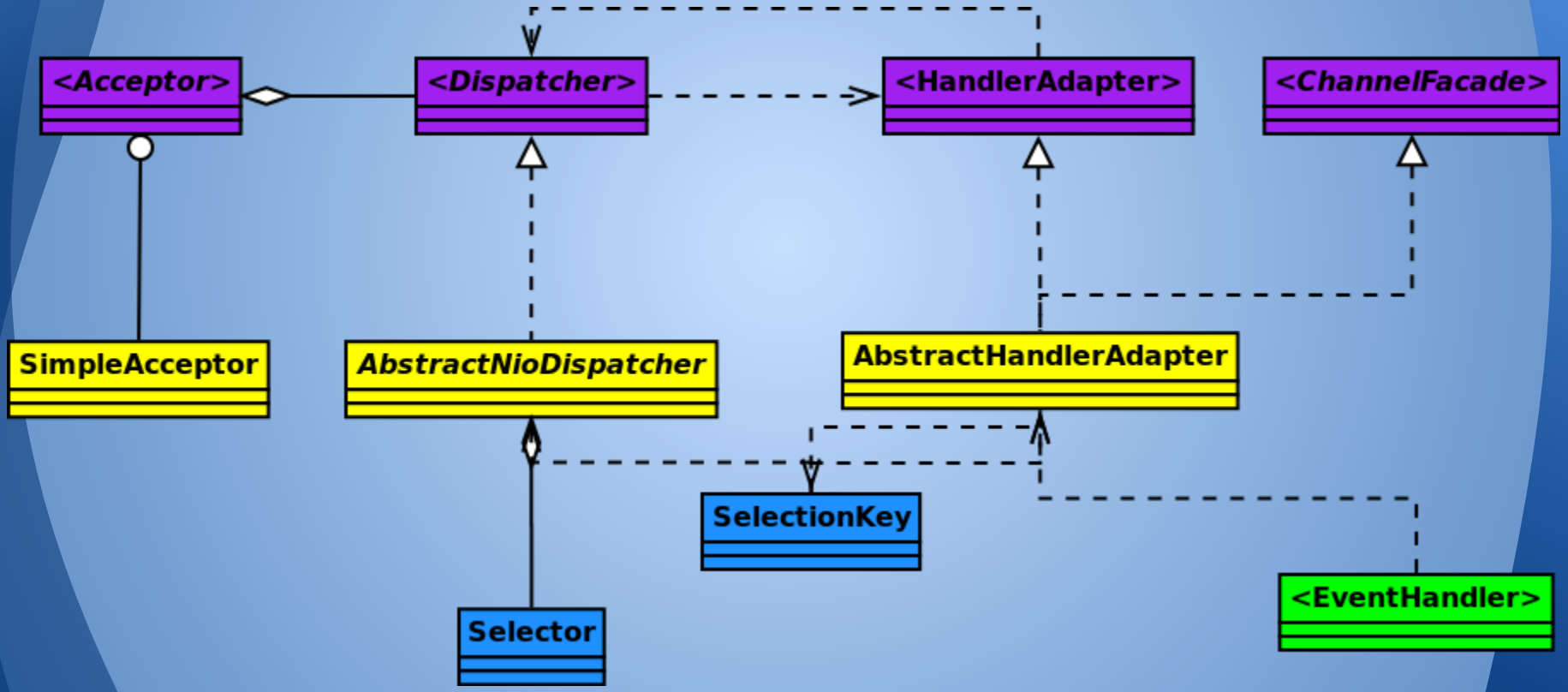
- Basado en Server Mark II de Ron Hitchens. [Java NIO. Hitchens, Ron. O'Reilly]
- Patrón Reactor.
- Es un framework para servidores usando canales NIO no bloqueantes
- Usando RFC: 6120, 6121, 6122

Diseño: Bases y Diseño original





Diseño: Nuestro Diseño



Acceptors y Dispatchers

- Un thread para cada uno. Se utilizará uno de cada uno por cada servicio ofrecido.
- Realizan acciones similares, pero su diferenciación aporta a la performance.
- Son componentes separadas dentro del sistema.

HandlerAdapters e EventHandlers, ¿por qué?

Separación de responsabilidades:

- HandlerAdapter: contiene la lógica de los mensajes dentro del proxy.
- EventHandler: contiene la lógica de negocios, es decir, lo que el cliente desea obtener.

La separación de responsabilidades entre lógica de negocios e implementación hace que Thorium sea un framework.

ChannelFacade y Queues, ¿por qué?

- ChannelFacade: abstracción de un canal de datos. Útil para no depender de un tipo de canal. De esta forma manejamos solamente ByteBuffers.
- Queues: consecuencia de la asincronía. No se puede asegurar la completitud de un mensaje. Es por esto que se implementó BasicInputQueue.

FutureTasks

- Core del Framework.
- Atención a varios eventos al mismo tiempo, paralelismo.
- Los threads que las atienden están separados de lo que atienden a Acceptors y Dispatchers.

Funcionamiento: Handshake

- Recibimos el request. En primer lugar almacenamos los headers, y el body se almacenará a medida que llegue.
- Una vez hecha la conexión, creamos un sibling con operaciones cruzadas.

Funcionamiento: envío de mensajes

- Totalmente transparente: el manejo de cada transacción es inexistente para los demás.
- Cada HandlerAdapter se encarga de su propia comunicación.
- La lógica de cada comunicación la tiene su EventHandler.
- Cada EventHandler hace a la comunicación única y aplica los filtros correspondientes.

Funcionamiento: envío de mensajes

- HttpEventHandler: Es quien tiene la lógica de negocios del proxy
- HttpRequestMessage y HttpResponseMessage representan la logica de HTTP dentro de HYTTROP.
- HttpMessage (abstracta) es padre de las anteriores.

Funcionamiento: filtros

- Cuando se quiere aplicar un filtro se lo agrega a una lista, la cual se recorre a la hora de aplicarlos.
- Se aplicarán cuando un response sea enviado al cliente a través del HttpEventHandler.

Issues

Administración

- Protocolo orientado a línea
- Basado en protocolo POP3
- Mejora extensibilidad sobre POP3
- Usa el framework Thorium, pero como servidor

Administración

```
C: comando [opciones ...]
s: +OK / -ERR
   .
```

```
facundo@facundo-U56E ~> nc 127.0.0.1 8888
+OK
.
set leet on
+OK
.
```

Administración

- Comandos base:
  - SET: setear configuraciones y filtros
  - GET: obtener datos y monitoreo
  - HELP: obtener una breve ayuda

Comandos soportados

- set l33t [on|off]
- set statistics [on|off|reset]
- get statistics
- [set|get] proxy-port <port>
- [set|get] admin-port <port>
- [set|get] default-origin-server host port

Posibles mejoras y extensiones

- Ayuda general: permitiria listar los comandos disponibles con su respectiva ayuda.

Manejo de estadisticas

- Bytes/tiempo
- Histograma de status codes recibidos.
- Cantidad de conexiones existentes.

Además, el muestreo puede detenerse mediante los comandos, sin eliminar las estadísticas obtenidas hasta el momento.

HYTTROP

gracias!

preguntas?