Netty 深入剖析

一 netty简介

1.1 业界使用netty的开源框架

- dubbo
- RocketMQ
- Spark
- ElasticSearch
- Cassandra 开源分布式nosql数据库
- Flink 分布式高性能高可用的流处理框架
- Netty-Socketlo socketlo协议的java服务端实现
- Spring5 使用netty作为http协议框架
- Grpc 谷歌开源的高性能Rpc框架

Dubbo	RocketMQ	Spark
Elasticsearch Ca	ssandra Flink	Netty-SocketIO
Spring5	Play	Grpc

1.2. Netty是什么? 为什么使用netty之后,几乎不用担心性能问题

- 异步事件驱动框架,用于快速开发高性能服务端和客户端
- 封装了JDK底层BIO和NIO模型,提供高度可用的API(提供了非常多的扩展点,使API更加灵活丰富,channelHandler热插拔机制,解放了业务逻辑之外的细节问题,使业务逻辑的热添加和删除变得容易)
- 自带编解码器解决了拆包粘包问题,用户只用关心业务逻辑
- 精心设计的reactor线程模型支持高并发海量连接(为什么netty只使用了少量的线程,就能管理成千上万甚至几十万的连接)
- 自带各种协议栈让你处理任何一种通用协议都几乎不用亲自动手

1.3. 为什么学netty

- 各大开源框架选择netty作为底层通信框架
- 更好的使用, 少走弯路
- 单机连接数上不去? 性能遇到瓶颈? 如何调优

- 详解reactor线程模型,实践中举一反三
- 庞大的项目是如何组织的,设计模式,体验优秀的设计
- 阅读源码 -- 可以作为第一个深入研究的开源框架

1.4. 目标

- 掌握netty底层核心原理,解决各类问题,深度调优
- 给netty官方提issue
- 实现一个简易版的netty
- 开启阅读源码之旅
- 加速掌握基于netty的各类中间件

1.5. 技术储备

- java基础,多线程
- TCP原理, NIO

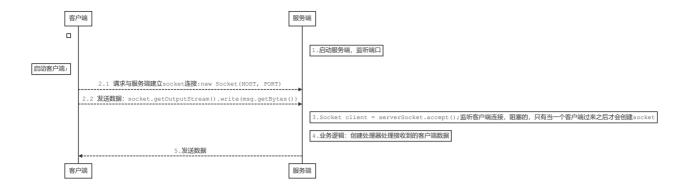
二 netty基本组件

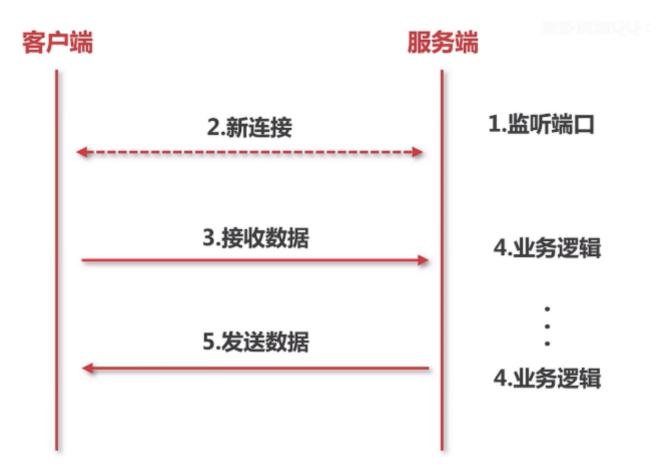
包括: NioEventLoop (发动机:起了两种类型的线程), Channel (对连接的封装,数据读写), ByteBuf (数据流), Pipeline (逻辑处理链), ChannelHandler (逻辑)

Netty基本组件

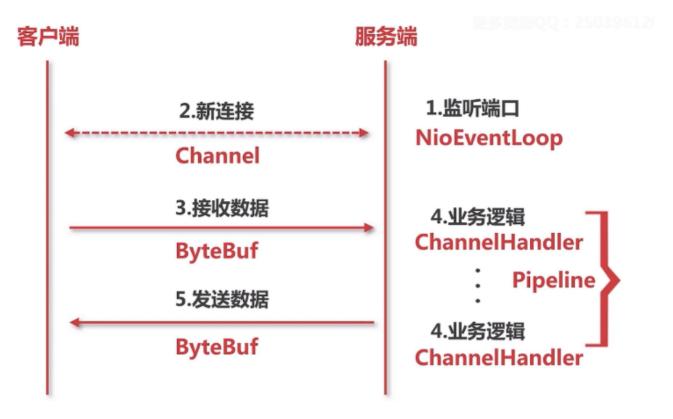


2.1 不使用netty情况下,模拟传统的客户端与服务端通信





- 2.1.1 监听端口实际包含两层含义: 对应两个while循环
 - a) server不断的在某个端口上监听新用户的连接
 - b)新用户的连接建立完成后,在对应的端口上不断的监听新连接的数据 netty实现:



2.2 netty - NioEventLoop

对应socket编程的线程



NioEventLoop: nio事件循环

2.2.1 新连接的接入

2.2.2 当前存在的连接上数据流的读写

2.3 netty - channel

channel 定义:

- * A nexus to a network socket or a component which is capable of I/O
- * operations such as read, write, connect, and bind.

对应socket编程的socket

端口上监听到的新用户的连接



io.netty.channel.nio.AbstractNioMessageChannel#doReadMessages

SocketChannel ch = javaChannel().accept();

java IO编程模型 -- 当作socket处理

NIO编程模型 -- socketChannel

netty -- 封装成自定义的channel

基于channel,一系列的读写都可以在这个连接上操作,其实就是对socket的抽象

2.4 netty - ByteBuf



服务端接受用户的数据流的载体都是基于ByteBuf,封装了很多api可以与底层的连接的数据流通信

2.5 netty - channelHandler

服务端处理业务逻辑的处理器



io.netty.channel.DefaultChannelPipeline#addFirst(io.netty.channel.ChannelHandler...) 通过ChannelPipeline可以动态的添加channelHandler

C > DefaultChannelPipeline

- m P DefaultChannelPipeline(Channel)
- 🛅 🖫 addAfter(EventExecutorGroup, String, String, ChannelHandler): ChannelPipeline 🛚
- 🛅 🖫 addAfter(String, String, ChannelHandler): ChannelPipeline †ChannelPipeline
- 📠 🔒 addAfter0(AbstractChannelHandlerContext, AbstractChannelHandlerContext): voi
- 🛅 🖫 addBefore(EventExecutorGroup, String, String, ChannelHandler): ChannelPipeline
- 🛅 🖫 addBefore(String, String, ChannelHandler): ChannelPipeline †ChannelPipeline
- 📠 🔒 addBefore0(AbstractChannelHandlerContext, AbstractChannelHandlerContext): v
- 🛅 🖫 addFirst(ChannelHandler...): ChannelPipeline †ChannelPipeline
- 🛅 🖫 addFirst(EventExecutorGroup, ChannelHandler...): ChannelPipeline †ChannelPipel
- 🛅 🖫 addFirst(EventExecutorGroup, String, ChannelHandler): ChannelPipeline †Channe
- 🛅 🖫 addFirst(String, ChannelHandler): ChannelPipeline †ChannelPipeline
- 🎟 🔒 addFirst0(AbstractChannelHandlerContext): void
- 🛅 🖫 addLast(ChannelHandler...): ChannelPipeline †ChannelPipeline
- 🛅 🖫 addLast(EventExecutorGroup, ChannelHandler...): ChannelPipeline †ChannelPipeli
- 🛅 🖫 addLast(EventExecutorGroup, String, ChannelHandler): ChannelPipeline †Channe
- 🛅 🖫 addLast(String, ChannelHandler): ChannelPipeline †ChannelPipeline
- m 🔒 addLast0(AbstractChannelHandlerContext): void
- 🛅 🖫 bind(SocketAddress): ChannelFuture ↑ChannelOutboundInvoker

```
@Override
public final ChannelPipeline addFirst(ChannelHandler... handlers) {
   return addFirst(null, handlers);
}
```

实际生产环境下,客户端与服务端的通信的时候都很复杂,

一般都需要定义二进制的协议,对二进制协议的数据进行数据包的拆分,对不同类型的协议数据包转换成不同的 java对象并作不同的处理。

netty把每一个处理过程都当作ChannelHandler。

将不同的处理过程交给不同的channelHandler处理。

用户可以自定义channelHandler。

例如:数据包分包器

2.6 netty - Pipeline - 逻辑链

netty什么时候将Pipeline加入到每一个客户端连接的处理过程的。

```
protected AbstractChannel(Channel parent, ChannelId id) {
   this.parent = parent;
   this.id = id;
   unsafe = newUnsafe();
   pipeline = newChannelPipeline();
}
```



三. netty服务端启动

- 3.1 思考:服务端的socket在哪里初始化?在哪里accept连接?
- 3.2 netty服务端启动的四个过程
- 3.2.1 创建服务端channel

```
ChannelFuture channelFuture = bootstrap.bind(8888).sync();
```

```
private ChannelFuture doBind(final SocketAddress localAddress) {
   final ChannelFuture regFuture = initAndRegister();
   final Channel channel = regFuture.channel();
   ....
}
```

```
final ChannelFuture initAndRegister() {
   Channel channel = null;
   try {
      channel = channelFactory.newChannel();
      init(channel);
   } catch (Throwable t) {
   .....
}
```

```
@Override
public T newChannel() {
    try {
        // 这里的clazz指什么? --> NioServerSocketChannel.class
        return clazz.newInstance();//反射
    } catch (Throwable t) {
        throw new ChannelException("Unable to create Channel from class " + clazz, t);
    }
}
```

创建服务端Channel

bind() [用户代码入口]

initAndRegister() [初始化并注册]

newChannel()[创建服务端channel]

NioServerSocketChannel**如何构造的

反射创建服务端Channel

newSocket() [通过jdk来创建底层jdk channel]

NioServerSocketChannelConfig() [tcp参数配置类]

AbstractNioChannel()

configureBlocking(false) [阻塞模式]

AbstractChannel() [创建id,unsafe,pipeline]

```
public NioServerSocketChannel() {
   this(newSocket(DEFAULT_SELECTOR_PROVIDER));
}
```

```
public NioServerSocketChannel(ServerSocketChannel channel) {
    super(null, channel, SelectionKey.OP_ACCEPT);
    config = new NioServerSocketChannelConfig(this, javaChannel().socket());
}
```

```
io.netty.channel.nio.AbstractNioChannel#AbstractNioChannel
protected AbstractNioChannel(Channel parent, SelectableChannel ch, int readInterestOp) {
    super(parent);
    this.ch = ch;
    this.readInterestOp = readInterestOp;
    try {
        ch.configureBlocking(false);
    } catch (IOException e) {
        ...
    }
}
```

```
io.netty.channel.AbstractChannel#AbstractChannel(io.netty.channel.Channel)
protected AbstractChannel(Channel parent) {
   this.parent = parent;
   id = newId();
   unsafe = newUnsafe();
   pipeline = newChannelPipeline();
}
```

3.2.2 初始化服务端channel

初始化服务端Channel

init() [初始化入口]

set ChannelOptions, ChannelAttrs

set ChildOptions, ChildAttrs

config handler [配置服务端pipeline]

add ServerBootstrapAcceptor [添加连接器]

- 3.2.3 注册事件轮询器selector
- 3.2.4 端口绑定