

.NET SOCKET

Сергеев Николай КБ-401

Сóкеты (англ. *socket* — углубление, гнездо, разъём) — программный интерфейс для обеспечения обмена данными между процессами.

Сокет Беркли

- Серверные
- Клиентские

System.Net

System.Net.Sockets

- Windows Sockets
- Дуплексное взаимодействие

Socket

- Синхронная передача
- Асинхронная передача

Конструктор

```
Socket sock = new  
Socket(AddressFamily.InterNetwork,  
        SocketType.Stream,  
        ProtocolType.Tcp);
```

Асинхронные методы

- `ConnectAsync`
- `SendAsync`
- `ReceiveAsync`
- `Shutdown`
- `Close`

SocketAsyncEventArgs

```
...
SocketAsyncEventArgs socketEventArgs = new SocketAsyncEventArgs();

DnsEndPoint hostEntry = new DnsEndPoint("http://www.google.com", 80);

socketEventArgs.Completed += new
    EventHandler<SocketAsyncEventArgs>(SocketEventArgs_Completed);

socketEventArgs.RemoteEndPoint = hostEntry;

socketEventArgs.UserToken = sock;

sock.ConnectAsync(socketEventArgs);
...
```

```
...
SocketAsyncEventArgs socketEventArgs = new SocketAsyncEventArgs();

DnsEndPoint hostEntry = new DnsEndPoint("http://www.google.com", 80);

socketEventArgs.Completed += new
    EventHandler<SocketAsyncEventArgs>(SocketEventArgs_Completed);

socketEventArgs.RemoteEndPoint = hostEntry;

socketEventArgs.UserToken = sock;

sock.ConnectAsync(socketEventArgs);
...
```

```
...  
SocketAsyncEventArgs socketEventArgs = new SocketAsyncEventArgs();  
  
DnsEndPoint hostEntry = new DnsEndPoint("http://www.google.com", 80);  
  
socketEventArgs.Completed += new  
    EventHandler<SocketAsyncEventArgs>(SocketEventArgs_Completed);  
  
socketEventArgs.RemoteEndPoint = hostEntry;  
  
socketEventArgs.UserToken = sock;  
  
sock.ConnectAsync(socketEventArgs);  
...
```

...

```
SocketAsyncEventArgs socketEventArgs = new SocketAsyncEventArgs();
```

```
DnsEndPoint hostEntry = new DnsEndPoint("http://www.google.com", 80);
```

```
socketEventArgs.Completed += new  
    EventHandler<SocketAsyncEventArgs>(SocketEventArgs_Completed);
```

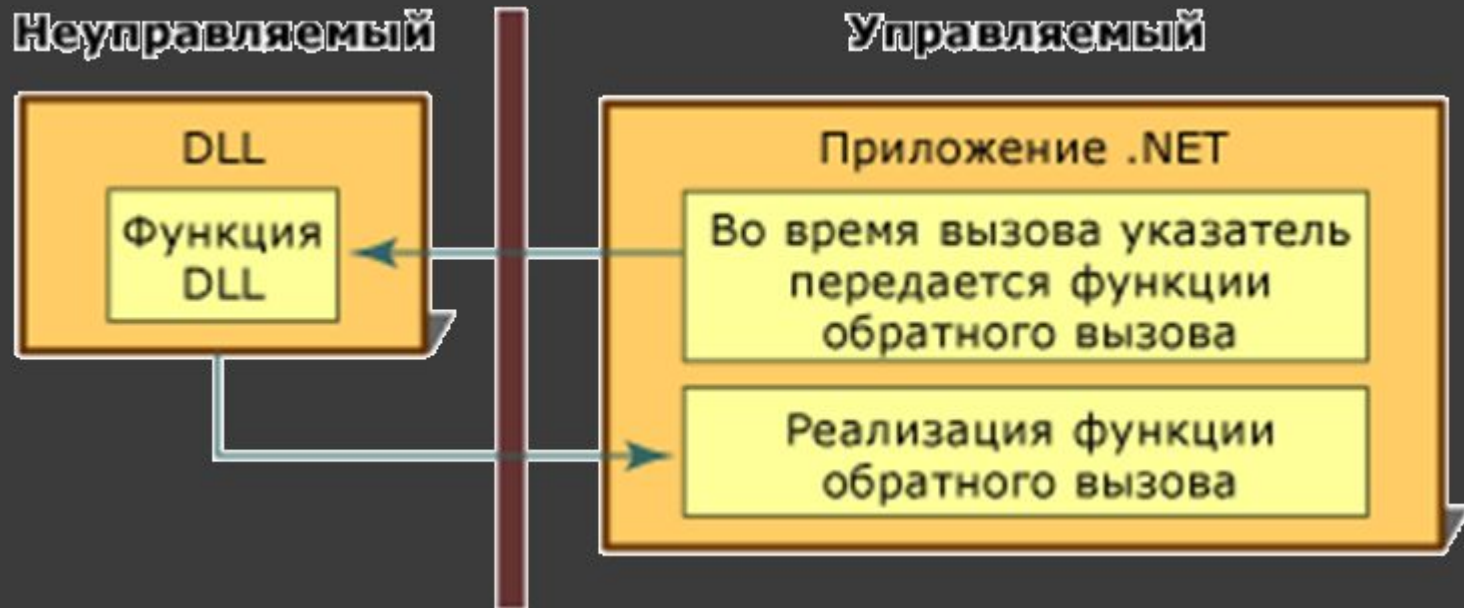
```
socketEventArgs.RemoteEndPoint = hostEntry;
```

```
socketEventArgs.UserToken = sock;
```

```
sock.ConnectAsync(socketEventArgs);
```

...

Функция обратного вызова



```
...
SocketAsyncEventArgs socketEventArgs = new SocketAsyncEventArgs();

DnsEndPoint hostEntry = new DnsEndPoint("http://www.google.com", 80);

socketEventArgs.Completed += new
    EventHandler<SocketAsyncEventArgs>(SocketEventArgs_Completed);

socketEventArgs.RemoteEndPoint = hostEntry;

socketEventArgs.UserToken = sock;

sock.ConnectAsync(socketEventArgs);
...
```

```
...
SocketAsyncEventArgs socketEventArgs = new SocketAsyncEventArgs();

DnsEndPoint hostEntry = new DnsEndPoint("http://www.google.com", 80);

socketEventArgs.Completed += new
    EventHandler<SocketAsyncEventArgs>(SocketEventArgs_Completed);

socketEventArgs.RemoteEndPoint = hostEntry;

socketEventArgs.UserToken = sock;

sock.ConnectAsync(socketEventArgs);
...
```



```
...
SocketAsyncEventArgs socketEventArgs = new SocketAsyncEventArgs();

DnsEndPoint hostEntry = new DnsEndPoint("http://www.google.com", 80);

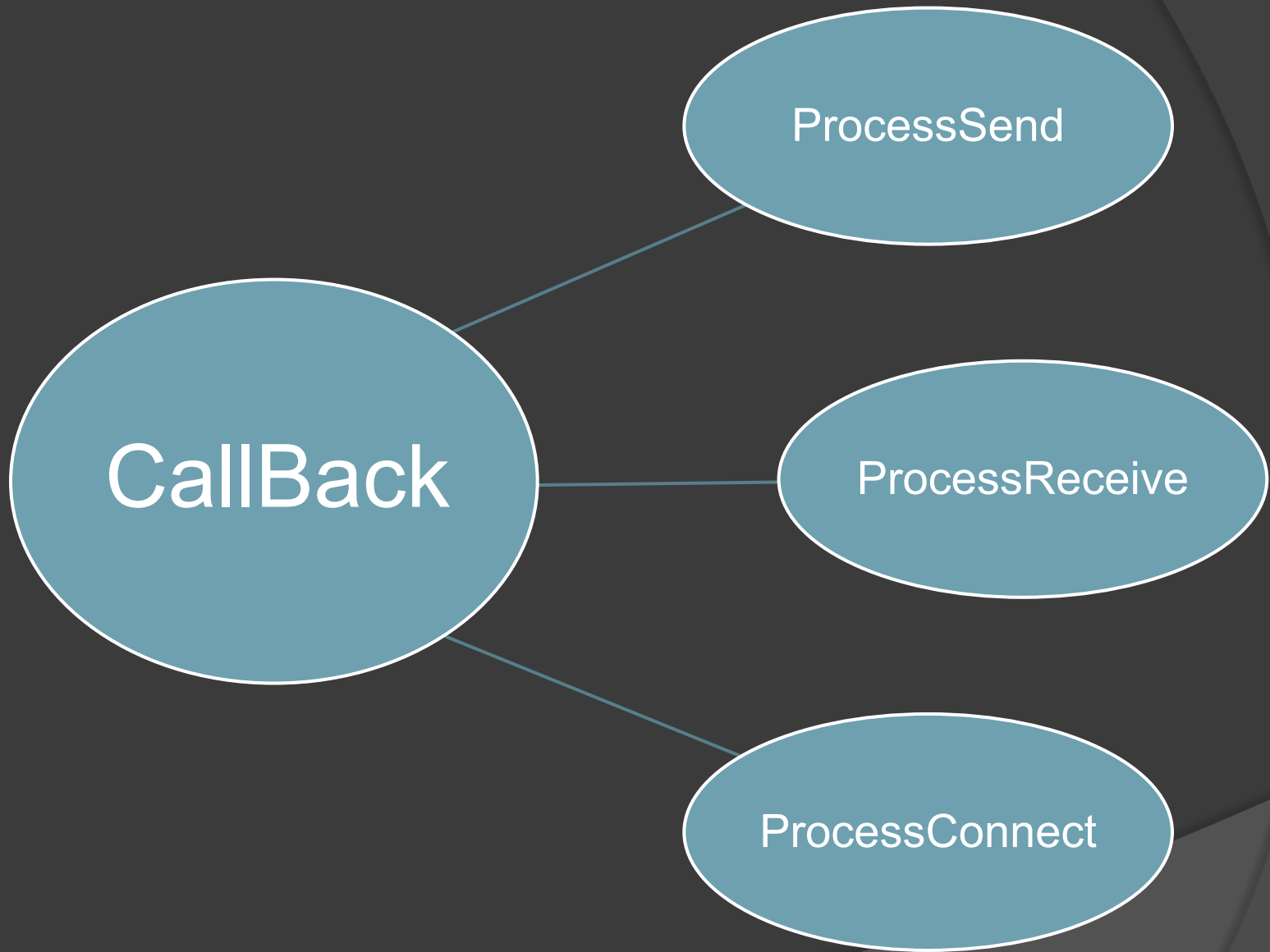
socketEventArgs.Completed += new
    EventHandler<SocketAsyncEventArgs>(SocketEventArgs_Completed);

socketEventArgs.RemoteEndPoint = hostEntry;

socketEventArgs.UserToken = sock;

sock.ConnectAsync(socketEventArgs);
...
```

```
static void SocketEventArgs_Completed(object sender,
SocketEventArgs e) {
    switch (e.LastOperation)
    {
        case SocketAsyncOperation.Connect:
            ProcessConnect(e);
            break;
        case SocketAsyncOperation.Receive:
            ProcessReceive(e);
            break;
        case SocketAsyncOperation.Send:
            ProcessSend(e);
            break;
        default: throw new Exception("Invalid operation completed"); }
    }
```



CallBack

ProcessSend

ProcessReceive

ProcessConnect

```
static void SocketEventArgs_Completed(object sender,
SocketAsyncEventArgs e) {
    switch (e.LastOperation)
    {
        case SocketAsyncOperation.Connect:
            ProcessConnect(e);
            break;
        case SocketAsyncOperation.Receive:
            ProcessReceive(e);
            break;
        case SocketAsyncOperation.Send:
            ProcessSend(e);
            break;
        default: throw new Exception("Invalid operation completed"); }
    }
```

```
static void SocketEventArgs_Completed(object sender,
SocketEventArgs e) {
    switch (e.LastOperation)
    {
        case SocketAsyncOperation.Connect:
            ProcessConnect(e);
            break;
        case SocketAsyncOperation.Receive:
            ProcessReceive(e);
            break;
        case SocketAsyncOperation.Send:
            ProcessSend(e);
            break;
        default: throw new Exception("Invalid operation completed"); }
    }
```

```
static void SocketEventArgs_Completed(object sender,
SocketAsyncEventArgs e) {
    switch (e.LastOperation)
    {
        case SocketAsyncOperation.Connect:
            ProcessConnect(e);
            break;
        case SocketAsyncOperation.Receive:
            ProcessReceive(e);
            break;
        case SocketAsyncOperation.Send:
            ProcessSend(e);
            break;
        default: throw new Exception("Invalid operation completed"); }
    }
```

```
static void SocketEventArgs_Completed(object sender,  
SocketEventArgs e) {  
    switch (e.LastOperation)  
    {  
        case SocketAsyncOperation.Connect:  
            ProcessConnect(e);  
            break;  
        case SocketAsyncOperation.Receive:  
            ProcessReceive(e);  
            break;  
        case SocketAsyncOperation.Send:  
            ProcessSend(e);  
            break;  
        default: throw new Exception("Invalid operation completed"); }  
    }
```

```
static void SocketEventArgs_Completed(object sender,  
SocketEventArgs e) {  
    switch (e.LastOperation)  
    {  
        case SocketAsyncOperation.Connect:  
            ProcessConnect(e);  
            break;  
        case SocketAsyncOperation.Receive:  
            ProcessReceive(e);  
            break;  
        case SocketAsyncOperation.Send:  
            ProcessSend(e);  
            break;  
        default: throw new Exception("Invalid operation completed"); }  
    }
```



```
private static void ProcessConnect(SocketAsyncEventArgs e)
{
    ...
    byte[] buffer = Encoding.UTF8.GetBytes("Hello World");
    e.SetBuffer(buffer, 0, buffer.Length);
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.SendAsync(e);
    if (!willRaiseEvent)
    {
        ProcessSend(e);
    }
    ...
}
```

```
private static void ProcessConnect(SocketAsyncEventArgs e)
{
    ...
    byte[] buffer = Encoding.UTF8.GetBytes("Hello World");
    e.SetBuffer(buffer, 0, buffer.Length);
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.SendAsync(e);
    if (!willRaiseEvent)
    {
        ProcessSend(e);
    }
    ...
}
```

```
private static void ProcessConnect(SocketAsyncEventArgs e)
{
    ...
    byte[] buffer = Encoding.UTF8.GetBytes("Hello World");
    e.SetBuffer(buffer, 0, buffer.Length);
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.SendAsync(e);
    if (!willRaiseEvent)
    {
        ProcessSend(e);
    }
    ...
}
```

```
private static void ProcessConnect(SocketAsyncEventArgs e)
{
    ...
    byte[] buffer = Encoding.UTF8.GetBytes("Hello World");
    e.SetBuffer(buffer, 0, buffer.Length);
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.SendAsync(e);
    if (!willRaiseEvent)
    {
        ProcessSend(e);
    }
    ...
}
```

```
private static void ProcessConnect(SocketAsyncEventArgs e)
{
    ...
    byte[] buffer = Encoding.UTF8.GetBytes("Hello World");
    e.SetBuffer(buffer, 0, buffer.Length);
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.SendAsync(e);
    if (!willRaiseEvent)
    {
        ProcessSend(e);
    }
    ...
}
```

```
private static void ProcessConnect(SocketAsyncEventArgs e)
{
    ...
    byte[] buffer = Encoding.UTF8.GetBytes("Hello World");
    e.SetBuffer(buffer, 0, buffer.Length);
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.SendAsync(e);
    if (!willRaiseEvent)
    {
        ProcessSend(e);
    }
    ...
}
```

```
static void SocketEventArgs_Completed(object sender,  
SocketEventArgs e) {  
    switch (e.LastOperation)  
    {  
        case SocketAsyncOperation.Connect:  
            ProcessConnect(e);  
            break;  
        case SocketAsyncOperation.Receive:  
            ProcessReceive(e);  
            break;  
        case SocketAsyncOperation.Send:  
            ProcessSend(e);  
            break;  
        default: throw new Exception("Invalid operation completed"); }  
    }
```

```
private static void ProcessSend(SocketAsyncEventArgs e)
{
    ...
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.ReceiveAsync(e);
    if (!willRaiseEvent)
    {
        ProcessReceive(e);
    }
    ...
}
```



```
private static void ProcessSend(SocketAsyncEventArgs e)
{
    ...
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.ReceiveAsync(e);
    if (!willRaiseEvent)
    {
        ProcessReceive(e);
    }
    ...
}
```

```
private static void ProcessSend(SocketAsyncEventArgs e)
{
    ...
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.ReceiveAsync(e);
    if (!willRaiseEvent)
    {
        ProcessReceive(e);
    }
    ...
}
```

```
private static void ProcessSend(SocketAsyncEventArgs e)
{
    ...
    Socket sock = e.UserToken as Socket;
    bool willRaiseEvent = sock.ReceiveAsync(e);
    if (!willRaiseEvent)
    {
        ProcessReceive(e);
    }
    ...
}
```

```
static void SocketEventArgs_Completed(object sender,
SocketEventArgs e) {
    switch (e.LastOperation)
    {
        case SocketAsyncOperation.Connect:
            ProcessConnect(e);
            break;
        case SocketAsyncOperation.Receive:
            ProcessReceive(e);
            break;
        case SocketAsyncOperation.Send:
            ProcessSend(e);
            break;
        default: throw new Exception("Invalid operation completed"); }
    }
```

```
private static void ProcessReceive(SocketAsyncEventArgs e)
{
    ...
    Socket sock = e.UserToken as Socket;
    sock.Shutdown(SocketShutdown.Send);
    sock.Close();
    ...
}
```

```
private static void ProcessReceive(SocketAsyncEventArgs e)
{
    ...
    Socket sock = e.UserToken as Socket;
    sock.Shutdown(SocketShutdown.Send);
    sock.Close();
    ...
}
```

```
private static void ProcessReceive(SocketAsyncEventArgs e)
{
    ...
    Socket sock = e.UserToken as Socket;
    sock.Shutdown(SocketShutdown.Send);
    sock.Close();
    ...
}
```

```
private static void ProcessReceive(SocketAsyncEventArgs e)
{
    ...
    Socket sock = e.UserToken as Socket;
    sock.Shutdown(SocketShutdown.Send);
    sock.Close();
    ...
}
```


Этапы выполнения асинхронной операции

1. Выделите `SocketAsyncEventArgs`
2. Задайте свойства объекта `SocketAsyncEventArgs`,
3. Вызовите соответствующий метод сокета.
4. Функции обратного вызова сделайте запрос состояния завершения и результатов операций.
5. Повторно используйте объект `SocketAsyncEventArgs` для другой операции.

СПАСИБО ЗА ВНИМАНИЕ!