Command and Command Loader

English (United States) v1.6

Keywords: Command, Command Loader, Multiple Command Assemblies

Command

Command in SuperSocket is designed to handle the requests coming from the clients, it play an important role in the business logic processing.

Command class must implement the basic command interface below:

```
public interface ICommand<TAppSession, TRequestInfo> : ICommand
    where TRequestInfo : IRequestInfo
    where TAppSession : IAppSession
{

    void ExecuteCommand(TAppSession session, TRequestInfo requestInfo);
}

public interface ICommand
{
    string Name { get; }
}
```

The request processing code should be placed in the method "ExecuteCommand(TAppSession session, TRequestInfo requestInfo)" and the property "Name" is used for matching the received requestInfo. When a requestInfo instance is received, SuperSocket will look for the command which has the responsibility to handle it by matching the requestInfo's Key and the command's name.

For a instance, if we receive a requestInfo like below:

```
Key: "ADD"
Body: "1 2"
```

Then SuperSocket will looking a command whose name is "ADD". If we have a command defined below:

```
public class ADD : StringCommandBase
{
    public override void ExecuteCommand(AppSession session, StringRequestInfo requestInfo)
    {
          session.Send((int.Parse(requestInfo[0] + int.Parse(requestInfo[1])).ToString());
    }
}
```

Then this command will be found, because the StringCommandBase instance's name is filled by the class's name.

But in some cases, the requestInfo's Key cannot be used as a class name. For example:

```
Key: "01"
Body: "1 2"
```

To make your ADD command work, you need to override the name attribute of the command class:

```
public class ADD : StringCommandBase
{
    public override string Name
    {
        get { return "01"; }
    }

    public override void ExecuteCommand(AppSession session, StringRequestInfo requestInfo)
    {
          session.Send((int.Parse(requestInfo[0] + int.Parse(requestInfo[1])).ToString());
    }
}
```

Command assemblies definition

Yes, it uses reflection to find public classes who implement the basic command interface, but it only look for from the assembly where your AppServer class is defined.

For example, your AppServer is defined in the assembly GameServer.dll, but your command ADD is defined in the assembly BasicModules.dll:

```
GameServer.dll
    + MyGameServer.cs
```

```
BasicModules.dll
    + ADD.cs
```

By default, the command "ADD" cannot be loaded into the game server instance. If you want to load the command, you should add the assembly BasicModules.dll into command assemblies in the configuration:

```
<?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="BroardcastService"/>
    </appSettings>
    <superSocket>
        <servers>
            <server name="SampleServer"
                    serverType="GameServer.MyGameServer, GameServer"
                    ip="Any" port="2012">
              <commandAssemblies>
                <add assembly="BasicModules"></add>
              </commandAssemblies>
            </server>
        </servers>
    </superSocket>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0" />
    </startup>
</configuration>
```

You also can add more command assemblies in the configuration.

Command Loader

In some cases, you prefer some special logic operations to return commands instead of by automatically reflection, you should implement a command loader by yourself:

```
public interface ICommandLoader<TCommand>
```

And then configure your server to use the command loader:

```
<superSocket>
    <servers>
      <server name="SampleServer"
              serverType="GameServer.MyGameServer, GameServer"
              ip="Any" port="2012"
              commandLoader="MyCommandLoader">
      </server>
    </servers>
    <commandLoaders>
        <add name="MyCommandLoader"
           type="GameServer.MyCommandLoader, GameServer" />
    </commandLoaders>
</superSocket>
```


- Prev: Implement Your Own Communication Protocol with IRequestInfo, IReceiveFilter and etc (/v1-6/en-US/Implement-Your-Own-Communication-Protocol-with-IRequestInfo,-IReceiveFilter-and-etc)
- Next: Get the Connected Event and Closed Event of a Connection (/v1-6/en-US/Get-the-connected-event-and-closed-event-of-a-connection)