The Built-in Command Line Protocol

English (United States) v1.6

Keywords: Command Line, Protocol, StringRequestInfo, Text Encoding

What's the Protocol?

What's the Protocol? Lots of people probably will answer "TCP" or "UDP". But to build a network application, only TCP or UDP is not enough. TCP and UDP are transport-layer protocols. It's far from enough to enable talking between two endpoints in the network if you only define transport-layer protocol. You need to define your application level protocol to convert your received binary data to the requests which your application can understand.

The Built-in Command Line Protocol

The command line protocol is a widely used protocols, lots of protocols like Telnet, SMTP, POP3 and FTP protocols are base on command line protocol etc. If you do not have a custom protocol, then SuperSocket will use command line protocol by default, which can simplify the development of this kind of protocols.

The command line protocol defines each request must be ended with a carriage return "\r\n".

If you use the command line protocol in SuperSocket, all requests will to translated into StringRequestInfo instances.

StringRequestInfo is defined like this:

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

    public string Body { get; }

    public string[] Parameters { get; }

    /*
    Other properties and methods
    */
}
```

Because the built-in command line protocol in SuperSocket uses a space to split request key and parameters, So when the client sends the data below to the server:

```
"LOGIN kerry 123456" + NewLine
```

the SuperSocket server will receive a StringRequestInfo instance, the properties of the request info instance will be:

```
Key: "LOGIN"
Body: "kerry 123456";
Parameters: ["kerry", "123456"]
```

If you have defined a Command with name "LOGIN", the command's ExecuteCommand method will be excuted with the StringRequestInfo instance as parameter:

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

Customize the Command Line Protocol

Some users might have different request format, for instance:

```
"LOGIN:kerry,12345" + NewLine
```

The request's key is separated with body by the char ':', and the parameters are separated by the char ','. This kind of request can be supported easily, just extend the command line protocol like the below code:

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

    }
}
```

If you want to customize the request format much deeper, you can implement a RequestInfoParser class base the interface IRequestInfoParser, and then pass in your own RequestInfoParser instance when instantiate the CommandLineReceiveFilterFactory instance:

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

    }
}
```

Text Encoding

The default encoding of the command line protocol is Ascii, but you can change it in the configuration by setting the **"textEncoding"** attribute of the server node:

```
<server name="TelnetServer"
      textEncoding="UTF-8"
      serverType="YourAppServer, YourAssembly"
      ip="Any" port="2020">
</server>
```

- Prev: SuperSocket Basic Configuration (/v1-6/en-US/SuperSocket-Basic-Configuration)
- Next: The Built-in Common Format Protocol Implementation Templates (/v1-6/en-US/The-Built-in-Common-Format-Protocol-Implementation-Templates)

METTLER TOLEDO

Reliable pH Control
Learn the Theory of pH Measurement, Download the Free pH Theory Guide.

OPEN

© 2019 - GetDocs.Net - Hosted by BuyVM (https://my.frantech.ca/aff.php?aff=2012)