# 高性能NIO框架Mina

# Java NIO 和IO的区别

**IO NIO**

面向流 面向缓冲

阻塞IO 非阻塞IO

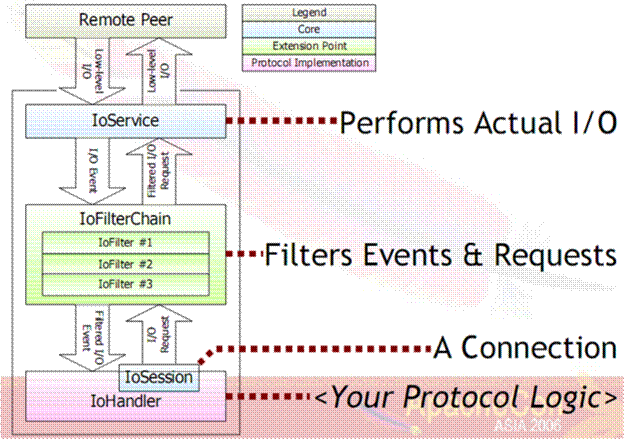
无 选择器

# Mina简介

Apache组织应用程序(Multipurpose Infrastructure for Network Applications) 是 Apache 组织一个较新的项目，它为开发高性能和高可用性的网络应用程序提供了非常便利的框架。

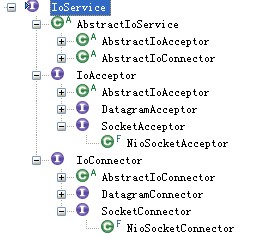
# Mina组件

## Mina核心类和处理流程



### IoService

IoService负责底层IO操作，他的好处主要在于隐藏底层IO操作，方便使用；基于事件机制将同步IO转化成异步IO。



### IoFilterChain

IoFilterChain主要负责数据包的处理工作，包括编码、解码等功能。其主要思想是——与业务逻辑代码分离。当然IofilterChain也是处理流程的扩展点，包括为处理流程添加线程池。

### IoHandler

IoHandler主要实现业务逻辑（调用业务逻辑接口）。

### IoSession

IoSession是对底层连接的封装，一个IoSession对应一个底层的IO连接。

发送数据其实是个异步的过程。发送的操作首先会逆向穿过IoFilterChain，到达IoService。但IoService上并不会直接调用底层IO接口来将数据发送出去，而是会将该次调用封装成一个WriteRequest，放入session的writeRequestQueue中，最后由IoProcessor线程统一调度flush出去。所以发送操作并不会引起上层调用线程的阻塞。

## 代码示例(TCP)

### 客户端

#### Client.java

**public** **class** Client {

**private** **static** **final** String ***HOSTNAME*** = "localhost";

**private** **static** **final** **int** ***PORT*** = 8083;

**private** **static** **final** **long** ***CONNECT\_TIMEOUT*** = 30 \* 1000L; // 30 seconds

// Set this to false to use object serialization instead of custom codec.

**private** **static** **final** **boolean** ***USE\_CUSTOM\_CODEC*** = **true**;

**public** **static** **void** main(String[] args) **throws** Throwable {

**if** (args.length == 0) {

System.***out***.println("Please specify the list of any integers");

**return**;

}

// prepare values to sum up

**int**[] values = **new** **int**[args.length];

**for** (**int** i = 0; i < args.length; i++) {

values[i] = Integer.*parseInt*(args[i]);

}

NioSocketConnector connector = **new** NioSocketConnector();

// Configure the service.

connector.setConnectTimeoutMillis(***CONNECT\_TIMEOUT***);

**if** (***USE\_CUSTOM\_CODEC***) {

connector.getFilterChain().addLast("codec", **new** ProtocolCodecFilter(**new** SumUpProtocolCodecFactory(**false**)));

} **else** {

connector.getFilterChain().addLast("codec", **new** ProtocolCodecFilter(**new** ObjectSerializationCodecFactory()));

}

connector.getFilterChain().addLast("logger", **new** LoggingFilter());

connector.setHandler(**new** ClientSessionHandler(values));

IoSession session;

**for** (;;) {

**try** {

ConnectFuture future = connector.connect(**new** InetSocketAddress(***HOSTNAME***, ***PORT***));

future.awaitUninterruptibly();

session = future.getSession();

**break**;

} **catch** (RuntimeIoException e) {

System.***err***.println("Failed to connect.");

e.printStackTrace();

Thread.*sleep*(5000);

}

}

// wait until the summation is done

session.getCloseFuture().awaitUninterruptibly();

connector.dispose();

}

}

#### ClientSessionHandler.java

**public** **class** ClientSessionHandler **extends** IoHandlerAdapter {

**private** **final** **static** Logger ***LOGGER*** = LoggerFactory.*getLogger*(ClientSessionHandler.**class**);

**private** **final** **int**[] values;

**private** **boolean** finished;

**public** ClientSessionHandler(**int**[] values) {

**this**.values = values;

}

**public** **boolean** isFinished() {

**return** finished;

}

@Override

**public** **void** sessionOpened(IoSession session) {

// send summation requests

**for** (**int** i = 0; i < values.length; i++) {

AddMessage m = **new** AddMessage();

m.setSequence(i);

m.setValue(values[i]);

session.write(m);

}

}

@Override

**public** **void** messageReceived(IoSession session, Object message) {

// server only sends ResultMessage. otherwise, we will have to identify

// its type using instanceof operator.

ResultMessage rm = (ResultMessage) message;

**if** (rm.isOk()) {

// server returned OK code.

// if received the result message which has the last sequence

// number,

// it is time to disconnect.

**if** (rm.getSequence() == values.length - 1) {

// print the sum and disconnect.

***LOGGER***.info("The sum: " + rm.getValue());

session.closeNow();

finished = **true**;

}

} **else** {

// seever returned error code because of overflow, etc.

***LOGGER***.warn("Server error, disconnecting...");

session.closeNow();

finished = **true**;

}

}

@Override

**public** **void** exceptionCaught(IoSession session, Throwable cause) {

session.closeNow();

}

}

### 服务端

#### Server.java

**public** **class** Server {

**private** **static** **final** **int** ***SERVER\_PORT*** = 8083;

// Set this to false to use object serialization instead of custom codec.

**private** **static** **final** **boolean** ***USE\_CUSTOM\_CODEC*** = **true**;

**public** **static** **void** main(String[] args) **throws** Throwable {

NioSocketAcceptor acceptor = **new** NioSocketAcceptor();

// Prepare the service configuration.

**if** (***USE\_CUSTOM\_CODEC***) {

acceptor.getFilterChain().addLast("codec", **new** ProtocolCodecFilter(**new** SumUpProtocolCodecFactory(**true**)));

} **else** {

acceptor.getFilterChain().addLast("codec", **new** ProtocolCodecFilter(**new** ObjectSerializationCodecFactory()));

}

acceptor.getFilterChain().addLast("logger", **new** LoggingFilter());

acceptor.setHandler(**new** ServerSessionHandler());

acceptor.bind(**new** InetSocketAddress(***SERVER\_PORT***));

System.***out***.println("Listening on port " + ***SERVER\_PORT***);

}

}

#### ServerSessionHandler.java

**public** **class** ServerSessionHandler **extends** IoHandlerAdapter {

**private** **static** **final** String ***SUM\_KEY*** = "sum";

**private** **final** **static** Logger ***LOGGER*** = LoggerFactory.*getLogger*(ServerSessionHandler.**class**);

@Override

**public** **void** sessionOpened(IoSession session) {

// set idle time to 60 seconds

session.getConfig().setIdleTime(IdleStatus.***BOTH\_IDLE***, 60);

// initial sum is zero

session.setAttribute(***SUM\_KEY***, Integer.*valueOf*(0));

}

@Override

**public** **void** messageReceived(IoSession session, Object message) {

// client only sends AddMessage. otherwise, we will have to identify

// its type using instanceof operator.

AddMessage am = (AddMessage) message;

// add the value to the current sum.

**int** sum = ((Integer) session.getAttribute(***SUM\_KEY***)).intValue();

**int** value = am.getValue();

**long** expectedSum = (**long**) sum + value;

**if** (expectedSum > Integer.***MAX\_VALUE*** || expectedSum < Integer.***MIN\_VALUE***) {

// if the sum overflows or underflows, return error message

ResultMessage rm = **new** ResultMessage();

rm.setSequence(am.getSequence()); // copy sequence

rm.setOk(**false**);

session.write(rm);

} **else** {

// sum up

sum = (**int**) expectedSum;

session.setAttribute(***SUM\_KEY***, Integer.*valueOf*(sum));

// return the result message

ResultMessage rm = **new** ResultMessage();

rm.setSequence(am.getSequence()); // copy sequence

rm.setOk(**true**);

rm.setValue(sum);

System.***out***.println("sum:" + sum);

session.write(rm);

}

}

@Override

**public** **void** sessionIdle(IoSession session, IdleStatus status) {

***LOGGER***.info("Disconnecting the idle.");

// disconnect an idle client

session.closeNow();

}

@Override

**public** **void** exceptionCaught(IoSession session, Throwable cause) {

// close the connection on exceptional situation

session.closeNow();

}

}

### 编码/解码

#### SumUpProtocolCodecFactory.java

**public** **class** SumUpProtocolCodecFactory **extends** DemuxingProtocolCodecFactory {

**public** SumUpProtocolCodecFactory(**boolean** server) {

**if** (server) {

**super**.addMessageDecoder(AddMessageDecoder.**class**);

**super**.addMessageEncoder(ResultMessage.**class**, ResultMessageEncoder.**class**);

} **else** // Client

{

**super**.addMessageEncoder(AddMessage.**class**, AddMessageEncoder.**class**);

**super**.addMessageDecoder(ResultMessageDecoder.**class**);

}

}

}

#### AddMessageEncoder.java

**public** **class** AddMessageEncoder<T **extends** AddMessage> **extends** AbstractMessageEncoder<T> {

**public** AddMessageEncoder() {

**super**(Constants.***ADD***);

}

@Override

**protected** **void** encodeBody(IoSession session, T message, IoBuffer out) {

out.putInt(message.getValue());

}

**public** **void** dispose() **throws** Exception {

}

}

#### AddMessageDecoder.java

**public** **class** AddMessageDecoder **extends** AbstractMessageDecoder {

**public** AddMessageDecoder() {

**super**(Constants.***ADD***);

}

@Override

**protected** AbstractMessage decodeBody(IoSession session, IoBuffer in) {

**if** (in.remaining() < Constants.***ADD\_BODY\_LEN***) {

**return** **null**;

}

AddMessage m = **new** AddMessage();

m.setValue(in.getInt());

**return** m;

}

**public** **void** finishDecode(IoSession session, ProtocolDecoderOutput out)

**throws** Exception {

}

}

#### ResultMessageEncoder.java

**public** **class** ResultMessageEncoder<T **extends** ResultMessage> **extends** AbstractMessageEncoder<T> {

**public** ResultMessageEncoder() {

**super**(Constants.***RESULT***);

}

@Override

**protected** **void** encodeBody(IoSession session, T message, IoBuffer out) {

**if** (message.isOk()) {

out.putShort((**short**) Constants.***RESULT\_OK***);

out.putInt(message.getValue());

} **else** {

out.putShort((**short**) Constants.***RESULT\_ERROR***);

}

}

**public** **void** dispose() **throws** Exception {

}

}

#### ResultMessageDecoder.java

**public** **class** ResultMessageDecoder **extends** AbstractMessageDecoder {

**private** **int** code;

**private** **boolean** readCode;

**public** ResultMessageDecoder() {

**super**(Constants.***RESULT***);

}

@Override

**protected** AbstractMessage decodeBody(IoSession session, IoBuffer in) {

**if** (!readCode) {

**if** (in.remaining() < Constants.***RESULT\_CODE\_LEN***) {

**return** **null**; // Need more data.

}

code = in.getShort();

readCode = **true**;

}

**if** (code == Constants.***RESULT\_OK***) {

**if** (in.remaining() < Constants.***RESULT\_VALUE\_LEN***) {

**return** **null**;

}

ResultMessage m = **new** ResultMessage();

m.setOk(**true**);

m.setValue(in.getInt());

readCode = **false**;

**return** m;

} **else** {

ResultMessage m = **new** ResultMessage();

m.setOk(**false**);

readCode = **false**;

**return** m;

}

}

**public** **void** finishDecode(IoSession session, ProtocolDecoderOutput out)

**throws** Exception {

}

}

#### Constants.java

**public** **class** Constants {

**public** **static** **final** **int** ***TYPE\_LEN*** = 2;

**public** **static** **final** **int** ***SEQUENCE\_LEN*** = 4;

**public** **static** **final** **int** ***HEADER\_LEN*** = ***TYPE\_LEN*** + ***SEQUENCE\_LEN***;

**public** **static** **final** **int** ***BODY\_LEN*** = 12;

**public** **static** **final** **int** ***RESULT*** = 0;

**public** **static** **final** **int** ***ADD*** = 1;

**public** **static** **final** **int** ***RESULT\_CODE\_LEN*** = 2;

**public** **static** **final** **int** ***RESULT\_VALUE\_LEN*** = 4;

**public** **static** **final** **int** ***ADD\_BODY\_LEN*** = 4;

**public** **static** **final** **int** ***RESULT\_OK*** = 0;

**public** **static** **final** **int** ***RESULT\_ERROR*** = 1;

**private** Constants() {

}

}

### Message

#### AbstractMessage.java

**public** **abstract** **class** AbstractMessage **implements** Serializable {

**static** **final** **long** ***serialVersionUID*** = 1L;

**private** **int** sequence;

**public** **int** getSequence() {

**return** sequence;

}

**public** **void** setSequence(**int** sequence) {

**this**.sequence = sequence;

}

}

#### AddMessage.java

**public** **class** AddMessage **extends** AbstractMessage {

**private** **static** **final** **long** ***serialVersionUID*** = -940833727168119141L;

**private** **int** value;

**public** AddMessage() {

}

**public** **int** getValue() {

**return** value;

}

**public** **void** setValue(**int** value) {

**this**.value = value;

}

@Override

**public** String toString() {

// it is a good practice to create toString() method on message classes.

**return** getSequence() + ":ADD(" + value + ')';

}

}

#### ResultMessage.java

**public** **class** ResultMessage **extends** AbstractMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 7371210248110219946L;

**private** **boolean** ok;

**private** **int** value;

**public** ResultMessage() {

}

**public** **boolean** isOk() {

**return** ok;

}

**public** **void** setOk(**boolean** ok) {

**this**.ok = ok;

}

**public** **int** getValue() {

**return** value;

}

**public** **void** setValue(**int** value) {

**this**.value = value;

}

@Override

**public** String toString() {

**if** (ok) {

**return** getSequence() + ":RESULT(" + value + ')';

} **else** {

**return** getSequence() + ":RESULT(ERROR)";

}

}

}

## 其他

官方demo下载地址:

<http://mina.apache.org/mina-project/downloads.html>