

NIO frameworks

Mikołaj Grajek

Dlaczego używamy grizzly?

- Binarna komunikacja (klient → serwer)
- Ewolucja w komunikacji między modułami
- Problemy wydajnościowe (słabe serwery)
- Rozproszenie obliczeń
- Poszukiwanie prostszych rozwiązań

Po co NIO?

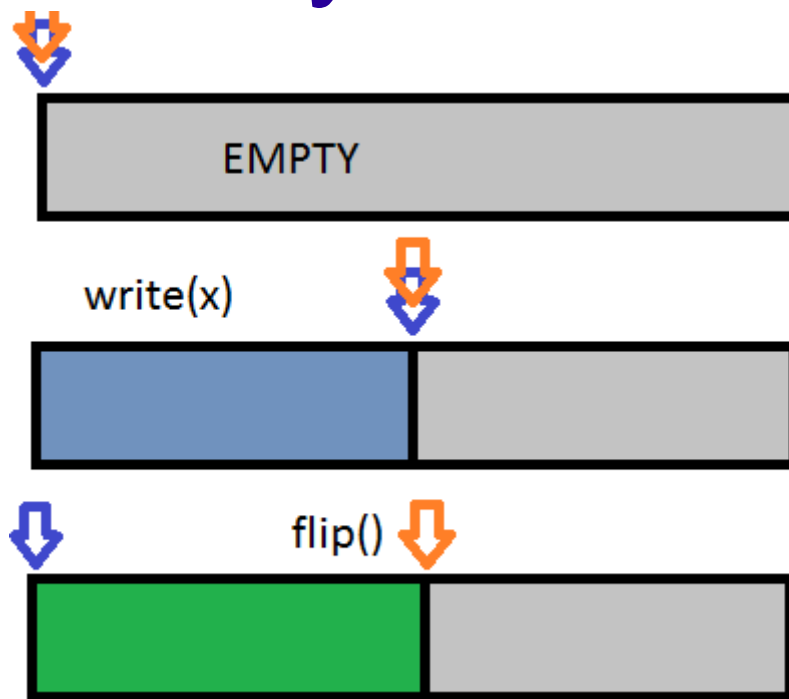
- Wydajność – kopiowanie danych,
- Warstwa sieciowa – nieblokujące operacje
- Przykłady HTTP
 - dekodowanie/pakowanie w locie (streamowanie filmu, upload kilku GB)
 - chat (30tys+ połączeń) - Servlet API 3.0
- Przykłady TCP
 - niestandardowe protokoły (stare systemy)
 - Protocol Buffers (binarne, wiele platform)

New IO

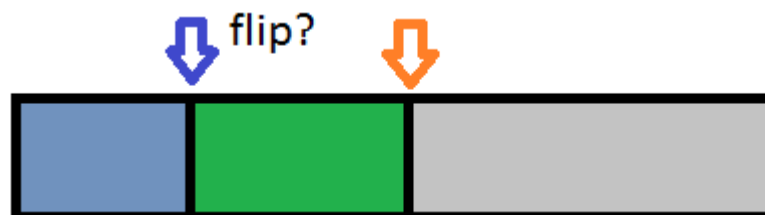
- Podstawowe obiekty:
 - Selector, SelectionKey, Channel
 - Nieobiektywne podejście (opiera się na flagach, wynikach operacji)
- Szybki dostęp do pamięci (ByteBuffer)
 - Flagi stanu (trzeba ciągle pamiętać)

Przykład

- Buffer



- jeden bufor do odczytu/zapisu



Grizzly



- Powstał jako element Glassfish (2005)
- Rozwiązać problemy z którymi borykało się wielu...
- Model zdarzeniowy
- Elementy wspólne:
 - Dekodery, potoki przetwarzania
 - kolejki zapisu/odczytu
 - Pule, fabryki (wątków i kanałów)

Dokumentacja, przykłady...

The best way to learn Grizzly is to read one of the following blogs:

- **Introduction to Grizzly**

- [Introduction to Grizzly Terminology.](#)
- [Building an Echo Server in less than 20 lines.](#)
- [How to create a simple client using Grizzly and get a complimentary server!.](#)
- [Very simple tutorial introduction](#)
- [Connection Pool Management](#)
- [Multi Selector Threads tutorial](#)
- [Writing non blocking SSL client side tutorial](#)
- [Overview of Grizzly 1.5 Architecture](#)
- [Grizzly Connection Caching Tutorial.](#)
- [Grizzly Protocol Parser Tutorial.](#)
- [Asynchronous Write Queue Tutorial.](#)



grizzly

```
GrizzlyWebServer ws = new GrizzlyWebServer(8080);

ServletAdapter sa = new ServletAdapter();
sa.setServletInstance(new StatusServlet());
sa.setProperty("display-name", "status");
sa.setContextPath("/");
ws.addGrizzlyAdapter(sa, new String[] { "/" });

System.out.println("Starting...");

ws.start();
```

- /api/test, /api/test2
- /api/test/ ?
- /api/test/a

jetty

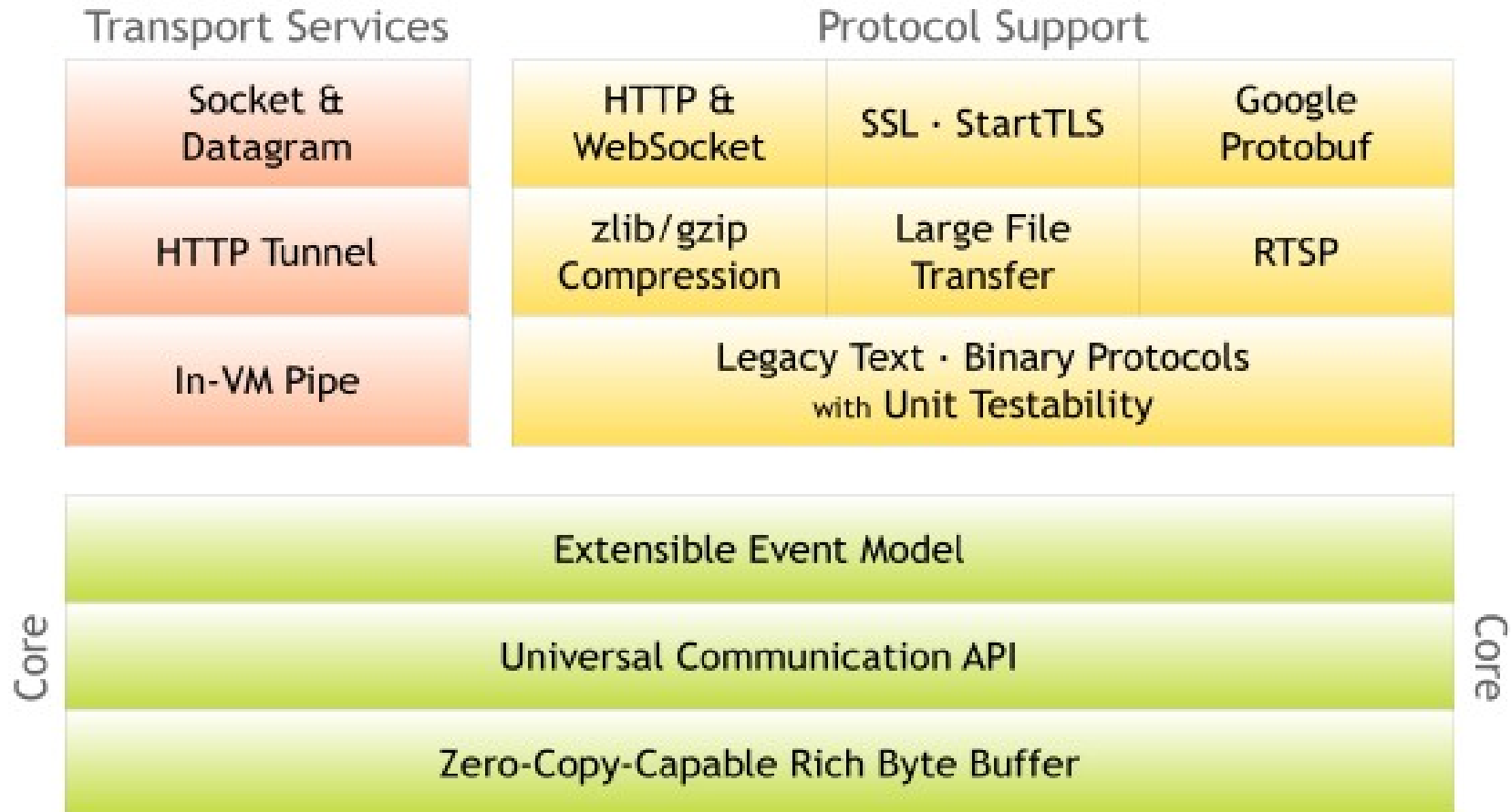
```
Server server = new Server(8080);
try {
    ServletContextHandler context = new ServletContextHandler();
    context.setContextPath("/");
    server.setHandler(context);

    context.addServlet(new ServletHolder(new HelloServlet()), "/*");
    context.addServlet(new ServletHolder(new ShutdownServlet(server)), "/shutdown");

    server.start();
    server.join();
}
```

Alternatywy

- Netty



HTTP szybkość obsługi

	odp [ms]
Jetty-http	35
Jetty-servlet	2
Netty-http	2
Grizzly	202

Nawiązanie połączeń

- ping time=0.281ms
- HTTP 1.1 / Keep-alive

	pierwsze[ms]	kolejne[ms]
jetty-clinet	18	2
netty-http-client	5	2
TCP	3-5	0-1

Test 2 serwery, sieć 100Mb/s

Server/client 160 tyś req.	Przybl. Max r/s
Jetty/jetty	8 051.9
Jetty/jetty (local)	10 892.5
Jetty/netty	26 105.4
Netty/jetty	27 957.4
Netty/netty	54 384.8
TCP	325 203.3
TCP (local)	393 120.4

Przesyłanie danych 1k

- Testy tylko localhost (sieć 100Mb/s)

	req/s
Jetty/jetty	10 905
Jetty/netty	24 038
Netty/jetty *	30 651
Netty/netty	43 573
Tcp	237 154

- *duże wahania

Przesyłanie danych 10k

- *Http – sztywna kolejność żądań (!)
- Jetty (conn per addr, threads)

	Max r/s
Jetty/jetty	5 500
Jetty/netty	709*
Netty/jetty	7 568
Netty/netty	4 520
Tcp	49 505

Dziękuję

