JABXY

A Jabber Proxy

Alderete, Facundo
Martinez Correa, Facundo

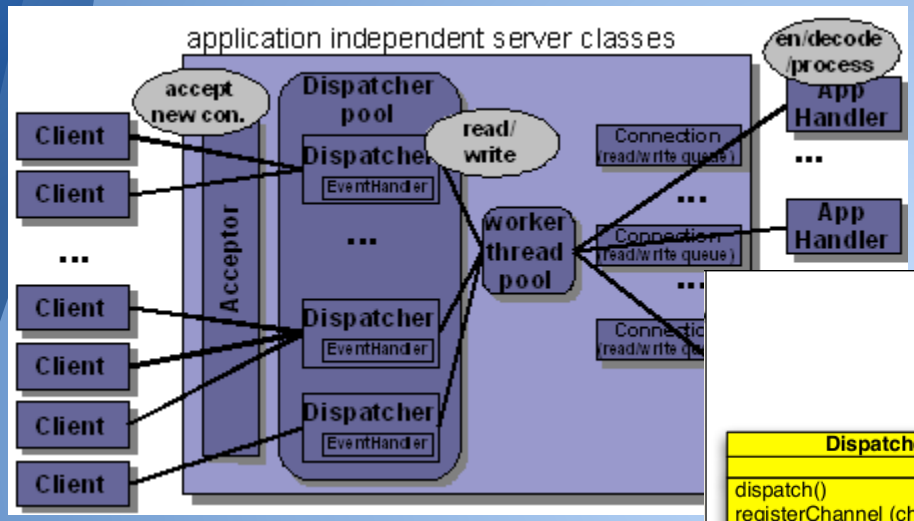
JABXY

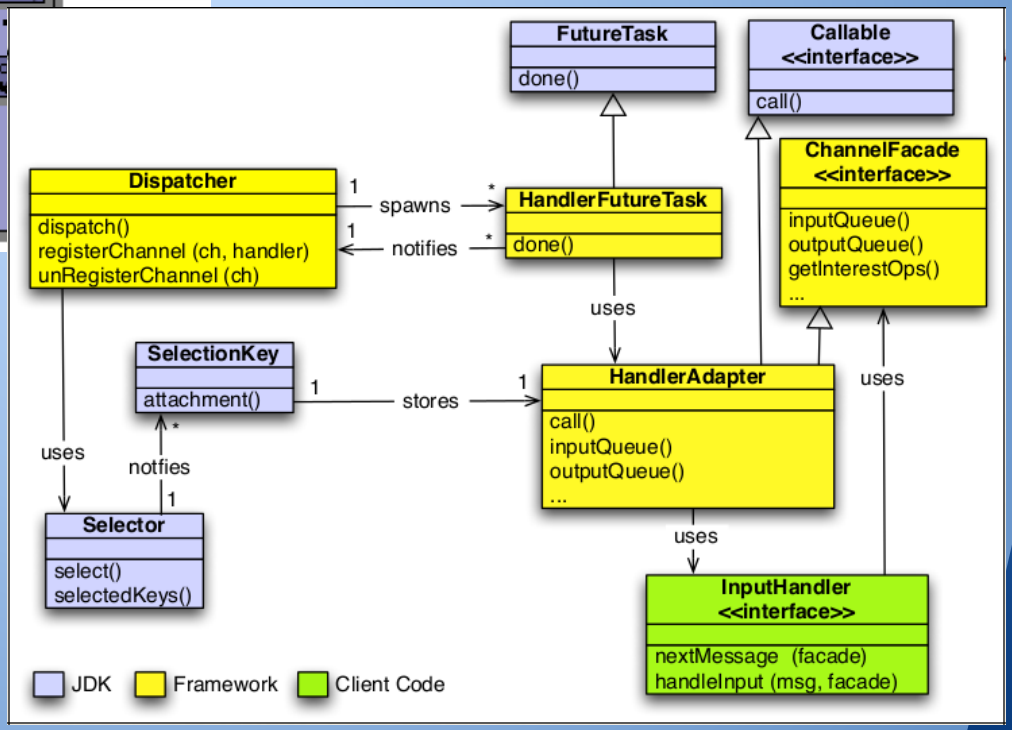
- Diseño
  - Bases y Diseño original
  - Nuestro diseño
  - ¿Por qué?
  - Acceptor y Dispatchers
  - HandlerAdapters e EventHandlers
  - ChannelFacade y Queues
  - FutureTasks
  - Servicios
- Funcionamiento
  - Comenzando la comunicación
  - Recibiendo y enviando mensajes
  - Administración
  - Posibles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

Diseño: Bases y Diseño original

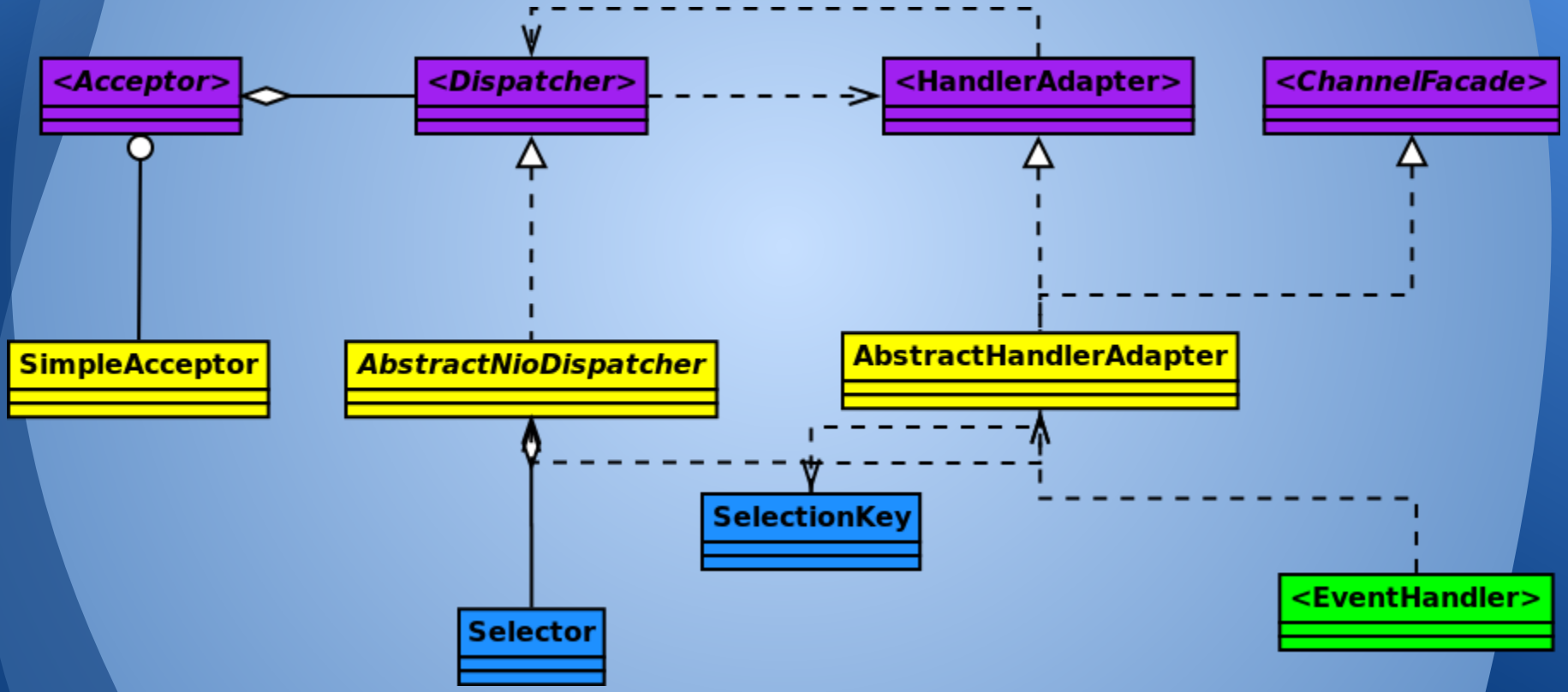
- Basado en Server Mark II de Ron Hitchens. [Java NIO. Hitchens, Ron. O'Reilly]
- Patrón Reactor.
- Es un framework para servidores usando canales NIO no bloqueantes
- Usando RFC: 6120, 6121, 6122

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 esto 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 implemento SAXInputQueue.

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.

servicios

- Solucionan los Crosscutting Concerns: UserService, ConfigurationService, slf4j
- Pertenecen a la lógica de negocio y no están incorporados en el framework

Funcionamiento: Handshake

- Recibimos el mensaje. El "to" es obligatorio. Siempre podemos resolver de primera.
- Una vez hecha la conexión, creamos un sibling con operaciones cruzadas.

Funcionamiento: envío de mensajes

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

Funcionamiento: envío de mensajes

- JabxyEventHandler: Es quien tiene la lógica de negocios del proxy
- JabberProtocol: Es quien tiene la lógica de negocios de Jabber / XMPP
- JabxyUser: Es quien tiene las configuraciones y datos propios de cada conexión

Funcionamiento: filtros

- Se encandenan uno con otros
- La salida puede ser entrada de otro o del mismo JabxyEventHandler

Administración

- Protocolo orientado a línea
- Basado en protocolo POP3
- Mejora extensibilidad sobre POP3
- Usa el framework de Jabxy 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: Sirve para setear configuraciones y filtros
  - GET: Sirve para obtener datos y monitoreo

Posibles mejoras y extensiones

JABXY

Gracias!

Preguntas?