Implement Your Own Communication Protocol with IRequestInfo, IReceiveFilter and etc

English (United States) ▾ v1.6 ▾

Keywords: Protocol Customization, IRequestInfo, IReceiveFilter, ReceiveFilterFactory

Why do you want to use Your Own Communication Protocol?

The communication protocol is used for converting your received binary data to the requests which your application can understand. SuperSocket provides a built-in communication protocol "Command Line Protocol" which defines each request must be ended with a carriage return "\r\n".

But some applications cannot use "Command Line Protocol" for many different reasons. In this case, you need to implement your own communication protocol using the tools below:

```
* RequestInfo
* ReceiveFilter
* ReceiveFilterFactory
* AppServer and AppSession
```

The RequestInfo

RequestInfo is the entity class which represents a request from the client. Each request of client should be instantiated as a RequestInfo. The RequestInfo class must implement the interface IRequestInfo which only have a property named "Key" in string type:

```
public interface IRequestInfo
{
    string Key { get; }
}
```

Talked in the previous documentation, The request info class StringRequestInfo is used in SuperSocket command line protocol.

You also can implement your own RequestInfo class as your application requirement. For instance, if all of your requests must have a DeviceID field, you can define a property for it in the RequestInfo class:

```
public class MyRequestInfo : IRequestInfo
{
     public string Key { get; set; }

     public int DeviceId { get; set; }

     /*
     // Other properties
     */
}
```

SuperSocket also provides another request info class "BinaryRequestInfo" used for binary protocol:

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

    public byte[] Body { get; }
}
```

You can use BinaryRequestInfo directly if it can satisfy your requirement.

The ReceiveFilter

The ReceiveFilteris used for converting received binary data to your request info instances.

To implement a ReceiveFilter, you need to implement the interface IReceiveFilter:

```
public interface IReceiveFilter<TRequestInfo>
    where TRequestInfo : IRequestInfo
{
    /// <summary>
    /// Filters received data of the specific session into request info.
    /// </summary>
    /// <param name="readBuffer">The read buffer.</param>
    /// <param name="offset">The offset of the current received data in this read buffer.</param>
    /// <param name="length">The length of the current received data.</param>
    /// <param name="toBeCopied">if set to <c>true</c> [to be copied].</param>
    /// <param name="rest">The rest, the length of the data which hasn't been parsed.</param>
    /// <returns></returns>
    TRequestInfo Filter(byte[] readBuffer, int offset, int length, bool toBeCopied, out int rest);

    /// <summary>
    /// Gets the size of the left buffer.
    /// </summary>
    /// <value>
    /// The size of the left buffer.
    /// </value>
    int LeftBufferSize { get; }

    /// <summary>
    /// Gets the next receive filter.
    /// </summary>
    IReceiveFilter<TRequestInfo> NextReceiveFilter { get; }

    /// <summary>
    /// Resets this instance to initial state.
    /// </summary>
    void Reset();
}
```

- TRequestInfo: the type parameter "TRequestInfo" is the request info class you want to use in the application
- LeftBufferSize: the data size which is cached in this request filter;
- NextReceiveFilter: the request filter which will be used when next piece of binary data is received;
- Reset(): resets this instance to initial state;
- Filter(....): the filter method is executed when a piece of binary data is received by SuperSocket, the received data locates in the parameter readBuffer. Because the readBuffer is shared by all connections in the same appServer instance, so you need to load the received data from the position "offset"(method parameter) and with the size "length" (method parameter).

  ```
  TRequestInfo Filter(byte[] readBuffer, int offset, int length, bool toBeCopied, out int rest);
  ```

  - readBuffer: the receiving buffer, the received data is stored in this array
  - offset: the received data's start position in the readBuffer
  - length: the length of the received data
  - toBeCopied: indicate whether should create a copy of the readBuffer instead of use it directly when we want to cache data in it
  - rest: it's a output parameter, it should be set to be the remaining received data size after you find a full request

There are many cases you need to handle:

- If you find a full request from the received data, your must return a request info instance of your request info type.
- If you haven't find a full request, you just return NULL.
- If you have find a full request from the received data, but the received data not only contain one request, set the remaining data size to the output parameter "rest". SuperSocket will examine the output parameter "rest", if it is bigger than 0, the Filter method will be executed again with the parameters "offset" and "length" adjusted.

The ReceiveFilterFactory

The ReceiveFilterFactory is used for creating receive filter for each session. To define you receive filter factory class, you must implement the interface IReceiveFilterFactory. The type parameter "TRequestInfo" is the request info class you want to use in the application

```
/// <summary>
/// Receive filter factory interface
/// </summary>
/// <typeparam name="TRequestInfo">The type of the request info.</typeparam>
public interface IReceiveFilterFactory<TRequestInfo> : IReceiveFilterFactory
    where TRequestInfo : IRequestInfo
{
    /// <summary>
    /// Creates the receive filter.
    /// </summary>
    /// <param name="appServer">The app server.</param>
    /// <param name="appSession">The app session.</param>
    /// <param name="remoteEndPoint">The remote end point.</param>
    /// <returns>
    /// the new created request filer assosiated with this socketSession
    /// </returns>
    IReceiveFilter<TRequestInfo> CreateFilter(IAppServer appServer, IAppSession appSession, IPEndPoint remoteEndPoint);
}
```

You also can use the default receive filter factory

```
DefaultReceiveFilterFactory<TReceiveFilter, TRequestInfo>
```

, it will return the TReceiveFilter instance which is instantiated by the non-parameter constructor of class TReceiveFilter when method CreateFilter is invoked.

Work together with AppSession and AppServer

Now, you have RequestInfo, ReceiveFilter and ReceiveFilterFactory, but you haven't started use them. If you want to make them available in your application, you need to define your AppSession and AppServer using your created RequestInfo, ReceiveFilter and ReceiveFilterFactory.

- Set RequestInfo for AppSession

```
public class YourSession : AppSession<YourSession, YourRequestInfo>
{
     //More code...
}
```

- Set RequestInfo and ReceiveFilterFactory for AppServer

```
public class YourAppServer : AppServer<YourSession, YourRequestInfo>
{
    public YourAppServer()
        : base(new YourReceiveFilterFactory())
    {

    }
}
```

After finish these two things, your custom communication protocol should work now.

- Prev: The Built-in Common Format Protocol Implementation Templates (/v1-6/en-US/The-Built-in-Common-Format-Protocol-Implementation-Templates)
- Next: Command and Command Loader (/v1-6/en-US/Command-and-Command-Loader)