

오픈소스 네트워크 엔진 SuperSocket 사용하기

NHN Next 겸임 교수(게임)
최흥배

<https://github.com/jacking75/choiHeungbae>

**Windows 플랫폼에서
고성능 네트워크
프로그램을 만들 때 가장
자주 사용하는 기술은
C++ & IOCP**

... PC 온라인 게임 시대 때...

Google Play

검색







앱

카테고리 | 홈 | 인기 차트 | 새 출시작

내 앱
쇼핑하기

게임
키즈
에디터 추천

최고 매출 - 앱

 <p>1. 모두의마블 for K Netmarble Games</p> <p>★★★★★</p>	 <p>2. 데스티니 차일드 Next Floor Corp.</p> <p>★★★★★</p>	 <p>3. 아덴 ITSGAMES</p> <p>★★★★★</p>	 <p>4. 뮤오리진 Webzen Inc.</p> <p>★★★★★</p>	 <p>5. 카카오톡 KakaoT Kakao Corporation</p> <p>★★★★★</p>	 <p>6. 세븐나이츠 for K Netmarble Games</p> <p>★★★★★</p>
--	--	---	---	--	--

설정
|프트 카드 사용
|프트 카드 구매



C#의 비동기 Socket은
Windows에서는 내부적으로
IOCP로 구현되어 있음.
즉 비동기 네트워크 측면만
보았을 때는 C++로 IOCP를
사용하는 것과 비슷



SuperSocket, an extensible socket server framework

SuperSocket is a light weight, cross platform and extensible .Net/Mono socket server application framework. You can use it to build application (like game server, GPS server, industrial control system, data acquisition server etc) easily without thinking about how to the socket connections and how socket works.

[Read Documentation »](#)[Download »](#)

Features

 Windows Server Windows Azure<http://www.supersocket.net/> mono

사용하기 쉽고,
고 성능이고,
안전하다

- .NET 플랫폼용 오픈 소스 네트워크 라이브러리 3.5 ~ 4.5까지 지원.
- Windows, Linux, Mono, Azure를 지원한다.
- 비동기 I/O를 지원. TCP, UDP
- SSL/TLS 지원, 확장 기능 등 다양한 기능 지원
- 공식 사이트 <http://www.supersocket.net>
문서는 <http://docs.supersocket.net/>

- 현재(2016.09.02) 기준
- nuget 최신 버전은 1.6.6.1
<https://www.nuget.org/packages/SuperSocket/>
- GitHub 버전은 1.6.7
<https://github.com/kerryjiang/SuperSocket>
- 2.0 버전을 준비 중
Visual Studio Code 지원.
어쩌면 .NET Core도 지원?

오픈 소스 답지 않게(?) 문서화와 예제 코드가 잘 만들어져 있어서 분석이 쉽다

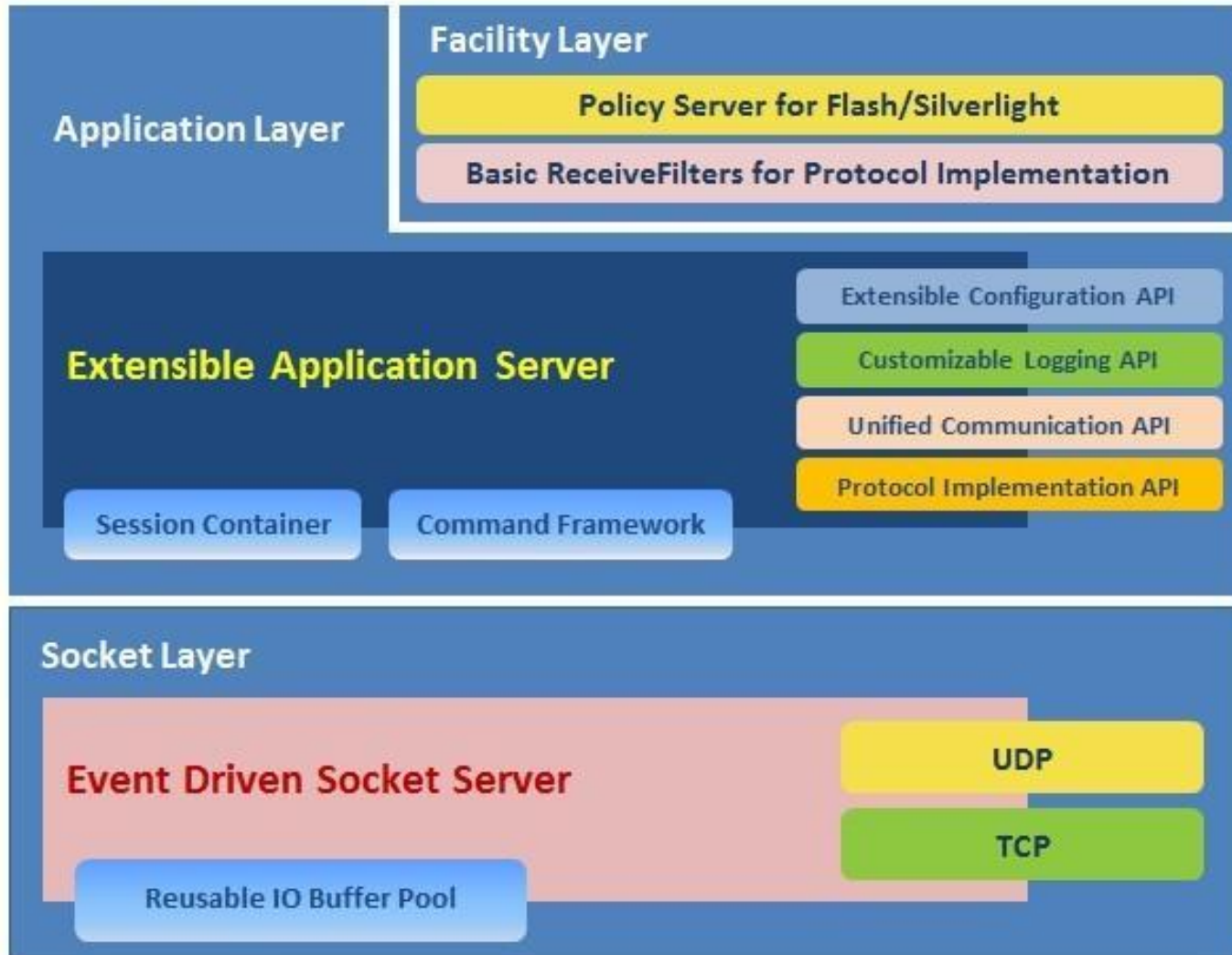
SuperSocket 1.6 Documentation

English (United States) v1.6

- [Architecture Diagrams](#)
- [A Telnet Example](#)
- [Implement Your AppServer and AppSession](#)
- [Start SuperSocket by Configuration](#)
- [SuperSocket Basic Configuration](#)
- [The Built-in Command Line Protocol](#)
- [The Built-in Common Format Protocol Implementation Templates](#)
- [Implement Your Own Communication Protocol with IRequestInfo, IReceiveFilter and etc](#)
- [Command and Command Loader](#)
- [Get the Connected Event and Closed Event of a Connection](#)
- [Push Data to Clients from Server Initiative](#)
- [Extend Server Configuration](#)
- [Server Configuration Hot Update](#)
- [Command Filter](#)
- [Connection Filter](#)
- [Multiple Listeners](#)
- [Multiple Server Instances](#)
- [Implement Your Commands by Dynamic Language](#)
- [Logging in SuperSocket](#)
- [The Built in Flash Silverlight Policy Server in SuperSocket](#)
- [Enable TLS/SSL transferring layer encryption in SuperSocket](#)
- [Run SuperSocket in Windows Azure](#)
- [Run SuperSocket in Linux/Unix](#)
- [SuperSocket ServerManager](#)
- [New Features and Breaking Changes](#)

Branch: v1.6 SuperSocket / QuickStart /	
skynetr add missing reference of System.configuration	
..	
AppDomainIsolation	add missing reference of System.configuration
Basic	fixed the building errors and warnings in QuickStart samp
BroadcastService	enabled GC server mode in the configuration files
CommandFilter	fixed the building errors and warnings in QuickStart samp
ConfigSample	enabled GC server mode in the configuration files
ConnectionFilter	enabled GC server mode in the configuration files
CountSpliterProtocol	enabled GC server mode in the configuration files
CustomProtocol	enabled GC server mode in the configuration files
CustomRequestInfoParser	enabled GC server mode in the configuration files
GPSSocketServer	enabled GC server mode in the configuration files
IronSocketServer	add missing reference of System.configuration
MultipleAppServer	added missing System.Configuration reference
MultipleCommandAssembly	add missing reference of System.configuration
RemoteProcessService	enabled GC server mode in the configuration files
ServerManagerSample	improved servermanager's sample
ServerPush	improved the PushServer sample to demonstrate config f
SwitchReceiveFilter	updated SwitchReceiveFilter project's namespace
TerminatorProtocol	enabled GC server mode in the configuration files
WindowsAzure	added log4net config file for Azure sample
QuickStart.2013.sln	upgraded QuickStart solution file
QuickStart.sln	removed ServerManagerSample from the QuickStart solu

SuperSocket Layers



SuperSocket Objects Model

Application Server (AppServer)

Commands

Command A

Command B

Command C

Command D

Session Container

Session A

Session B

Session C

Session D

Session E

Session F

Session G

Session H

Session I

Session J

Session K

Session L

Config

Command Filters

Log/LogFactory

Command Loaders

ReceiveFilterFactory

Connection Filters

Socket Server

Listeners

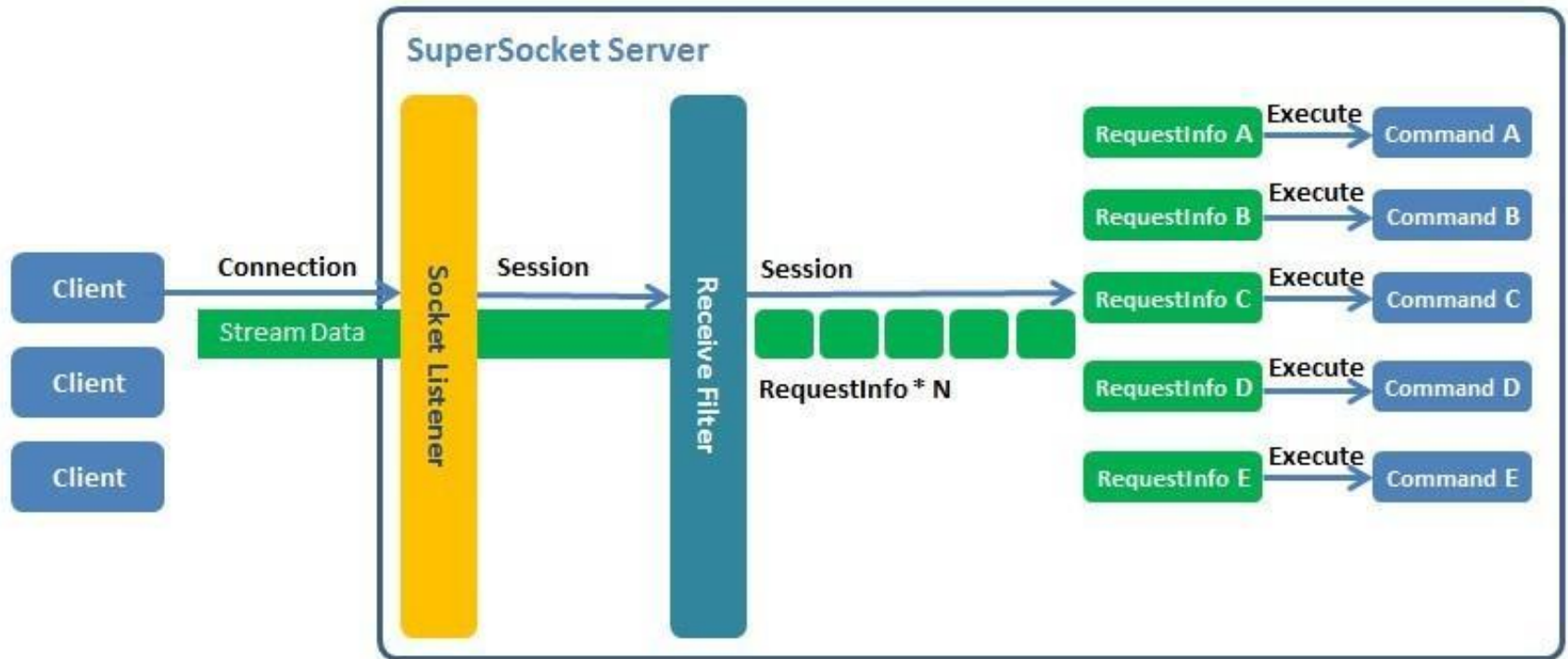
TCP Listener on Port A

TCP Listener on Port B

TCP Listener on Port C

UDP Listener on Port D

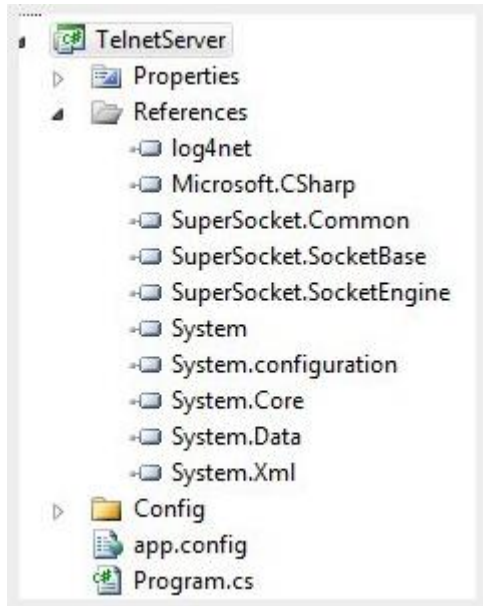
SuperSocket Request Handling Model



SuperSocket Isolation Model

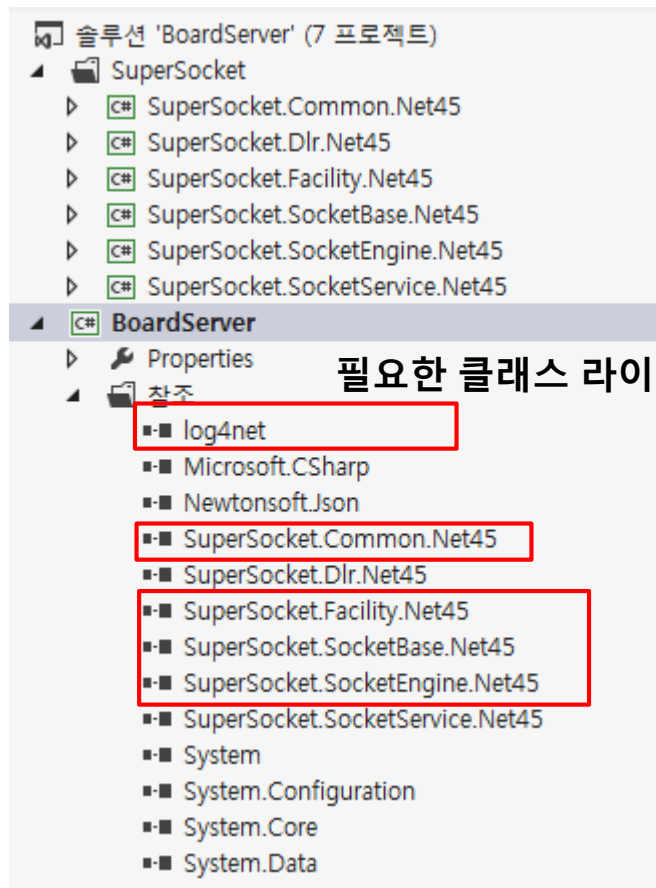


설치하기 - 소스 코드에서




SuperSocket 소스에 있는 log4net.dll을 포함한다.


SuperSocket.Common, SuperSocket.SocketBase, SuperSocket.SocketEngine는 왼쪽 그림처럼 dll을 참조에 포함해도 되고 아니면 프로젝트를 바로 포함해도 된다(아래).



필요한 클래스 라이브러리

설치하기 – NuGet에서

 **nuget**

Search Packages 

Register / Sign in

Home Packages Upload Package Statistics Documentation Downloads Blog



SuperSocket 1.6.6.1

SuperSocket is a light weight, cross platform and extensible socket server application framework. You can use it to build a server side socket application (like game server, GPS server, industrial control system, data acquisition server etc) easily without thinking about how to use socket, how to maintain the socket connections and how socket works.

To install SuperSocket, run the following command in the [Package Manager Console](#)


```
PM> Install-Package SuperSocket
```


52,871

Downloads

42,305

Downloads of v 1.6.6.1

 **nuget**

Search Packages 

Register / Sign in

Home Packages Upload Package Statistics Documentation Downloads Blog



SuperSocket.Engine 1.6.6.1

SuperSocket is a light weight, cross platform and extensible socket server application framework. You can use it to build a server side socket application (like game server, GPS server, industrial control system, data acquisition server etc) easily without thinking about how to use socket, how to maintain the socket connections and how socket works.

To install SuperSocket.Engine, run the following command in the [Package Manager Console](#)

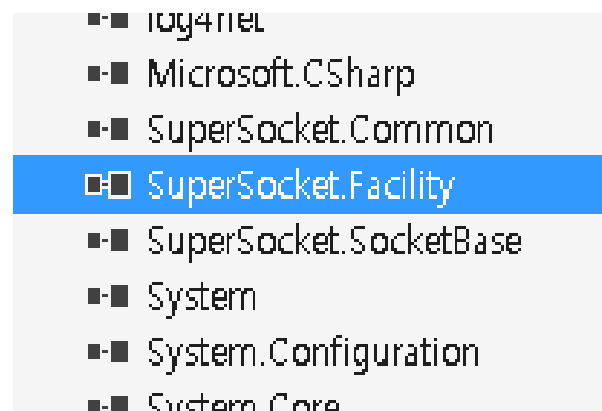
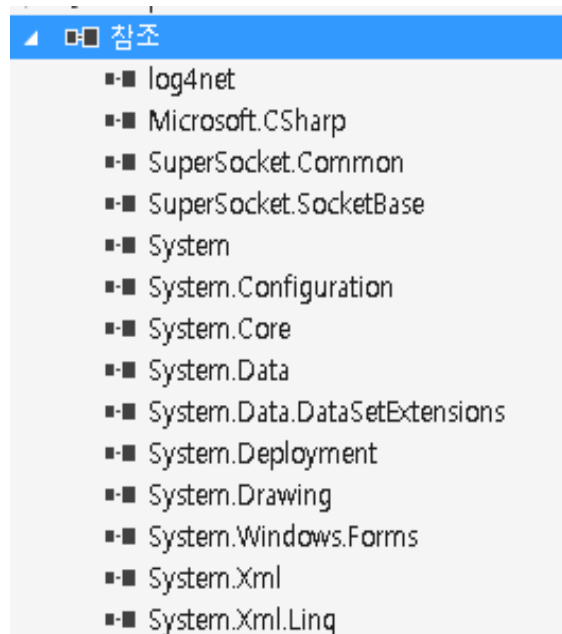
```
PM> Install-Package SuperSocket.Engine
```

42,533

Downloads

40,349

Downloads of v 1.6.6.1



CountSpliterReceiveFilter, FixedSizeReceiveFilter, BeginEndMarkReceiveFilter, FixedHeaderReceiveFilter 를 사용하기 위해서는 'SuperSocket.Facility'가 필요한데 기본으로 등록되지 않으므로 NuGet으로 받은 package 디렉토리에서 선택해서 추가한다.

서버 실행 - 설정

```
void InitConfig()
{
    m_Config = new ServerConfig
    {
        Port = 23478,
        Ip = "Any",
        MaxConnectionNumber = 100,
        Mode = SocketMode.Tcp,
        Name = "BoardServerNet"
    };
}
```



```
void CreateServer()
{
    m_Server = new BoardServerNet();
    bool bResult = m_Server.Setup(new RootConfig(),
                                   m_Config,
                                   logFactory: new Log4NetLogFactory()
                                   );

    if (bResult == false)
    {
    }

    .....
}
```

```
class BoardServerNet : AppServer<NetworkSession,  
                       EFBinaryRequestInfo>  
{  
}
```

서버 실행 - 네트워크 설정 및 시작/중단

```
// 포트 번호 2012로 설정
if (! m_Server.Setup(2012))
{
    return;
}

....

// 네트워크 시작
if (! m_Server.Start())
{
    return;
}

....

// 네트워크 중지
m_Server.Stop();
```

서버 실행 - 핸들러 등록

새로운 클라이언트가 연결되면 호출될 핸들러 등록

```
appServer.NewSessionConnected += new  
SessionHandler<AppSession>(appServer_NewSessionConnected);
```

....

```
static void appServer_NewSessionConnected(AppSession session)  
{  
    session.Send("Welcome to SuperSocket Telnet Server");  
}
```

클라이언트가 보낸 데이터를 받으면 호출될 핸들러 등록

```
appServer.NewRequestReceived += new RequestHandler<AppSession,  
StringRequestInfo>(appServer_NewRequestReceived);
```

```
....
```

```
static void appServer_NewRequestReceived(AppSession session,  
StringRequestInfo requestInfo)
```

```
{  
    switch (requestInfo.Key.ToUpper())  
    {  
        case("ECHO"):  
            session.Send(requestInfo.Body);  
            break;  
  
            .....  
  
    }
```

AppServer와 AppSession 구현

- AppSession
서버에 연결된 **Socket**의 로직 클래스.
이 클래스를 통해 클라이언트의 연결, 끊어짐, 데이터 주고 받기를 한다.
- AppServer
네트워크 서버 클래스. 모든 AppSession 객체를 관리한다.
SuperSocket의 몸통이다.

```
public class TelnetSession : AppSession<TelnetSession>
{
    protected override void OnSessionStarted()
    {
        this.Send("Welcome to SuperSocket Telnet Server");
    }

    protected override void HandleUnknownRequest(StringRequestInfo requestInfo)
    {
        this.Send("Unknow request");
    }

    protected override void HandleException(Exception e)
    {
        this.Send("Application error: {0}", e.Message);
    }

    protected override void OnSessionClosed(CloseReason reason)
    {
        //add you logics which will be executed after the session is closed
        base.OnSessionClosed(reason);
    }
}
```

```
public class TelnetServer : AppServer<TelnetSession>
{
    protected override bool Setup(IRootConfig rootConfig, IServerConfig config)
    {
        return base.Setup(rootConfig, config);
    }

    protected override void OnStartup()
    {
        base.OnStartup();
    }

    protected override void OnStopped()
    {
        base.OnStopped();
    }
}
```

서버 네트워크 옵션 설정하기

name: the name of appServer instance

serverType: the full name of the AppServer your want to run

ip: listen ip

port: listen port

위 설정을 아래와 같이 config 파일에 정의할 수 있다.

```
<superSocket>
  <servers>
    <server name="TelnetServer"
      serverType="SuperSocket.QuickStart.TelnetServer_StartByConfig.TelnetServer,
        SuperSocket.QuickStart.TelnetServer_StartByConfig"
      ip="Any" port="2020">
    </server>
  </servers>
</superSocket>
```


위의 Config 파일 사용 예

```
static void Main(string[] args)
{
    var bootstrap =
    BootstrapFactory.CreateBootstrap(
    );

    if (!bootstrap.Initialize())
    {
        return;
    }

    var result = bootstrap.Start();
```

```
if (result == StartResult.Failed)
{
    return;
}

//Stop the appServer
bootstrap.Stop();
}
```

서버 네트워크 옵션 설정하기

Config 파일 - 멀티 인스턴스 사용 예

```
<superSocket>
  <servers>
    <server name="TelnetServerA"
      serverTypeName="TelnetServer"
      ip="Any" port="2020">
    </server>
    <server name="TelnetServerB"
      serverTypeName="TelnetServer"
      ip="Any" port="2021">
    </server>
  </servers>
  <serverTypes>
    <add name="TelnetServer"
type="SuperSocket.QuickStart.TelnetServer_StartByConfig.TelnetServer,
SuperSocket.QuickStart.TelnetServer_StartByConfig"/>
  </serverTypes>
</superSocket>
```

App.Config 파일 이외의 설정 파일을 사용하기 위해서는 아래처럼 파일 이름을 지정한다.

```
m_Bootstrap = BootstrapFactory.CreateBootstrapFromConfigFile("SuperSocket.config");
```

SuperSocket 라이브러리가 호스트 프로그램이 아닌 다른 프로젝트에서 사용하는 경우의 설정

솔루션 'ChatServerForm' (4 프로젝트)

ChatServerForm

Properties

참조

Config

App.config

MainForm.cs

packages.config

Program.cs

ChatServerLib

Properties

참조

CommonLibrary

CommonServerLib

CSBaseLib

log4net

Microsoft.CSharp

Newtonsoft.Json

SuperSocket.Common

SuperSocket.Facility

SuperSocket.SocketBase

SuperSocket.SocketEngine

System

System.Configuration

System.Core

System.Data

System.Data.DataSetExtensions

System.Threading.Tasks.Dataflow

System.Xml

System.Xml.Linq

Z.ExtensionMethods

```
App.config App.config
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="superSocket" type="SuperSocket.SocketEngine.Configuration.SocketServiceConfig, SuperS
    <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System, Version=4.0.
    <section name="ChatServerForm.Properties.Settings" type="System.Configuration.ClientSettingsSectio
  </sectionGroup>
</configSections>

  <superSocket disablePerformanceDataCollector="true">
    <servers>
      <server name="ChatServer" serverType="ChatServerLib.ServerNetwork, ChatServerLib" ip="Any" port="1
    </server>
    </servers>
  </superSocket>
```

SuperSocket의 'AppServer'
클래스를 상속한 클래스

Config 파일의 설정 값 확인

The image shows a C# code editor with a method `CreateServer()` and its configuration object `SocketServiceConfig` in the Visual Studio debugger.

Code Snippet:

```
42 public void CreateServer()  
43 {  
44     var bootstrap = BootstrapFactory.CreateBootstrap();  
45  
46     if (!bootstrap.Initialize())  
47     {  
48         DevLog.WriteLine("Bootstrap failed to initialize.");  
49         return;  
50     }  
51  
52     var result = bootstrap.Configure();  
53  
54     if (result != Success)  
55     {  
56         DevLog.WriteLine("Configuration failed: {0}", result);  
57         return;  
58     }  
59  
60     //bool bResult = true;  
61  
62     //if (bResult == true)  
63     //{  
64     //    DevLog.WriteLine("Server is running.");  
65     //    return;  
66     //}  
67  
68     DevLog.WriteLine("Server is running.");  
69 }  
70  
71  
72 public bool IsRunning(ServerState state)  
73 {  
74     if (state == ServerState.Running)  
75     {  
76         return true;  
77     }  
78  
79     return false;  
80 }  
81
```

Debugger Configuration:

The debugger shows the `SocketServiceConfig` object with the following properties:

- `MaxCompletionPortThreads`: 1
- `MaxWorkingThreads`: 1
- `MinCompletionPortThreads`: 1
- `MinWorkingThreads`: 1
- `OptionElements`: null
- `PerformanceDataCollectInterval`: 1000
- `ReceiveFilterFactories`: null
- `Servers`:
 - `base`: Count = 1
 - `결과 뷰`: 결과 뷰를 확장하면 IEnumerable이 열거됩니다.
 - `[0]`:
 - `LogBasicSessionActivity`: true
 - `LogCommand`: false
 - `LogFactory`: null
 - `MaxConnectionNumber`: 3000
 - `MaxRequestLength`: 1024
 - `Mode`: Tcp
 - `Name`: "ChatingServer"
 - `OptionElements`: null
 - `Options`: {System.Collections.Generic.List`1[System.Object]}
 - `Port`: 18732
 - `ReceiveBufferSize`: 16384
 - `ReceiveFilterFactory`: null
 - `Security`: "None"
 - `SendBufferSize`: 16384
 - `SendingQueueSize`: 5

네트워크 옵션 파라미터

루트 설정(모든 서버 네트워크에 적용)에 사용하는 파라미터 **IRootConfig**

maxWorkingThreads: maximum working threads count of .NET thread pool;

minWorkingThreads: minimum working threads count of .NET thread pool;

maxCompletionPortThreads: maximum completion threads count of .NET thread pool;

minCompletionPortThreads: minimum completion threads count of .NET thread pool;

disablePerformanceDataCollector: whether disable performance data collector;

performanceDataCollectInterval: performance data collecting interval (in seconds, default value: 60);

isolation: SuperSocket instances isolation level

None - no isolation

AppDomain - server instances will be isolated by AppDomains

Process - server instances will be isolated by processes

logFactory: the name of default logFactory, all log factories are defined in the child node "logFactories" which will be introduced in following documentation;

defaultCulture: default thread culture for the global application, only available in .Net 4.5;

서버 인스턴스 옵션 파라미터 **IServerconfig**

name: the name of the server instance;

serverType: the full name the AppServer's type which you want to run;

serverTypeName: the name of the selected server types, all server types should be defined in serverTypes node which will be introduced in following documentation;

ip: the ip of the server instance listens. You can set an exact ip, you also can set the below values Any - all IPv4 address IPv6Any - all IPv6 address

port: the port of the server instance listens;

listenBacklog: the listen back log size;

mode: the socket server's running mode, Tcp (default) or Udp;

disabled: whether the server instance is disabled;

startupOrder: the server instance start order, the bootstrap will start all server instances order by this value;

서버 인스턴스 옵션 파라미터

sendTimeOut: sending data timeout;

sendingQueueSize: the sending queue's maximum size;

maxConnectionNumber: maximum connection number the server instance allow to connect at the same time;

receiveBufferSize: receiving buffer size; 세션당

sendBufferSize: sending buffer size; 세션당

syncSend: sending data in sync mode, default value: false;

logCommand: whether log command execution record;

logBasicSessionActivity: whether log the session's basic activities like connected and closed;

clearIdleSession: true or false, whether clear idle sessions, default value is false;

clearIdleSessionInterval: the clearing timeout idle session interval, default value is 120, in seconds;

idleSessionTimeOut: The session timeout period. Default value is 300, in seconds;

security: Empty, Tls, Ssl3. The security option of the socket server, default value is empty;

서버 인스턴스 옵션 파라미터

maxRequestLength: The maximum allowed request length, default value is 1024;

textEncoding: The default text encoding in the server instance, default value is ASCII;

defaultCulture: default thread culture for this appserver instance, only available in .Net 4.5 and cannot be set if the isolation model is 'None';

disableSessionSnapshot: Indicate whether disable session snapshot, default value is false. 세션 수 기록

sessionSnapshotInterval: The interval of taking session snapshot, default value is 5, in seconds;

keepAliveTime: The interval of keeping alive, default value is 600, in seconds;

keepAliveInterval: The interval of retry after keep alive fail, default value is 60, in seconds;

Commnad-Line Protocol

"WrWn" 로 끝나는 라인 단위 문자열을 패킷 프로토콜로 사용할 수 있다.

문자열의 인코딩은 기본은 Ascii. UTF-8 등의 다른 인코딩으로 바꿀 수 있다.

```
public class StringRequestInfo
{
    public string Key { get; }

    public string Body { get; }

    public string[] Parameters { get; }

    //Other properties and methods
}
```

"LOGIN kerry 123456" + NewLine

Key: "LOGIN"

Body: "kerry 123456";

Parameters: ["kerry", "123456"]

```
public class LOGIN : CommandBase<AppSession, StringRequestInfo>
{
    public override void ExecuteCommand( AppSession session, StringRequestInfo requestInfo)
    {
        //Implement your business logic
    }
}
```

독자적으로 변경하기

"LOGIN:kerry,12345" + NewLine

```
public class YourServer : AppServer<YourSession>
{
    public YourServer()
        : base(new CommandLineReceiveFilterFactory(Encoding.Default,
                                                    new BasicRequestInfoParser(":", ",")))
    {
    }
}
```

```
public class YourServer : AppServer<YourSession>
{
    public YourServer()
        : base(new CommandLineReceiveFilterFactory(Encoding.Default,
                                                    new YourRequestInfoParser()))
    {
    }
}
```

AppSession 조작

데이터 보내기

```
session.Send(data, 0, data.Length);  
or  
session.Send("Welcome to use SuperSocket!");
```

AppServer에서 세션 찾기

- GetSessionByID 멤버를 사용한다.

```
var session = appServer.GetSessionByID(sessionID);  
if(session != null)  
    session.Send(data, 0, data.Length);
```

sessionID는 AppSession 객체를 생성할 때 GUID를 string으로 할당한다.
UDP의 경우 UdpRequestInfo를 사용하면 GUID로 만들고, 아니면 리모트의 IP와 Port로 만든다.

연결된 모든 세션에 메시지 보내기

```
foreach(var session in appServer.GetAllSessions())  
{  
    session.Send(data, 0, data.Length);  
}
```

자작용 Key로 세션들 찾기

- 아래의 CompanyId 처럼 새로운 Key를 사용하여 검색이 가능하다.

```
var sessions = appServer.GetSessions(s => s.CompanyId == companyId);  
foreach(var s in sessions)  
{  
    s.Send(data, 0, data.Length);  
}
```

Connection Filter

- IConnectionFactory라는 인터페이스를 통해서 접속한 클라이언트를 접속 허용할 건지 차단할 건지 정의할 수 있다. ip 범위를 지정하여 특정 ip 범위에서만 접속을 허용할 수 있다.

```
public class IPConnectionFactory : IConnectionFactory
{
    private Tuple<long, long>[] m_IpRanges;

    public bool Initialize(string name, IAppServer appServer)
    {
        Name = name;

        var ipRange = appServer.Config.Options.GetValue("ipRange");

        string[] ipRangeArray;

        if (string.IsNullOrEmpty(ipRange)
            || (ipRangeArray = ipRange.Split(new char[] { ',', ';' }, StringSplitOptions.RemoveEmptyEntries)).Length <= 0)
        {
            throw new ArgumentException("The ipRange doesn't exist in configuration!");
        }

        m_IpRanges = new Tuple<long, long>[ipRangeArray.Length];

        for (int i = 0; i < ipRangeArray.Length; i++)
        {
            var range = ipRangeArray[i];
            m_IpRanges[i] = GenerateIpRange(range);
        }

        return true;
    }
}
```

```
private Tuple<long, long> GenerateIpRange(string range)
{
    var ipArray = range.Split(new char[] { '-' }, StringSplitOptions.RemoveEmptyEntries);

    if(ipArray.Length != 2)
        throw new ArgumentException("Invalid ipRange exist in configuration!");

    return new Tuple<long, long>(ConvertIpToLong(ipArray[0]), ConvertIpToLong(ipArray[1]));
}

private long ConvertIpToLong(string ip)
{
    var points = ip.Split(new char[] { '.' }, StringSplitOptions.RemoveEmptyEntries);

    if(points.Length != 4)
        throw new ArgumentException("Invalid ipRange exist in configuration!");

    long value = 0;
    long unit = 1;

    for (int i = points.Length - 1; i >= 0; i--)
    {
        value += unit * points[i].ToInt32();
        unit *= 256;
    }

    return value;
}

public string Name { get; private set; }
```

```
public bool AllowConnect(IPEndPoint remoteAddress)
{
    var ip = remoteAddress.Address.ToString();
    var ipValue = ConvertIpToLong(ip);

    for (var i = 0; i < m_IpRanges.Length; i++)
    {
        var range = m_IpRanges[i];

        if (ipValue > range.Item2)
            return false;

        if (ipValue < range.Item1)
            return false;
    }

    return true;
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="superSocket" type="SuperSocket.SocketEngine.Configuration.SocketServiceConfig,
SuperSocket.SocketEngine"/>
  </configSections>
  <appSettings>
    <add key="ServiceName" value="EchoService"/>
  </appSettings>
  <superSocket>
    <servers>
      <server name="EchoServer"
serverTypeName="EchoService"
ip="Any" port="2012"
connectionFilter="IpRangeFilter"
ipRange="127.0.1.0-127.0.1.255">
    </server>
  </servers>
  <serverTypes>
    <add name="EchoService"
type="SuperSocket.QuickStart.EchoService.EchoServer, SuperSocket.QuickStart.EchoService" />
  </serverTypes>
  <connectionFilters>
    <add name="IpRangeFilter"
type="SuperSocket.QuickStart.ConnectionFilter.IPConnectionFilter, SuperSocket.QuickStart.ConnectionFilter" />
  </connectionFilters>
</superSocket>
<startup>
  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0" />
</startup>
</configuration>
```


다중 Listeners

- 하나의 서버인스턴스에서 복수의 listen을 할 수 있다.

```
<superSocket>
  <servers>
    <server name="EchoServer" serverTypeName="EchoService">
      <listeners>
        <add ip="127.0.0.2" port="2012" />
        <add ip="IPv6Any" port="2012" />
      </listeners>
    </server>
  </servers>
  <serverTypes>
    <add name="EchoService"
      type="SuperSocket.QuickStart.EchoService.EchoServer, SuperSocket.QuickStart.EchoService" />
  </serverTypes>
</superSocket>
```

```
<superSocket>
  <servers>
    <server name="EchoServer" serverTypeName="EchoService">
      <certificate filePath="localhost.pfx" password="supersocket"></certificate>
      <listeners>
        <add ip="Any" port="80" />
        <add ip="Any" port="443" security="tls" />
      </listeners>
    </server>
  </servers>
  <serverTypes>
    <add name="EchoService"
      type="SuperSocket.QuickStart.EchoService.EchoServer, SuperSocket.QuickStart.EchoService" />
  </serverTypes>
</superSocket>
```

동적 언어 지원

- 닷넷에서 지원하는 동적언어들은 SuperSocket을 사용할 수 있다.
- 대표적인 닷넷용 동적언어는 IronPython, Ironruby, F# 등이 있다.

The Built-in Common Format Protocol Implementation Templates

- <http://docs.supersocket.net/v1-6/en-US/The-Built-in-Common-Format-Protocol-Implementation-Templates>
- TerminatorReceiveFilter - Terminator Protocol
: 특정 지시어를 지정하여 패킷을 구분한다.
- CountSpliterReceiveFilter - Fixed Number Split Parts with Separator Protocol
: 특정 지시어로 구분된 단위의 크기를 숫자로 지정
- FixedSizeReceiveFilter - Fixed Size Request Protocol
: 고정된 바이너리 크기로 패킷을 구분한다.
- BeginEndMarkReceiveFilter - The Protocol with Begin and End Mark
: 시작과 끝을 구분하는 지시어를 사용하여 패킷을 구분한다.
- FixedHeaderReceiveFilter - Fixed Header with Body Length Protocol
: 헤더와 보디로 나누어서 이것들의 크기에 의해서 패킷을 구분한다.

Custome 프로토콜 정의(binary 기반)

- SuperSocket 예제를 보면 대부분 string 기반의 프로토콜을 사용하고 있으나 **binary 기반의 프로토콜을 정의해서 사용할 수 있다.**

1. BinaryRequestInfo 클래스와 FixedHeaderReceuveFilter 클래스를 재 정의 한다.

```
// 헤더는 4 바이트 정수값으로 key, 그 다음 body byte[]의 크기를 가리키는 4 바이트 정수값
public class EFBinaryRequestInfo : BinaryRequestInfo
{
    public int nKey
    {
        get;
        private set;
    }

    public EFBinaryRequestInfo(int nKey, byte[] body)
        : base(null, body)
    {
        this.nKey = nKey;
    }
}
```

```

class ReceiveFilter : FixedHeaderReceiveFilter<EFBinaryRequestInfo>
{
    public ReceiveFilter()
        : base(8)
    {

    }

    protected override int GetBodyLengthFromHeader(byte[] header, int offset, int length)
    {
        if (!BitConverter.IsLittleEndian)
            Array.Reverse(header, offset + 4, 4);

        var nBodySize = BitConverter.ToInt32(header, offset+4);
        return nBodySize;
    }

    protected override EFBinaryRequestInfo ResolveRequestInfo(ArraySegment<byte> header,
                                                                byte[] bodyBuffer, int offset, int length)
    {
        if (!BitConverter.IsLittleEndian)
            Array.Reverse(header.Array, 0, 4);

        return new EFBinaryRequestInfo(BitConverter.ToInt32(header.Array, 0), bodyBuffer.CloneRange(offset,
length));
    }
}

```

2. 패킷 핸들러 정의

```
public class PacketData
{
    public NetworkSession session;
    public EFBinaryRequestInfo reqInfo;
}
```

```
public enum PACKETID : int
{
    REQ_DUMMY_CHAT = 1,
    REQ_LOGIN      = 11,
}
```

```
public class CommonHandler
{
    public void RequestLogin(NetworkSession session, EFBinaryRequestInfo requestInfo)
    {
    }

    public void RequestDummyChat(NetworkSession session, EFBinaryRequestInfo requestInfo)
    {
        string jsonstring = System.Text.Encoding.GetEncoding("utf-8").GetString(requestInfo.Body);
        var deserializedProduct = JsonConvert.DeserializeObject<PK_CHAT>(jsonstring);
        session.Send(deserializedProduct.sender + ":" + deserializedProduct.msg);
    }
}
```

```
public class PK_LOGON
{
    public string ID;
    public string PW;
}
```

```
public class PK_CHAT
{
    public string sender;
    public string msg;
}
```

3. 데이터 받기 이벤트 등록 및 프로토콜 해석하기

```
.....
var HandlerMap = new Dictionary<int, Action<NetworkSession, EFBinaryRequestInfo>>();
CommonHandler CommonHan = new CommonHandler();
.....

public BoardServerNet()
    : base(new DefaultReceiveFilterFactory<ReceiveFilter, EFBinaryRequestInfo>())
{
    NewSessionConnected += new SessionHandler<NetworkSession>(OnConnected);
    SessionClosed += new SessionHandler<NetworkSession, CloseReason>(OnClosed);
    NewRequestReceived += new RequestHandler<NetworkSession, EFBinaryRequestInfo>(RequestReceived);
}

public void RegistHandler()
{
    HandlerMap.Add( (int)PACKETID.REQ_LOGIN, CommonHan.RequestLogin );
    HandlerMap.Add((int)PACKETID.REQ_DUMMY_CHAT, CommonHan.RequestDummyChat);
}

public void StartPacketThread()
{
    IsRunningPacketThread = true;
    PakcetThread = new System.Threading.Thread(this.ProcessPacket);
    PakcetThread.Start();
}
```

```
public void ProcessPacket()
{
    while (IsRunningPacketThread)
    {
        PacketData packet;

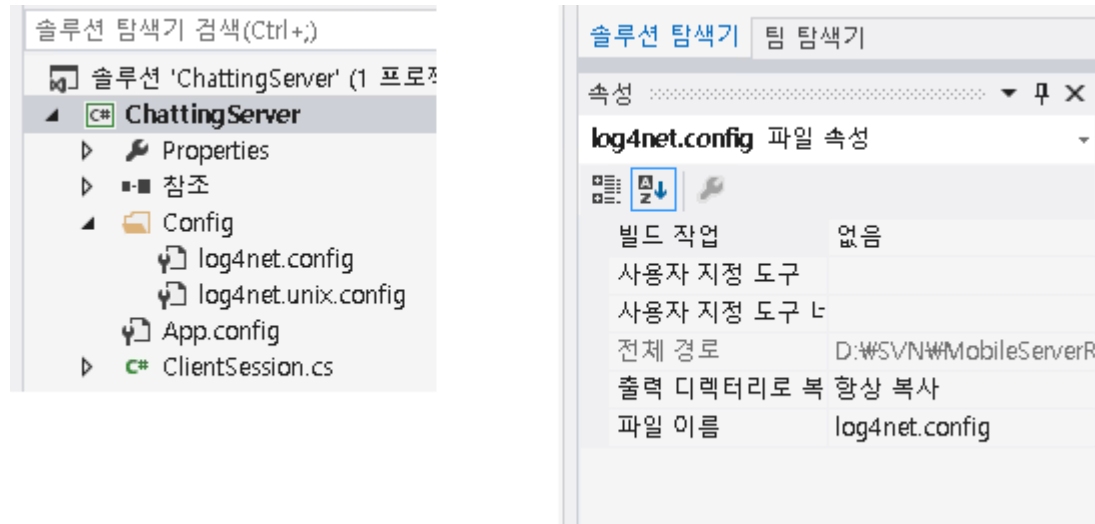
        if (PacketQueue.TryDequeue(out packet))
        {
            var PacketID = packet.reqInfo.nKey;

            if (HandlerMap.ContainsKey(PacketID))
            {
                HandlerMap[PacketID](packet.session, packet.reqInfo);
            }
        }

        System.Threading.Thread.Sleep(1);
    }
}
```


로그 시스템

- log4net 라이브러리를 사용한다.
: 설정 파일에 따로 설정하지 않으면 'log4NetLogFactory' 생성된다.
- 로그 관련 설정 파일은 log4net.config 혹은 log4nte.unix.config 이다. 이 파일은 'Config' 라는 폴더 안에 있어야 한다.



로그 설정 파일을 솔루션에 등록하고, 출력 디렉토리로 복사하도록 하면 빌드 할 때마다 아래처럼 복사해 준다.

로컬 디스크 (D:) > SVN > MobileServerRND > trunk > Server > SuperSocketDemo > ChattingServer > bin > Debug > Config			
이름	수정한 날짜	유형	크기
log4net.config	2013-10-28 오전...	CONFIG 파일	3KB
log4net.unix.config	2013-10-28 오전...	CONFIG 파일	3KB

- 로그 시스템 사용을 위해 SuperSocket에 있는 log4net.dll 이 필요하다.
- log4net의 객체를 가져오는 방법

```
log4net.ILog log = log4net.LogManager.GetLogger(  
System.Reflection.MethodBase.GetCurrentMethod().DeclaringType);
```

- 직접 만든 로그 시스템을 사용하고 싶으면 ILogFactory와 ILog 인터페이스를 구현한다.

```
public class RemoteProcessSession : AppSession<RemoteProcessSession>
{
    protected override void HandleUnknownRequest(StringRequestInfo requestInfo)
    {
        Logger.Error("Unknow request");
    }
}
```

```
<appender name="myBusinessAppender">
    <!--Your appender details-->
</appender>
<logger name="MyBusiness" additivity="false">
    <level value="ALL" />
    <appender-ref ref="myBusinessAppender" />
</logger>
```

```
var myLogger = server.LogFactory.GetLog("MyBusiness");
```

Azure, Mono, 그 외

- Work Role을 사용하면 손쉽게 Azure에서 사용할 수 있다.
- Mono 기반을 사용하여 Unix/Linux에서 사용할 수 있다. Mono 2.10 이상 필요.
- 1.6 버전에서는 설정에 의해서 클라이언트와 주고 받는 텍스트 메시지의 포맷을 UTF-8 이외의 것으로 설정할 수 있다.
- 닷넷플랫폼 4.5 이상이라면 각 서버 인스턴스 별로 defaultCulture 설정 가능.
- Process level isolation 에 의해 하나의 애플리케이션에서 복수의 인스턴스를 생성하는 경우 각 인스턴스 별로 프로세스를 만들 수 있다.

Ubuntu + mono에서 사용

- Linux에서는 mono를 사용하면 SuperSocket으로 만든 프로그램을 실행 가능(다만 mono가 지원하는 닷넷 라이브러리를 사용해야 한다)
- 실행 방법
Windows으로 빌드한 파일을 그대로 가져와서 실행한다.
"mono 실행파일 이름.exe"
실행 권한만 있다면 **"실행파일 이름.exe"** 만으로도 실행 가능
- 클라이언트와 서버간에 CommandText를 사용할 때 에러가 발생할 수 있음.
이유는 Windows와 Linux 간의 개행 코드가 다르기 때문.
Windows는 CRLF, Linux는 LF

SuperSocket을 사용한 SuperWebSocket의 경우

```
session.SocketSession.SendResponse(responseBuilder.ToString());
```

->

```
session.SocketSession.SendResponse(responseBuilder.ToString().Replace(Environment.NewLine, "\r\n"));
```

SuperSocket ServerManager

- 서버 모니터링 컴포넌트를 지원한다.
- 현재 클라이언트는 실버라이트, WPF용 클라이언트를 지원한다.

The screenshot displays the 'SuperSocket ServerManager Client' window. At the top, it shows 'localhost [Connected]'. Below this, system metrics are listed: 'CPU Usage: 0.01%', 'Physical Memory Usage: 60,9', 'Available Completion Port Threads: 1000', and 'Maximum Working Threads:'. A table lists two servers, 'ServerA' and 'ServerB', both with 'Is Running' status set to 'True'. A red circle highlights a context menu over the 'Is Running' column, with 'Start' and 'Stop' options visible.

Name	Started Time	Is Running	Total Connections	Maximum Allowed Connection Nu
■ ServerA	8/19/2013 4:13:15 PM	True	0	100
■ ServerB	8/19/2013 4:13:15 PM	True	0	100

SuperSocket을 더 잘 이해하려면

혹은 C# 고성능 네트워크 프로그래밍을 하려면

- C# SocketAsyncEventArgs High Performance Socket Code
<http://www.codeproject.com/Articles/83102/C-SocketAsyncEventArgs-High-Performance-Socket-Cod>
- (e-book)유니티 개발자를 위한 C#으로 온라인 게임 서버 만들기
http://www.hanbit.co.kr/realtime/books/book_view.html?p_code=E6015792502

강연 문서와 예제 코드는 아래에...

<https://github.com/jacking75/kgc2016> SuperSocket