Chapter 16. 향상된 서버 만들기

Originally made by Prof. Hanku Lee
Modified by Mingyu Lim

Collaborative Computing Systems Lab.
School of Internet & Multimedia Engineering
Konkuk University, Seoul, Korea

1. 효율적인 메모리 사용

❑ 15장 서버 프로그램 메모리 / 가비지 미고려

```
private void read(SelectionKey key) {
    // SelectionKey 로부터 소켓채널을 얻어옴.
    SocketChannel sc = (SocketChannel) key.channel();
    // 바이트버퍼 생성.
    ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
    try {
        // 요청한 클라이언트의 소켓채널로부터 데이터를 읽어들임.
        int read = sc.read(buffer);
        info(read + " byte 를 읽었습니다.");
    } catch (IOException e) {
        try {
            sc.close();
        } catch (IOException e1) {
        }

        removeUser(sc);

        info(sc.toString() + " 클라언트가 접속을 해제했습니다.");
    }

    try {
        // 클라이언트가 보낸 메세지를 채팅방 안에 모든 사용자에게 브로드캐스트해줌.
        broadcast(buffer);
    } catch (IOException e) {
        log(Level.WARNING, "SimpleChatServer.broadcast()", e);
    }

    // 버퍼 메모리를 해제해줌.
    clearBuffer(buffer);
}

private void clearBuffer(ByteBuffer buffer) {
    if (buffer != null) {
        buffer.clear();
        buffer = null;
    }
}
```

1. 효율적인 메모리 사용(계속)

❑ 15장 서버 프로그램 메모리 / 가비지 미고려

- ——부분은 클라이언트들이 보낸 메시지 처리때마다 ByteBuffer 생성해서 사용하고 사용이 끝나면 버퍼를 null로 만들어 가비지 콜렉션 대상으로 만듦

- 다이렉트 ByteBuffer를 생성하는것은 상당히 느린 작업

- 이부분에서 재사용 필요
  : ByteBufferPool 을 사용하면 효율적으로 메모리 재사용 가능
  : 매번 ByteBuffer 객체의 생성, 해제에 따른 가비지 생성 부담 줄임

2. 비효율적인 데이터 전송에 대한 고려

❑ 비효율적인 데이터 전송

- 서버 동작

(클라이언트가 100byte인 메시지 전송, 그 중 1Byte 만이 먼저 서버에 도착)

1. 해당 클라이언트 읽기 준비 완료되었다고 SelectionKey의 Ready Set 에 표시
2. Selector의 select() 메소드 호출되고 해당 이벤트를 처리 / 이벤트 구분을 위해 isAcceptable(), isReadable() 등의 메소드 호출
3. read 이벤트에 따라 클라이언트가 보내 메시지 읽음(1byte 의미 없음) 따라서 다시 1 번부터 반복적으로 실행되는 과정 거친후 남은 메시지 읽음

: 의미없이 도착한 1byte 때문에 서버는 같은 과정 두번 거침
(서버의 성능 저하 현상)

: 쓰기 부분도 위와 같은 현상

=> 해결방안 (무조건 read () 두번 호출하게 만듬)

: 쓰기경우는 compact() / hasRemaining() 메소드 이용

3. 동시성을 이용한 성능 극대화

❑ Accept, Process 의 분리

```
SelectionKey key = (SelectionKey) it.next();
if (key.isAcceptable()) {
    // 서버소켓채널에 클라이언트가 접속을 시도한 경우.
    accept(key);
} else if (key.isReadable()) {
    // 이미 연결된 클라이언트가 메세지를 보낸 경우.
    read(key);
}
```

- 클라이언트 접속 여부 / 클라이언트 메시지 전송 판단하는 코드

- accept() 와 같이 클라이언트와 접속을 맺는 것은 상당히 느린 작업
  (한명 접속 0.01초 / 만명 접속시 1분 30초 소요)

- accept에 해당하는 부분 별도의 스레드와 셀렉터로 분리 시켜
  클라이언트의 접속을 전담하게 만드는것 필요

- Connect, Read, Write 도 각각 별도의 처리로직으로 분리하는것이
  대용량 서버에 유리

3. 동시성을 이용한 성능 극대화

❑ SelectorPool 사용

- Accept 부분 셀렉터로 분리한후 접속 자체가 빈속 하다면 Accept를 수행하는 셀렉터를 n개로 사용하는것이 효과적

- Read를 수행하는 셀렉터 하나존재, 동시접속자가 만명일 경우 병목현상 발생 : Read 를 수행하는 셀렉터를 n개로 사용하는것이 효과적

- 병목부분이 될수 있는 셀렉터 부분에 사용빈도나 관리 채널의 개수에 따라 일정한 한도 안에서 자동적으로 셀렉터의 개수가 늘거나 줄수있는 SelectorPool를 사용하면 서버성능 향상에 효율

3. 동시성을 이용한 성능 극대화

❑ ThreadPool 사용

```
private void accept(SelectionKey key) {
    ServerSocketChannel server = (ServerSocketChannel) key.channel();
    SocketChannel sc;
    try {
        // 서버소켓채널의 accept() 메소드로 서버소켓 생성.
        sc = server.accept();
        // 생성된 소켓채널을 비블록킹 및 읽기 모드로 실렉터에 등록.
        registerChannel(selector, sc, SelectionKey.OP_READ);
        info(sc.toString() + " 클라이언트가 접속했습니다.");
    } catch (ClosedChannelException e) {
        log(Level.WARNING, "SimpleChatServer.accept()", e);
    } catch (IOException e) {
        log(Level.WARNING, "SimpleChatServer.accept()", e);
    }
}

private void read(SelectionKey key) {
    // SelectionKey 로부터 소켓채널을 얻어옴.
    SocketChannel sc = (SocketChannel) key.channel();
    // 바이트버퍼 생성.
    ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
    try {
        // 요청한 클라이언트의 소켓채널로부터 데이터를 읽어들임.
        int read = sc.read(buffer);
        info(read + " byte 를 읽었습니다.");
    } catch (IOException e) {
        try {
            sc.close();
        } catch (IOException e1) {
        }

        removeUser(sc);

        info(sc.toString() + " 클라언트가 접속을 해제했습니다.");
    }

    try {
        // 클라이언트가 보낸 메세지를 채팅방 안에 모든 사용자에게 브로드캐스트해줌.
        broadcast(buffer);
    } catch (IOException e) {
        log(Level.WARNING, "SimpleChatServer.broadcast()", e);
    }

    // 버퍼 메모리를 해제해줌.
    clearBuffer(buffer);
}
```

3. 동시성을 이용한 성능 극대화

❑ ThreadPool 사용

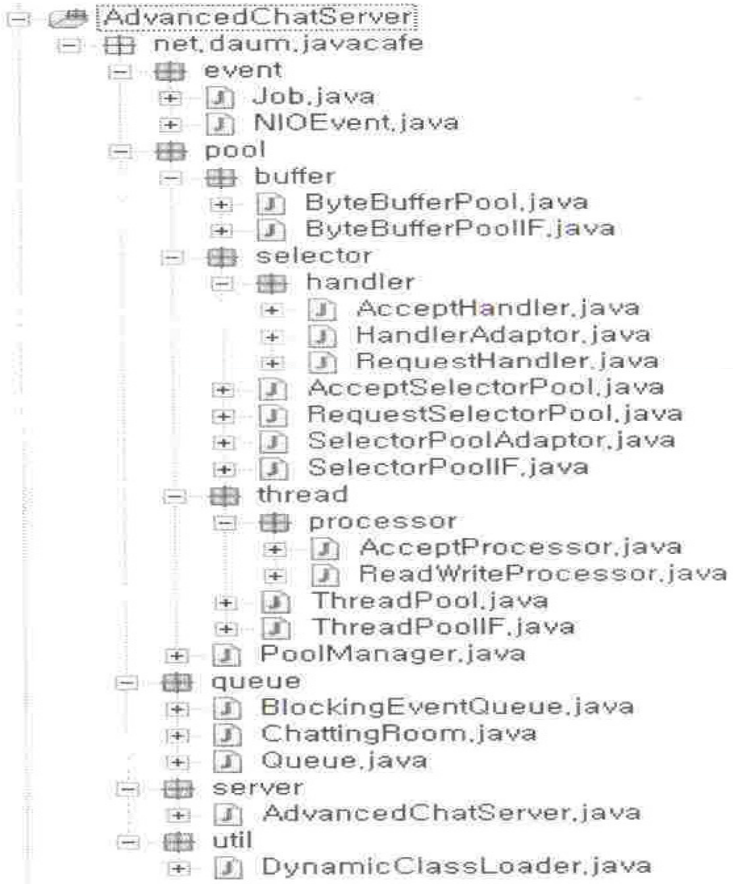
- Accept / Read 요청을 싱글 스레드로 처리

- ThreadPool 을 사용해서 스레드 여러 개로 동시에 처리하는것이 효율적

- Accept / Processe를 처리하는 것으로 나눠서 별도의 독립적인 ThreadPool 운영되게 하는 것이 좋다.

4. 향상된 서버 만들기

❑ ByteBufferPool, ThreadPool, Accept 분리 적용한 서버
(15장에 만들어본 서버 업그레이드)



[그림 16-1] AdvancedChatServer 패키지 구조

4. 향상된 서버 만들기

❑ ByteBufferPool, ThreadPool, Accept 분리 적용한 서버

- event

: NIOEvent 인터페이스와 이벤트를 처리하기 위해 사용할 Job 클래스존재
: AdvancedChatServer 는 두가지 이벤트 처리
1. 클라이언트가 서버로 Accept 하는 것
2. 클라이언트가 보낸 메시지를 Read 하는것

- pool

: 기존 서버 보다 뛰어난 성능을 보일수 있는 핵심적 역할을 하는 클래스
(ByteBufferPool, ThreadPool, SelectorPool)
: ByteBufferPool => 14장 채널 파일 매핑 참조
: ThreadPool => accept, read, write 처리
: SelectorPool => accept, read 존재 / 준비된 채널들을 빠르게 처리

4. 향상된 서버 만들기

❑ ByteBufferPool, ThreadPool, Accept 분리 적용한 서버

- queue

: 셀렉터에서 발생한 이벤트를 해당 형식의 속성 갖는 Job 객체로 만들고 이 패키의 queue 객체에 저장

: ThreadPool 객체의 각 스레드들은 queue 객체에서 해당 Job 객체를 꺼내서 처리

: 서버에 접속한 전체 사용자들을 등록하기 위해서 ChattingRom 이라는 Vector를 상속한 클래스 만들어 사용

- Server

: AdvancedChatServer 서버를 실행시키는 클래스가 위치한 클래스

4. 향상된 서버 만들기

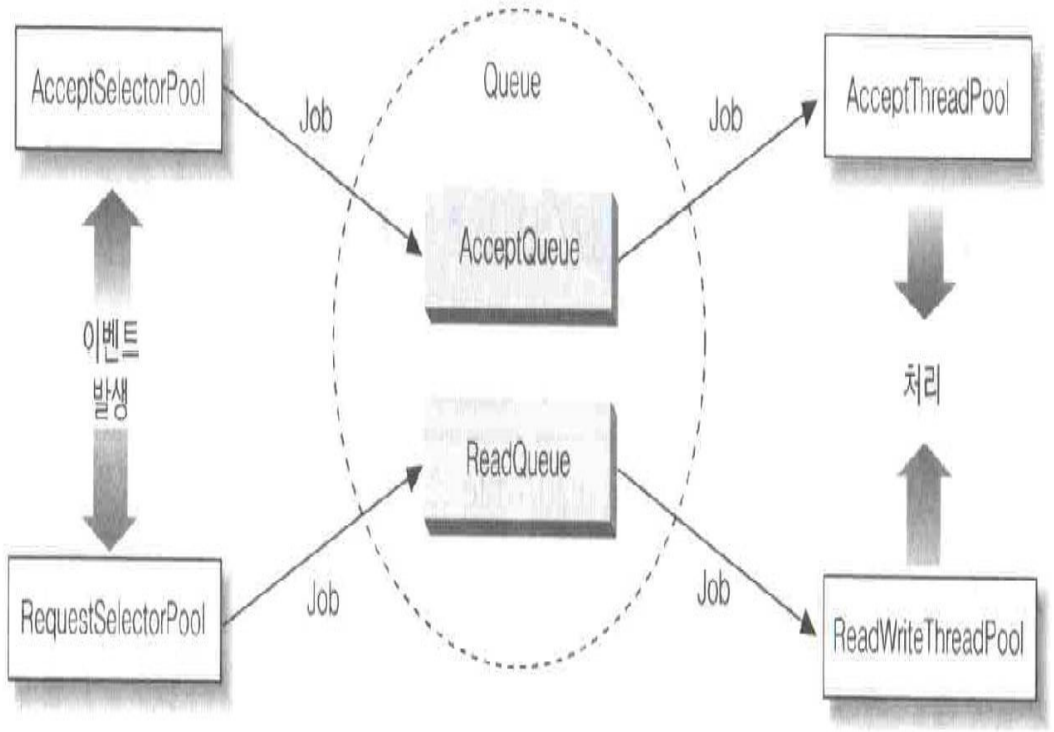
❑ ByteBufferPool, ThreadPool, Accept 분리 적용한 서버

- util

: ThreadPool에서 사용하는 ClassLoader 클래스가 있는 패키지

4. 향상된 서버 만들기

❑ AdvancedChatServer 구조



[그림 16-2] AdvancedChatServer 구조

4. 향상된 서버 만들기

❑ AdvancedChatServer.java

```
package net.daum.javacafe.server;

import java.io.File;
import java.io.IOException;

import net.daum.javacafe.pool.PoolManager;
import net.daum.javacafe.pool.buffer.ByteBufferPool;
import net.daum.javacafe.pool.buffer.ByteBufferPoolIF;
import net.daum.javacafe.pool.selector.AcceptSelectorPool;
import net.daum.javacafe.pool.selector.RequestSelectorPool;
import net.daum.javacafe.pool.selector.SelectorPoolIF;
import net.daum.javacafe.pool.thread.ThreadPool;
import net.daum.javacafe.pool.thread.ThreadPoolIF;
import net.daum.javacafe.queue.BlockingEventQueue;
import net.daum.javacafe.queue.Queue;


public class AdvancedChatServer {

    private Queue queue = null;
    private SelectorPoolIF acceptSelectorPool = null;
    private SelectorPoolIF requestSelectorPool = null;

    private ByteBufferPoolIF byteBufferPool = null;

    ThreadPoolIF acceptThreadPool = null;
    ThreadPoolIF readWriteThreadPool = null;

    public AdvancedChatServer() {
        try {
            initResource();
            startServer();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

4. 향상된 서버 만들기

❑ AdvancedChatServer.java

```
    private void initResource() throws IOException {
        // ByteBufferPool 생성..
        File bufferFile = new File("C:/Buffer.tmp");
        if (!bufferFile.exists()) {
            bufferFile.createNewFile();
        }
        bufferFile.deleteOnExit();
        byteBufferPool = new ByteBufferPool(20*1024, 40*2048, bufferFile);

        // Queue 생성..
        queue = BlockingEventQueue.getInstance();

        // PoolManager 에 ByteBufferPool 등록..
        PoolManager.registByteBufferPool(byteBufferPool);

        // ThreadPool 생성..
        acceptThreadPool = new ThreadPool(
                queue, "net.daum.javacafe.pool.thread.processor.AcceptProcessor");
        readWriteThreadPool = new ThreadPool(
                queue, "net.daum.javacafe.pool.thread.processor.ReadWriteProcessor");

        // SelectorPool 생성..
        acceptSelectorPool = new AcceptSelectorPool(queue);
        requestSelectorPool = new RequestSelectorPool(queue);

        // PoolManager 에 SelectorPool 등록..
        PoolManager.registAcceptSelectorPool(acceptSelectorPool);
        PoolManager.registRequestSelectorPool(requestSelectorPool);
    }

    private void startServer() {
        acceptThreadPool.startAll();
        readWriteThreadPool.startAll();

        acceptSelectorPool.startAll();
        requestSelectorPool.startAll();
    }

    public static void main(String[] args) {
        AdvancedChatServer server = new AdvancedChatServer();
    }

}
```

4. 향상된 서버 만들기

❑ BlockingEventQueue 클래스

```
package net.daum.javacafe.queue;

import java.util.ArrayList;
import java.util.List;

import net.daum.javacafe.event.Job;
import net.daum.javacafe.event.NIOEvent;

public class BlockingEventQueue implements Queue {

    private final Object acceptMonitor = new Object();
    private final Object readMonitor = new Object();

    private final List acceptQueue = new ArrayList();
    private final List readQueue = new ArrayList();

    private static BlockingEventQueue instance = new BlockingEventQueue();

    public static BlockingEventQueue getInstance() {
        if (instance == null) {
            synchronized (BlockingEventQueue.class) {
                instance = new BlockingEventQueue();
            }
        }
        return instance;
    }

    private BlockingEventQueue() {}


    /* (non-Javadoc)
     * @see net.daum.javacafe.queue.Queue#pop(int)
     */
    public Job pop(int eventType) {
        switch (eventType) {
            case NIOEvent.ACCEPT_EVENT : return getAcceptJob();
            case NIOEvent.READ_EVENT   : return getReadJob();
            default : throw new IllegalArgumentException("Illegal EventType..");
        }
    }

    /**
     * @return
     */
```

4. 향상된 서버 만들기

❑ BlockingEventQueue 클래스

```
public Job pop(int eventType) {
    switch (eventType) {
        case NIOEvent.ACCEPT_EVENT : return getAcceptJob();
        case NIOEvent.READ_EVENT   : return getReadJob();
        default : throw new IllegalArgumentException("Illegal EventType..");
    }
}

/**
 * @return
 */
private Job getAcceptJob() {
    synchronized (acceptMonitor) {
        if (acceptQueue.isEmpty()) {
            try {
                acceptMonitor.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return (Job) acceptQueue.remove(0);
    }
}

/**
 * @return
 */
private Job getReadJob() {
    synchronized (readMonitor) {
        if (readQueue.isEmpty()) {
            try {
                readMonitor.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return (Job) readQueue.remove(0);
    }
}


/* (non-Javadoc)
 * @see net.daum.javacafe.queue.Queue#push(net.daum.javacafe.event.Job)
 */
```

4. 향상된 서버 만들기

❑ BlockingEventQueue 클래스

```
    public void push(Job job) {
        if (job != null) {
            int eventType = job.getEventType();
            switch (eventType) {
                case NIOEvent.ACCEPT_EVENT : putAcceptJob(job); break;
                case NIOEvent.READ_EVENT   : putReadJob(job); break;
                default : throw new IllegalArgumentException("Illegal EventType..");
            }
        }
    }

    /**
     * @param job
     */
    private void putAcceptJob(Job job) {
        synchronized (acceptMonitor) {
            acceptQueue.add(job);
            acceptMonitor.notify();
        }
    }

    /**
     * @param job
     */
    private void putReadJob(Job job) {
        synchronized (readMonitor) {
            readQueue.add(job);
            readMonitor.notify();
        }
    }

}
```

4. 향상된 서버 만들기

❑ Job 클래스

```
package net.daum.javacafe.event;

import java.util.Map;

/**
 * @author SongJiHoon
 */
public class Job {

    private int eventType;
    private Map session = null;

    public Job() {}

    public Job(int eventType, Map session) {
        this.eventType = eventType;
        this.session = session;
    }

    /**
     * @return Returns the session.
     */
    public Map getSession() {
        return session;
    }
    /**
     * @param session The session to set.
     */
    public void setSession(Map session) {
        this.session = session;
    }

    /**
     * @return Returns the eventType.
     */
    public int getEventType() {
        return eventType;
    }
    /**
     * @param eventType The eventType to set.
     */
    public void setEventType(int eventType) {
        this.eventType = eventType;
    }
```

4. 향상된 서버 만들기

❑ PoolManager

```
package net.daum.javacafe.pool;

import java.util.HashMap;
import java.util.Map;

import net.daum.javacafe.pool.buffer.ByteBufferPoolIF;
import net.daum.javacafe.pool.selector.SelectorPoolIF;

public class PoolManager {

    private static Map map = new HashMap();

    private static PoolManager instance = new PoolManager();

    private PoolManager() {}



    public static void registAcceptSelectorPool(SelectorPoolIF selectorPool) {
        map.put("AcceptSelectorPool", selectorPool);
    }

    public static void registRequestSelectorPool(SelectorPoolIF selectorPool) {
        map.put("RequestSelectorPool", selectorPool);
    }
```

4. 향상된 서버 만들기

❑ PoolManager

```
    public static SelectorPoolIF getAcceptSelectorPool() {
        return (SelectorPoolIF) map.get("AcceptSelectorPool");
    }

    public static SelectorPoolIF getRequestSelectorPool() {
        return (SelectorPoolIF) map.get("RequestSelectorPool");
    }



    public static void registByteBufferPool(ByteBufferPoolIF byteBufferPool) {
        map.put("ByteBufferPool", byteBufferPool);
    }

    public static ByteBufferPoolIF getByteBufferPool() {
        return (ByteBufferPoolIF) map.get("ByteBufferPool");
    }

}
```

4. 향상된 서버 만들기

❑ SelectorPoolIF

```
package net.daum.javacafe.pool.selector;

public interface SelectorPoolIF {

	public Thread get();
	public void put(Thread handler);
	public int size();
	public boolean isEmpty();
	public void startAll();
	public void stopAll();

}
```

4. 향상된 서버 만들기

❑ SelectorPoolAdaptor

```
package net.daum.javacafe.pool.selector;

import java.util.ArrayList;
import java.util.List;

public abstract class SelectorPoolAdaptor implements SelectorPoolIF {

    protected int size = 2;

    private int roundRobinIndex = 0;

    private final Object monitor = new Object();
    protected final List pool = new ArrayList();

    protected abstract Thread createHandler(int index);
    public abstract void startAll();
    public abstract void stopAll();

    /* (non-Javadoc)
     * @see net.daum.javacafe.pool.SelectorPoolIF#pop()
     */
    public Thread get() {
        synchronized (monitor) {
            return (Thread) pool.get( roundRobinIndex++ % size );
        }
    }

    /* (non-Javadoc)
     * @see net.daum.javacafe.pool.SelectorPoolIF#push(net.daum.javacafe.pool.handler.SelectorHandlerIF)
     */
```

4. 향상된 서버 만들기

❑ SelectorPoolAdaptor

```
public void put(Thread handler) {
    synchronized (monitor) {
        if (handler != null) {
            pool.add(handler);
        }
        monitor.notify();
    }
}

/* (non-Javadoc)
 * @see net.daum.javacafe.pool.SelectorPoolIF#size()
 */
public int size() {
    synchronized (monitor) {
        return pool.size();
    }
}

/* (non-Javadoc)
 * @see net.daum.javacafe.pool.SelectorPoolIF#isEmpty()
 */
public boolean isEmpty() {
    synchronized (monitor) {
        return pool.isEmpty();
    }
}
```

4. 향상된 서버 만들기

❑ AcceptSelectorPool

```
package net.daum.javacafe.pool.selector;

import java.io.IOException;
import java.nio.channels.Selector;
import java.util.Iterator;

import net.daum.javacafe.pool.selector.handler.AcceptHandler;
import net.daum.javacafe.queue.Queue;

public class AcceptSelectorPool extends SelectorPoolAdaptor {

    private int port = 9090;
    private Queue queue = null;

    public AcceptSelectorPool(Queue queue) {
        this(queue, 1, 9090);
    }

    public AcceptSelectorPool(Queue queue, int size, int port) {
        super.size = size;
        this.queue = queue;
        this.port = port;
        init();
    }

    private void init() {
        for (int i = 0; i < size; i++) {
            pool.add(createHandler(i));
        }
    }

    /* (non-Javadoc)
     * @see net.daum.javacafe.pool.SelectorPoolAdaptor#createHandler(int)
     */
```

4. 향상된 서버 만들기

❑ AcceptSelectorPool

```
protected Thread createHandler(int index) {
    Selector selector = null;
    try {
        selector = Selector.open();
    } catch (IOException e) {
        e.printStackTrace();
    }
    Thread handler = new AcceptHandler(queue, selector, port, index);

    return handler;
}

/* (non-Javadoc)
 * @see net.daum.javacafe.pool.selector.SelectorPoolIF#startAll()
 */
public void startAll() {
    Iterator iter = pool.iterator();
    while (iter.hasNext()) {
        Thread handler = (Thread) iter.next();
        handler.start();
    }
}

/* (non-Javadoc)
 * @see net.daum.javacafe.pool.selector.SelectorPoolIF#stopAll()
 */
public void stopAll() {
    Iterator iter = pool.iterator();
    while (iter.hasNext()) {
        Thread handler = (Thread) iter.next();
        handler.interrupt();
        handler = null;
    }
}
```

4. 향상된 서버 만들기

❑ AcceptHandler

```
package net.daum.javacafe.pool.selector.handler;

import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import net.daum.javacafe.event.Job;
import net.daum.javacafe.event.NIOEvent;
import net.daum.javacafe.queue.Queue;

public class AcceptHandler extends Thread {

    private Queue queue = null;
    private Selector selector = null;
    private int port = 9090;
    private String name = "AcceptHandler-";

    public AcceptHandler(Queue queue, Selector selector, int port, int index) {
        this.queue = queue;
        this.selector = selector;
        this.port = port;
        setName(name + index);
        init();
    }

    private void init() {
        try {
            ServerSocketChannel ssc = ServerSocketChannel.open();
            ssc.configureBlocking(false);

            InetSocketAddress address = new InetSocketAddress("localhost", port);
            ssc.socket().bind(address);

            System.out.println("@AcceptHandler(" + getName() + ") Bound to " + address);

            ssc.register(this.selector, SelectionKey.OP_ACCEPT);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

4. 향상된 서버 만들기

❑ AcceptHandler

```
public void run() {
    try {
        while (!Thread.currentThread().isInterrupted()) {
            int keysReady = selector.select();
            acceptPendingConnections();
        }
    } catch (Exception e) {
        //e.printStackTrace();
    }
}

private void acceptPendingConnections() throws Exception {
    Iterator iter = selector.selectedKeys().iterator();
    while (iter.hasNext()) {
        SelectionKey key = (SelectionKey) iter.next();
        iter.remove();

        ServerSocketChannel readyChannel = (ServerSocketChannel) key.channel();
        SocketChannel sc = readyChannel.accept();

        System.out.println("@AcceptHandler(" + getName() + ") connection accepted from " + sc.socket().getInetAddress());

        pushMyJob(sc);
    }
}

private void pushMyJob(SocketChannel sc) {
    Map session = new HashMap();
    session.put("SocketChannel", sc);
    Job job = new Job(NIOEvent.ACCEPT_EVENT, session);
    queue.push(job);
}
```

4. 향상된 서버 만들기

❑ RequestHandler

```
package net.daum.javacafe.pool.selector.handler;

import java.nio.channels.ClosedChannelException;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Vector;

import net.daum.javacafe.event.Job;
import net.daum.javacafe.event.NIOEvent;
import net.daum.javacafe.queue.ChattingRoom;
import net.daum.javacafe.queue.Queue;

public class RequestHandler extends HandlerAdaptor {

    private Queue queue = null;
    private Selector selector = null;
    private String name = "RequestHandler-";

    private Vector newClients = new Vector();

    public RequestHandler(Queue queue, Selector selector, int index) {
        this.queue = queue;
        this.selector = selector;
        setName(name + index);
    }
```

4. 향상된 서버 만들기

❑ RequestHandler

```
public void run() {
    try {
        while (!Thread.currentThread().isInterrupted()) {
            processNewConnection();
            int keysReady = selector.select(1000);
            System.out.println("@RequestHandler(" + getName() + ") selected : " + keysReady );
            if (keysReady > 0) {
                processRequest();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private synchronized void processNewConnection() throws ClosedChannelException {
    Iterator iter = newClients.iterator();
    while (iter.hasNext()) {
        SocketChannel sc = (SocketChannel) iter.next();
        sc.register(selector, SelectionKey.OP_READ);
        ChattingRoom.getInstance().add(sc);
        System.out.println("@RequestHandler(" + getName() + ") success regist : " + sc.toString() );
    }
    newClients.clear();
}

private void processRequest() {
    Iterator iter = selector.selectedKeys().iterator();
    while (iter.hasNext()) {
        SelectionKey key = (SelectionKey) iter.next();
        iter.remove();
        pushMyJob(key);
    }
}
```

4. 향상된 서버 만들기

❑ RequestHandler

```
    private void pushMyJob(SelectionKey key) {
        Map session = new HashMap();
        session.put("SelectionKey", key);
        Job job = new Job(NIOEvent.READ_EVENT, session);
        queue.push(job);
    }

    public void addClient(SocketChannel sc) {
        newClients.add(sc);
    }

}
```

4. 향상된 서버 만들기

❑ ThreadPoolIF

```
package net.daum.javacafe.pool.thread;

public interface ThreadPoolIF {

    public void addThread();
    public void removeThread();
    public void startAll();
    public void stopAll();

}
```

4. 향상된 서버 만들기

❑ ThreadPool

```
package net.daum.javacafe.pool.thread;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import net.daum.javacafe.queue.Queue;
import net.daum.javacafe.util.DynamicClassLoader;

public class ThreadPool implements ThreadPoolIF {

    private int max = 10;
    private int min = 2;
    private int current = 0;

    private final Object monitor = new Object();
    private final List pool = new ArrayList();

    private Queue queue = null;
    private String type = null;

    public ThreadPool(Queue queue, String type) {
        this(queue, type, 2, 10);
    }

    public ThreadPool(Queue queue, String type, int min, int max) {
        this.queue = queue;
        this.type = type;
        this.min = min;
        this.max = max;
        init();
    }
```

4. 향상된 서버 만들기

❑ ThreadPool

```
private void init() {
    for (int i = 0; i < min; i++) {
        pool.add(createThread());
    }
}

private synchronized Thread createThread() {
    Thread thread = null;
    try {
        thread = (Thread) DynamicClassLoader.createInstance(type, queue);
    } catch (Exception e) {
        e.printStackTrace();
    }
    current++;
    return thread;
}

/* (non-Javadoc)
 * @see net.daum.javacafe.pool.thread.ThreadPoolIF#startAll()
 */
public void startAll() {
    synchronized (monitor) {
        Iterator iter = pool.iterator();
        while (iter.hasNext()) {
            Thread thread = (Thread) iter.next();
            thread.start();
        }
    }
}

/* (non-Javadoc)
 * @see net.daum.javacafe.pool.thread.ThreadPoolIF#stopAll()
 */
```

4. 향상된 서버 만들기

❑ ThreadPool

```
public void stopAll() {
    synchronized (monitor) {
        Iterator iter = pool.iterator();
        while (iter.hasNext()) {
            Thread thread = (Thread) iter.next();
            thread.interrupt();
            thread = null;
        }
        pool.clear();
    }
}

/* (non-Javadoc)
 * @see net.daum.javacafe.pool.thread.ThreadPoolIF#addThread()
 */
public void addThread() {
    synchronized (monitor) {
        if (current < max) {
            Thread t = createThread();
            t.start();
            pool.add(t);
        }
    }
}

/* (non-Javadoc)
 * @see net.daum.javacafe.pool.thread.ThreadPoolIF#removeThread()
 */
public void removeThread() {
    synchronized (monitor) {
        if (current > min) {
            Thread t = (Thread) pool.remove(0);
            t.interrupt();
            t = null;
        }
    }
}
```

4. 향상된 서버 만들기

❑ AcceptProcessor

```
package net.daum.javacafe.pool.thread.processor;

import java.nio.channels.SocketChannel;

import net.daum.javacafe.event.Job;
import net.daum.javacafe.event.NIOEvent;
import net.daum.javacafe.pool.PoolManager;
import net.daum.javacafe.pool.selector.handler.HandlerAdaptor;
import net.daum.javacafe.queue.Queue;

public class AcceptProcessor extends Thread {

    private Queue queue = null;

    public AcceptProcessor(Queue queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            while (!Thread.currentThread().isInterrupted()) {
                Job job = queue.pop(NIOEvent.ACCEPT_EVENT);
                SocketChannel sc = (SocketChannel) job.getSession().get("SocketChannel");
                sc.configureBlocking(false);
                HandlerAdaptor handler = (HandlerAdaptor) PoolManager.getRequestSelectorPool().get();
                handler.addClient(sc);
            }
        } catch (Exception e) {
            //e.printStackTrace();
        }
    }

}
```

4. 향상된 서버 만들기

❑ ReadWriteProcessor

```
package net.daum.javacafe.pool.thread.processor;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.SocketChannel;
import java.util.Iterator;

import net.daum.javacafe.event.Job;
import net.daum.javacafe.event.NIOEvent;
import net.daum.javacafe.pool.PoolManager;
import net.daum.javacafe.pool.buffer.ByteBufferPoolIF;
import net.daum.javacafe.queue.ChattingRoom;
import net.daum.javacafe.queue.Queue;

public class ReadWriteProcessor extends Thread {

    private Queue queue = null;

    public ReadWriteProcessor(Queue queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            while (!Thread.currentThread().isInterrupted()) {
                Job job = queue.pop(NIOEvent.READ_EVENT);
                SelectionKey key = (SelectionKey) job.getSession().get("SelectionKey");
                SocketChannel sc = (SocketChannel) key.channel();

                try {
                    broadcast(sc);
                } catch (IOException e) {
                    closeChannel(sc);
                }
            }
        } catch (Exception e) {
            //e.printStackTrace();
        }
    }
```

4. 향상된 서버 만들기

❑ ReadWriteProcessor

```
private void broadcast(SocketChannel sc) throws IOException {
    ByteBufferPoolIF bufferPool = PoolManager.getByteBufferPool();
    ByteBuffer buffer = null;
    try {
        buffer = bufferPool.getMemoryBuffer();

        for (int i = 0; i < 2; i++) {
            sc.read(buffer);
        }

        buffer.flip();

        Iterator iter = ChattingRoom.getInstance().iterator();
        while (iter.hasNext()) {
            SocketChannel member = (SocketChannel) iter.next();
            if (member != null && member.isConnected()) {
                while (buffer.hasRemaining()) {
                    member.write(buffer);
                }
                buffer.rewind();
            }
        }
    } finally {
        bufferPool.putBuffer(buffer);
    }
}

private void closeChannel(SocketChannel sc) {
    try {
        sc.close();
        ChattingRoom.getInstance().remove(sc);
    } catch (IOException e) {
    }
}
```