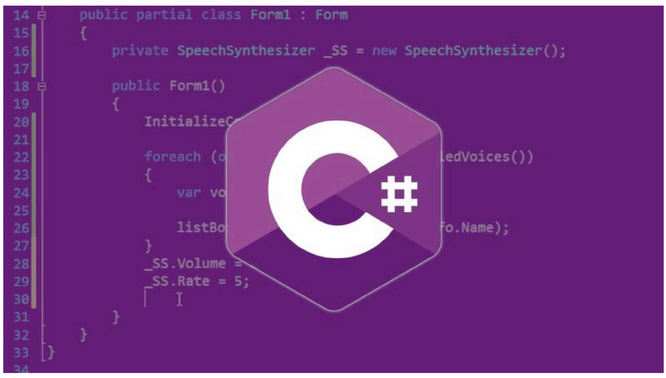
C# Socket通信：灵活传输DTO对象的完整实现

作者：微信公众号：【架构师老卢】 12-28 8:18



概述： 通过C#中的Socket通信，我们实现了一种灵活、通用的数据传输方法，将DTO对象序列化为字节数组，并在通信中采用统一的传输对象。服务端根据数据类型动态解析DTO对象，使得不同类型的数据能够轻松传递，提高了代码的灵活性。

1. 原理说明

在C#中使用Socket进行数据发送和接收时，可以通过序列化和反序列化来实现DTO对象的传输。一种常见的做法是将DTO对象序列化为字节数组，发送到服务端，然后服务端接收字节数组并进行反序列化，得到原始的DTO对象。

2. 方法说明

2.1 DTO对象序列化与反序列化

使用C#内置的`BinaryFormatter`进行对象的序列化和反序列化。这需要确保DTO对象是可序列化的（即标记为`[Serializable]`）。

```
[Serializable]
public class MyDTO
{
    public int Id { get; set; }
    public string Name { get; set; }
    // Add other properties as needed
}
```

2.2 定义通用数据传输格式

为了实现通用性，可以定义一个包含数据类型和对象数据的通用传输对象。

```
[Serializable]
public class TransferObject
{
    public string DataType { get; set; }
    public byte[] Data { get; set; }
}
```

3. 步骤说明

3.1 客户端发送DTO对象

1. 创建DTO对象实例。
2. 序列化DTO对象为字节数组。
3. 创建`TransferObject`，设置`DataType`为DTO对象类型，`Data`为序列化后的字节数组。
4. 将`TransferObject`序列化为字节数组。
5. 使用Socket发送字节数组到服务端。

3.2 服务端接收并解析DTO对象

1. 使用Socket接收字节数组。
2. 反序列化得到`TransferObject`。
3. 根据`DataType`将`Data`反序列化为具体的DTO对象。
4. 将DTO对象传递给业务逻辑进行处理。

4. 实例源代码

4.1 服务端代码

```
// 服务器端

// 在服务器端定义一个异步方法用于接收客户端数据
private async Task HandleClient(Socket clientSocket)
{
    // 接收数据
    byte[] buffer = new byte[1024];
    int bytesRead = await clientSocket.ReceiveAsync(new ArraySegment<byte>(buffer), SocketFlags.None);

    // 反序列化TransferObject
    TransferObject transferObject;
    using (MemoryStream stream = new MemoryStream(buffer, 0, bytesRead))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        transferObject = (TransferObject)formatter.Deserialize(stream);
    }

    // 根据DataType反序列化具体DTO对象
    Type objectType = Type.GetType(transferObject.DataType);
    object dataObject;
    using (MemoryStream stream = new MemoryStream(transferObject.Data))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        dataObject = formatter.Deserialize(stream);
    }

    // 处理业务逻辑，这里假设有一个HandleData方法
    HandleData(dataObject);

    // 关闭客户端连接
    clientSocket.Close();
}

// 服务器启动代码
public void StartServer()
{
    Socket listener = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    listener.Bind(new IPEndPoint(IPAddress.Any, 12345));
    listener.Listen(10);

    while (true)
    {
        Socket clientSocket = await listener.AcceptAsync();
        _ = HandleClient(clientSocket);
    }
}
```

4.2 客户端代码

```
// 客户端

// 创建DTO对象
MyDTO myDto = new MyDTO
{
    Id = 1,
    Name = "Example"
    // Set other properties
};

// 序列化DTO对象为字节数组
byte[] dataBytes;
using (MemoryStream stream = new MemoryStream())
{
    BinaryFormatter formatter = new BinaryFormatter();
    formatter.Serialize(stream, myDto);
    dataBytes = stream.ToArray();
}

// 创建TransferObject
TransferObject transferObject = new TransferObject
{
    DataType = typeof(MyDTO).AssemblyQualifiedName,
    Data = dataBytes
};

// 将TransferObject序列化为字节数组
byte[] transferBytes;
using (MemoryStream stream = new MemoryStream())
{
    BinaryFormatter formatter = new BinaryFormatter();
    formatter.Serialize(stream, transferObject);
    transferBytes = stream.ToArray();
}

// 创建Socket连接服务器
using (Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp))
{
    clientSocket.Connect(IPAddress.Parse("127.0.0.1"), 12345);

    // 发送数据到服务器
    await clientSocket.SendAsync(new ArraySegment<byte>(transferBytes), SocketFlags.None);
}
```

5. 注意事项及建议

- DTO对象需要标记为可序列化（使用`[Serializable]`标记）。
- 在实际应用中，要确保DTO对象的属性和顺序一致，以避免反序列化时出现问题。
- 考虑使用异步方法，如`ReceiveAsync`，以提高性能和并发处理能力。

说明

通过将DTO对象序列化为字节数组，并使用通用的传输对象包装数据类型和字节数组，可以实现在C#中使用Socket进行数据发送和接收的通用方法。服务端根据接收到的数据类型进行反序列化，得到原始的DTO对象，然后传递给业务逻辑进行处理。这种方法可以适用于不同类型的DTO对象，提高了代码的灵活性和可维护性。