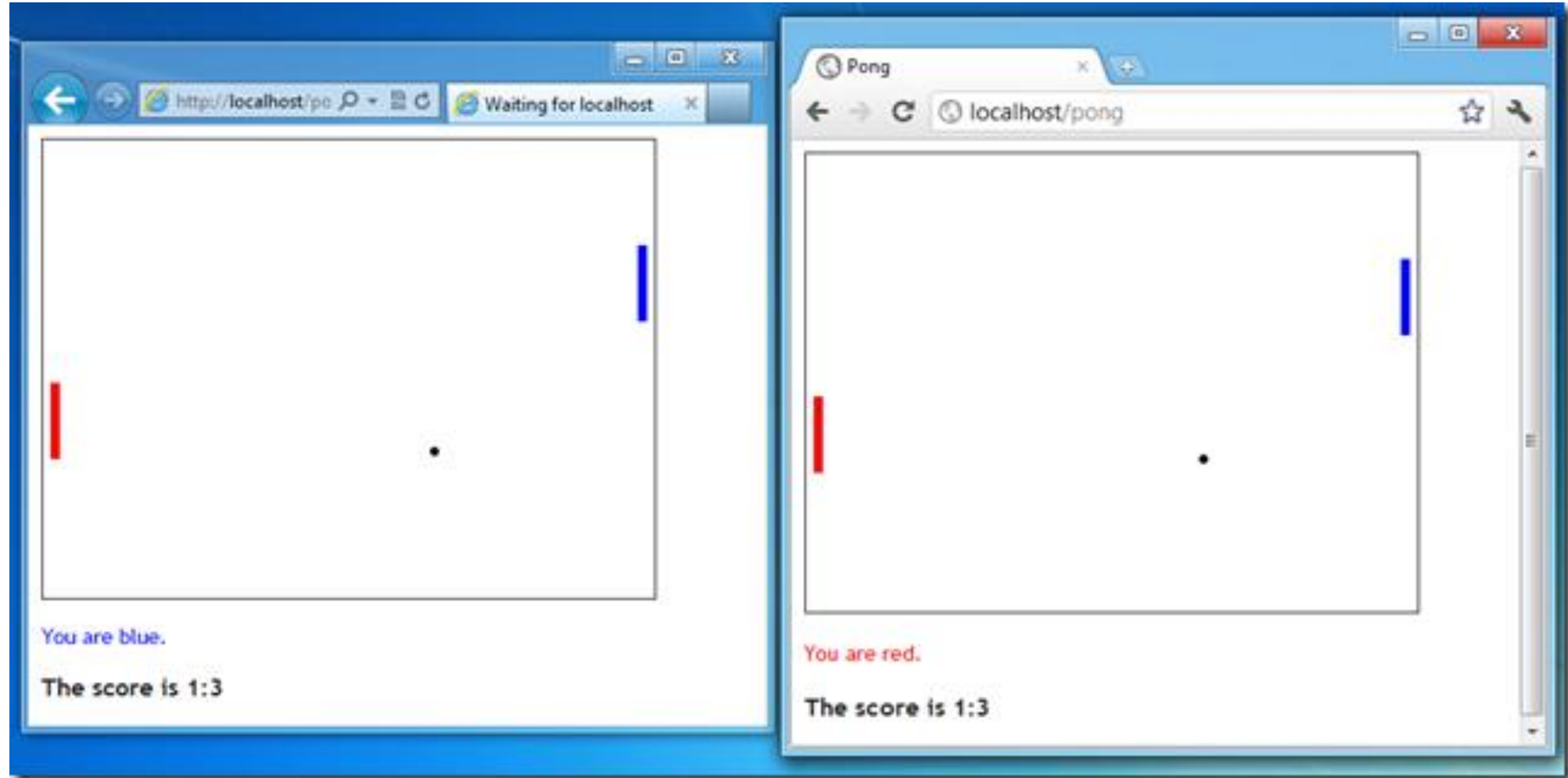


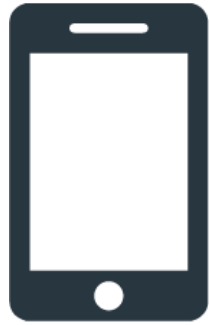
ASP.NET Core SignalR

WebSocket RPC Framework

2018. 09. 19
최흥배

WebSocket ?

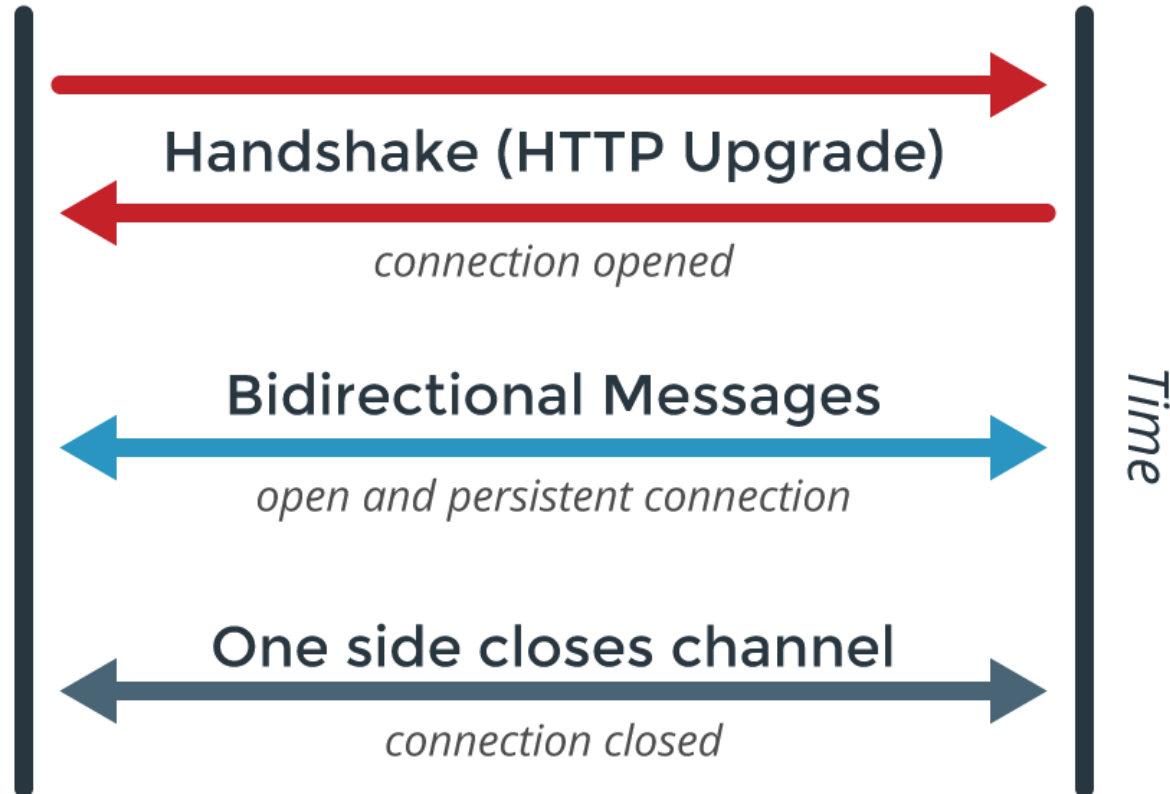




Client

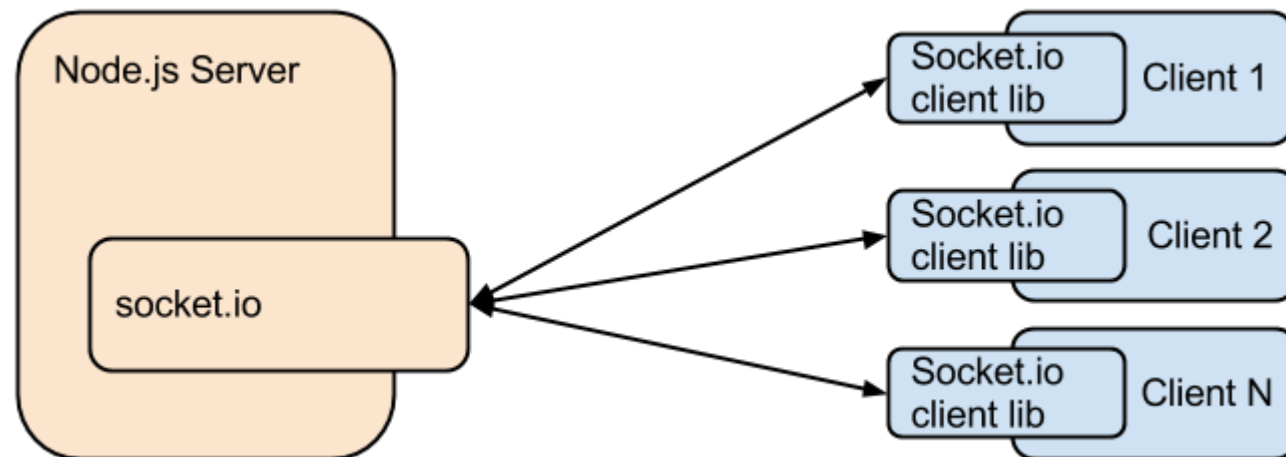


Server





<https://socket.io/>



[WebSocket과 Socket.io](#)

AWS 서비스 내부 통신에 많이 사용되고 있음

AWS Services

Deployment & Management

Application Services



Amazon
SQS



Amazon
ElasticTranscoder



Amazon
SES



Amazon
AppStream



Amazon
CloudSearch

Mobile Services



Amazon
Cognito



Amazon
Mobile Analytics



Amazon
SNS

Enterprise Applications



Amazon
WorkDocs



Amazon
WorkSpaces



Amazon
WorkMail

Application Services

Administration & Security



AWS
DirectoryService



AWS
IAM



AWS
Trusted Advisor



AWS
Config



AWS
CloudTrail



Amazon
CloudWatch

Deployment & Management



Amazon
CloudFormation



AWS
OpsWorks



AWS
CodeDeploy

Analytics



Amazon
Kinesis



AWS
Data Pipeline



Amazon
EMR

Foundation Services

Compute



Amazon
EC2



AWS
Lambda

Storage & Content Delivery



Amazon
CloudFront



Amazon
Glacier



AWS
Storage Gateway



Amazon
Content Delivery

Database



Amazon
DynamoDB



Amazon
RDS



Amazon
Redshift



Amazon
Elastic Cache

Networking



Amazon
Route 53



Amazon
VPC



AWS
Direct Connect

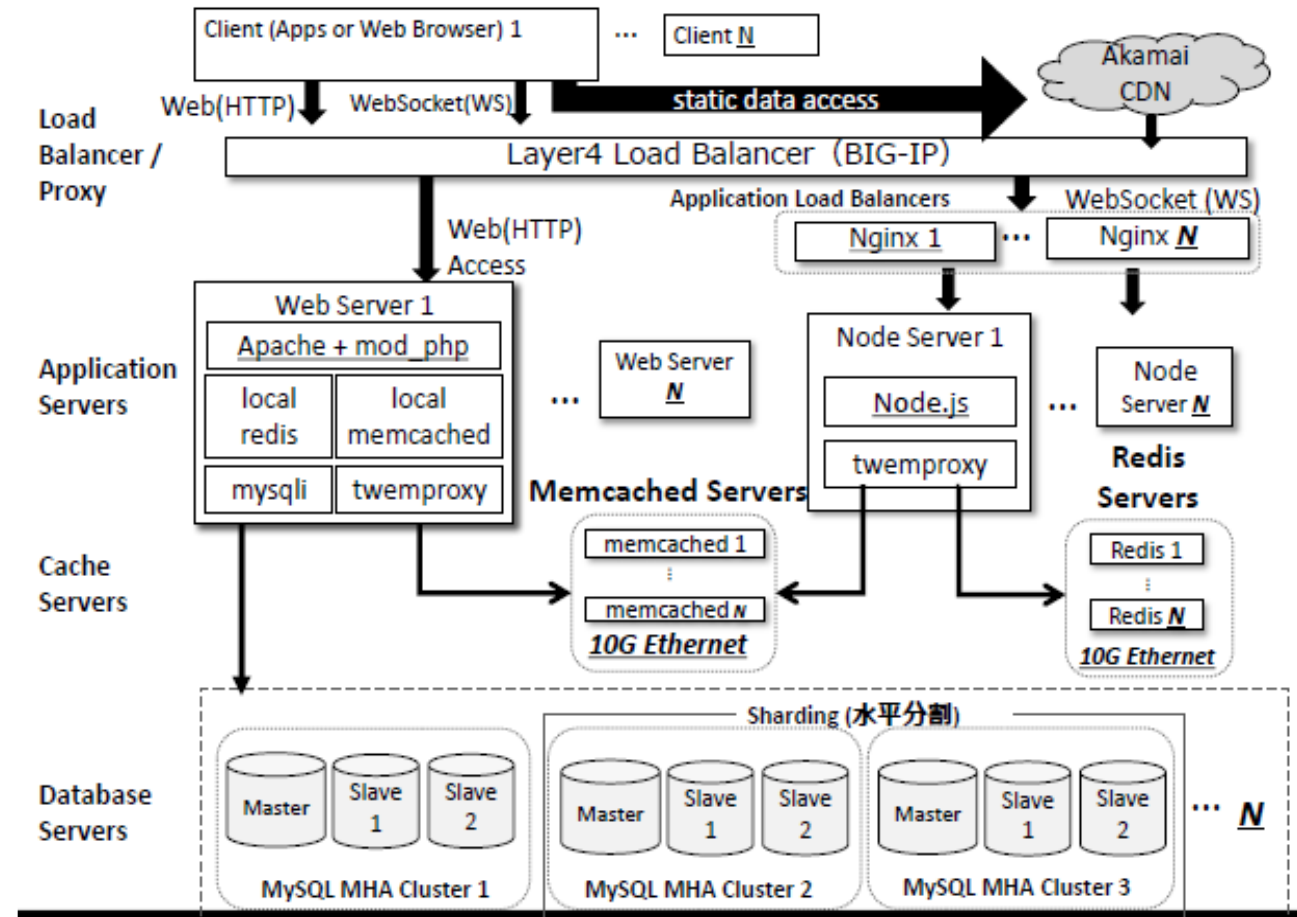
해외의 모바일 플랫폼 온라인 게임 중 WebSocket을 사용하는 경우가 많음.
해외는 한국과 달리 실시간 통신 게임 서버를 개발한 경험이 적음.

WebSocket을 사용하면 **빠르고 쉽게, 실시간 통신**을 할 수 있기 때문.



공격 버튼이나 어빌리티 버튼을 누를 때 마다 리퀘스트(약 3초에 1회)

30인 동시에 싸우는 멀티 배틀



ASP.NET

Core
2.0



localhost:5000

Live Feed About Source code Github Twitter Facebook

Team	Score
Team 1	0
Team 2	0
Basketball	Subscribe to feed
Update	
Match started	

Team	Score
Team 3	0
Team 4	0
Basketball	Subscribe to feed
Update	
Match started	

ID	Match
1	Team 1 vs Team 2
2	Team 3 vs Team 4
b500406f-1f10-44be-91f5-b6c23a364189	
Message	

Developed by Chris S. for chsakell's Blog

Live Feed About Source code Github Twitter Facebook

Team	Score
Team 1	0
Team 2	0
Basketball	Subscribe to feed
Update	
Match started	

Team	Score
Team 3	0
Team 4	0
Basketball	Subscribe to feed
Update	
Match started	

ID	Match
1	Team 1 vs Team 2
2	Team 3 vs Team 4
a7980a17-df2b-474e-b2cf-741dee69f179	
Message	

Developed by Chris S. for chsakell's Blog

<https://chsakell.com/2016/10/10/real-time-applications-using-asp-net-core-signalr-angular/>

SignalR

ASP.NET Core가 나오지 전에 이미 .NET Framework에 있었음.

I 자습서: SignalR 시작하기 (C#)

등록일시: 2013-03-07 12:03, 수정일시: 2014-01-05 23:22

조회수: 3,934

이 문서는 **ASP.NET SignalR** 기술을 널리 알리고자 하는 개인적인 취지로 제공되는 번역문서입니다. 이 문서에 대한 모든 저작권은 마이크로소프트에 있으며 요청이 있을 경우 언제라도 게시가 중단될 수 있습니다. 번역 내용에 오역이 존재할 수 있고 주석은 번역자 개인의 의견일 뿐이며 마이크로소프트는 이에 관한 어떠한 보장도 하지 않습니다. 번역이 완료된 이후에도 대상 제품 및 기술이 개선되거나 변경됨에 따라 원문의 내용도 변경되거나 보완되었을 수 있으므로 주의하시기 바랍니다.

- 본 번역문서의 원문은 [Tutorial: Getting Started with SignalR \(C#\)](#) www.asp.net 입니다.
- 본 번역문서는 [ASP.NET SignalR 시작하기 \(C#\)](#) www.taeyo.net 에서도 함께 제공됩니다.

SignalR 소개

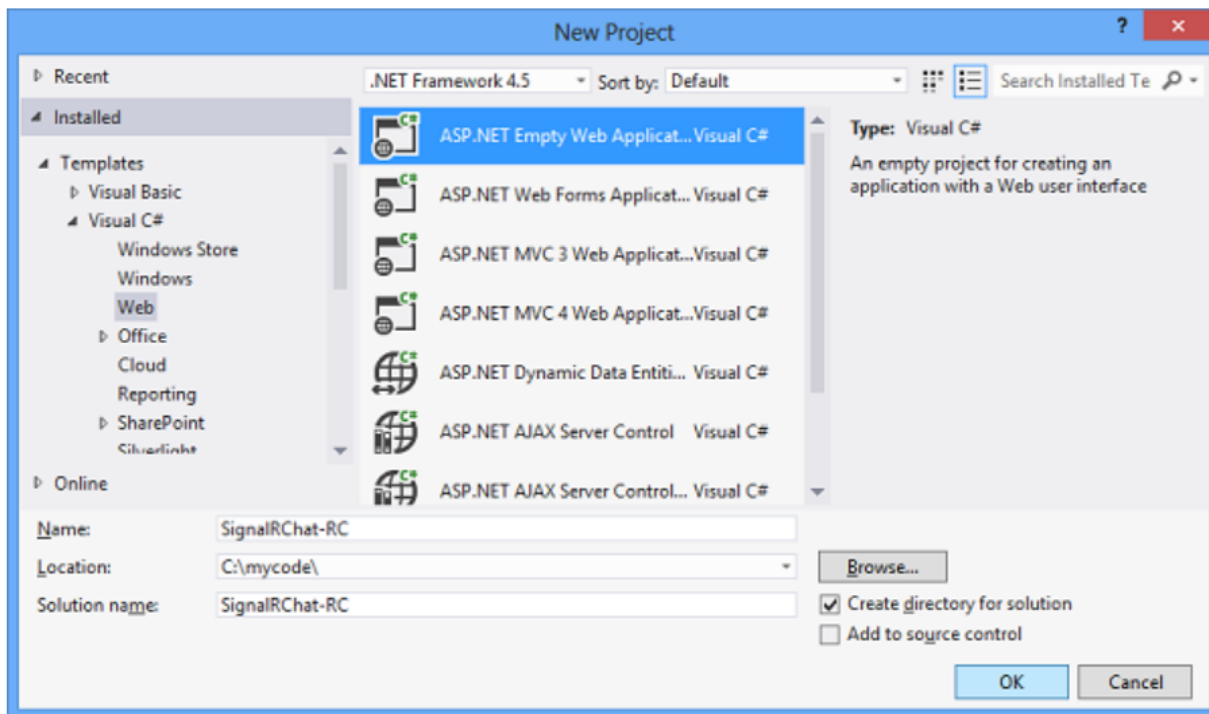
SignalR은 즉각적인 사용자 상호작용이나 실시간 데이터 갱신이 필요한 웹 응용 프로그램 구축을 지원하기 위한 오픈 소스 .NET 라이브러리입니다. 가령, 소셜 응용 프로그램이나 다중 사용자 게임, 업무 협업, 그리고 뉴스나 날씨 또는 금융 갱신 응용 프로그램 등이 이에 해당됩니다. 이런 응용 프로그램들을 종종 실시간 응용 프로그램이라고 부릅니다.

SignalR은 실시간 응용 프로그램의 구축 과정을 단순화시켜 줍니다. ASP.NET 서버 라이브러리와 자바스크립트 클라이언트 라이브러리의 조합을 통해서, 클라이언트와 서버 간의 연결 관리 작업이나 갱신된 콘텐츠를 클라이언트에 푸시하는 작업을 손쉽게 만들어 줍니다. 실시간 기능을 구현하기 위해서 기존 ASP.NET 응용 프로그램에 SignalR 라이브러리를 추가할 수도 있습니다.

프로젝트 설정하기

이번 절에서는 새로운 ASP.NET 웹 응용 프로그램을 생성하고, SignalR 라이브러리를 추가하고, 챗 응용 프로그램을 작성하는 방법을 살펴보겠습니다. Visual Studio 2010 SP1 또는 2012와 .NET 프레임워크 4.0 또는 4.5를 사용합니다. 본문에서는 Visual Studio 2012와 ASP.NET 4.5를 사용하겠습니다.

1. Visual Studio에서 ASP.NET 빈 웹 응용 프로그램(ASP.NET Empty Web Application)을 생성합니다.



2. 도구(Tools) | 라이브러리 패키지 관리자(Library Package Manager) | 패키지 관리자 콘솔(Package Manager Console) 메뉴를 열고 프롬프트에 다음의 명령을 입력합니다.

```
install-package Microsoft.AspNet.SignalR -pre
```

ASP.NET Core SignalR

- .NET Core 버전으로 리뉴얼. 이전 버전과 호환 되지 않음
- 성능 향상
- routing
- Hub 파라미터 binding
- JSON 및 MessagePack 프로토콜
- Controller 에서 push 통지
- 그룹과 background 서비스에서의 push 통지
- 타입이 있는 Hub
- Hub 메소드에서의 Stream
- 지원 플랫폼: .NET Framework 4.6.1 이상, .NET Core 2.1 이상

<> Code

! Issues 166

🔗 Pull requests 5

📁 Projects 0

📖 Wiki

📊 Insights

Incredibly simple real-time web for ASP.NET Core

aspnet-product

📶 1,674 commits

🌿 54 branches

📦 14 releases

👤 54 contributors

Branch: release/2.2 ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾



aspnetci Update dependencies.props ...

Latest commit caa5bfe 6 hours ago

📁 .github	Update issue template for security issues (#2693)	a month ago
📁 .vscode	Add VSCode debug configuration for Java (#2916)	3 days ago
📁 .vsts-pipelines/builds	Revert "Merge branch 'master' into release/2.2"	a month ago
📁 benchmarkapps	Revert "Merge branch 'master' into release/2.2"	a month ago
📁 benchmarks/Microsoft.AspNetCore...	Revert "Merge branch 'master' into release/2.2"	a month ago
📁 build	Update dependencies.props	6 hours ago
📁 clients	Spotless for the Java Client (#2924)	2 days ago
📁 docs	change to use Karma for Functional Tests (#2450)	3 months ago
📁 samples	Flow a cancellation token in to AsChannelReader (#2491)	3 months ago

<https://github.com/aspnet/SignalR>

ASP.NET Core SignalR 소개

📅 2018. 04. 25. • ⌚ 읽는 데 3분 • 참가자 🧑🧑🧑

SignalR이란?

ASP.NET Core SignalR은 실시간 웹 기능을 앱에 추가 간소화 하는 오픈 소스 라이브러리입니다. 실시간 웹 기능 클라이언트로 푸시 콘텐츠를 서버 쪽 코드를 즉시 있습니다.

SignalR에 대 한 것이 좋습니다.

- 서버에서 자주 업데이트가 필요한 앱입니다. 게임, 소셜 네트워크, 투표, 경매, 지도 및 GPS 앱을 예로 들 수 있습니다.
- 대시보드 및 모니터링 앱입니다. 예를 들어 회사 대시보드, 즉석 판매 업데이트 또는 여행 경고가.
- 공동 작업 앱입니다. 화이트 보드 앱 및 팀 회의 소프트웨어가 공동 작업 앱의 예입니다.
- 알림이 필요한 앱입니다. 소셜 네트워크, 이메일, 채팅, 게임, 여행 경고 및 다른 많은 앱 알림을 사용합니다.

서버에서 클라이언트를 만들기 위한 API를 제공 하는 SignalR [원격 프로시저 호출\(RPC\)](#)합니다. Rpc 서버 쪽.NET Core 코드에서 클라이언트에서 JavaScript 함수를 호출합니다.

<https://docs.microsoft.com/ko-kr/aspnet/core/signalr/introduction?view=aspnetcore-2.1>

ASP.NET core SignalR의 기능 중 일부는 다음과 같습니다.

- 연결 관리를 자동으로 처리합니다.
- 동시에 연결 된 모든 클라이언트에 메시지를 보냅니다. 예를 들어 대화방입니다.
- 특정 클라이언트 또는 클라이언트 그룹에 메시지를 보냅니다.
- 증가 트래픽을 처리 하도록 확장 합니다.

원본에서 호스트 되는 [GitHub의 SignalR 리포지토리](#)합니다.

전송

SignalR 실시간 통신을 처리 하기 위한 몇 가지 기법을 지원 합니다.

- [WebSockets](#)
- 서버에서 전송 이벤트
- 긴 폴링

SignalR에는 자동으로 서버와 클라이언트의 기능 내에 있는 최상의 전송 방법을 선택 합니다.

허브

사용 하 여 SignalR *hubs* 클라이언트와 서버 간 통신.

허브는 클라이언트와 서버에서 서로 다른 메서드를 호출할 수 있는 높은 수준의 파이프라인 클라이언트가 서버의 그 반

하는 방법에 강력한 형식의 매개 변수를 전달할 수 있습니다. SignalR 제공 두 가지 기본 제공 허브 **프로토콜: JSON** 및 기반 **이진 프로토콜을** 기반으로 하는 텍스트 프로토콜 **MessagePack** 합니다. 일반적으로 MessagePack JSON에 비해 크기가 작은 메시지를 만듭니다. 이전 버전의 브라우저를 지원 해야 합니다 [XHR 수준 2](#) MessagePack 프로토콜 지원을 제공 합니다.

허브 이름 및 클라이언트 쪽 메서드의 매개 변수를 포함 하는 메시지를 전송 하 여 클라이언트 쪽 코드를 호출 합니다. 메서드 매개 변수로 보낸 개체는 구성된 된 프로토콜을 사용 하 여 역직렬화 됩니다. 클라이언트는 클라이언트 쪽 코드에서 메서드 이름을 찾으려고 합니다. 클라이언트에 일치 항목을 찾으면 메서드를 호출 하 고 deserialize 된 매개 변수 데이터를 전달 합니다.

SignalR Hub Protocol

The SignalR Protocol is a protocol for two-way RPC over any Message-based transport. Either party in the connection may invoke procedures on the other party, and procedures can return zero or more results or an error.

Terms

- Caller - The node that is issuing an `Invocation`, `StreamInvocation`, `CancelInvocation`, `Ping` messages and receiving `Completion`, `StreamItem` and `Ping` messages (a node can be both Caller and Callee for different invocations simultaneously)

<https://github.com/aspnet/SignalR/blob/release/2.1/specs/HubProtocol.md>

ASP.NET Core 2.1 ▾

검색

Microsoft.AspNetCore.SignalR

- > ClientProxyExtensions
- > DefaultHubLifetimeManager<THub>
- > DefaultUserIdProvider
- > DynamicHub
- > DynamicHubClients
- > GetHttpContextExtensions
- > Hub
- > Hub<T>
- > HubCallerContext
- > HubClientsExtensions
- > HubConnectionContext
- > HubConnectionHandler<THub>
- > HubConnectionStore
- > HubConnectionStore.Enumerator

Microsoft.AspNetCore.SignalR Namespace

Classes

ClientProxyExtensions	Extension methods for IClientProxy .
DefaultHubLifetimeManager<THub>	A default in-memory lifetime manager abstraction for Hub instances.
DefaultUserIdProvider	The default provider for getting the user ID from a connection. This provider gets the user ID from the connection's User name identifier claim.
DynamicHub	A base class for SignalR hubs that use <code>dynamic</code> to represent client invocations.
DynamicHubClients	A class that provides <code>dynamic</code> access to connections, including the one that sent the current invocation.
GetHttpContextExtensions	Extension methods for accessing HttpContext from a hub context.
Hub	A base class for a SignalR hub.

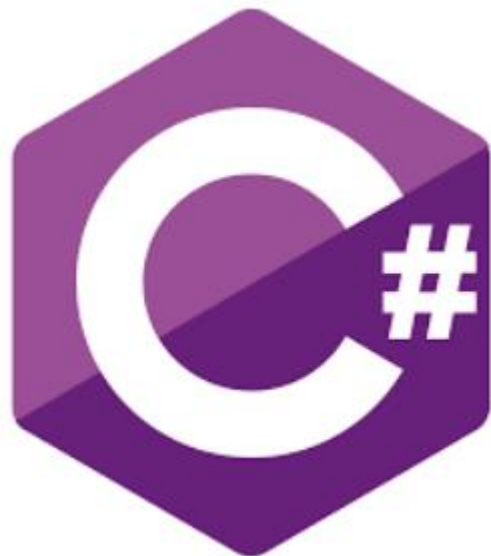
<https://docs.microsoft.com/ko-kr/dotnet/api/microsoft.aspnetcore.signalr?view=aspnetcore-2.1>

클라이언트 지원

2.2 버전에 예정



JavaScript



2.2 버전에 예정



ASP.NET Core 2.2 Roadmap #307

🔔 Open

glennc opened this issue on 26 Jun · 0 comments



glennc commented on 26 Jun • edited ▾

Member

...

We have grouped the ASP.NET Core 2.2 release into a few themes:

- APIs & Services
- Server improvements
- SignalR

Rough Schedule

SignalR

Add Java & C++ clients

Allow customers using Java or C++ to connect to SignalR servers, and the [Azure SignalR Service](#).

<https://github.com/aspnet/Announcements/issues/307>

<https://blogs.msdn.microsoft.com/webdev/2018/08/23/asp-net-core-2-2-0-preview1-signalr-java-client/>

ASP.NET Blog

.NET web development and tools at Microsoft

ASP.NET Core 2.2.0-preview1: SignalR Java Client



August 23, 2018 by Sourabh Shirhatti [MSFT] // 0 Comments

 Share 18

 0

 0

This post was authored by [Mikael Mengistu](#).

In ASP.NET Core 2.2 we are introducing a Java Client for SignalR. The first preview of this new client is available now. This client supports connecting to an ASP.NET Core SignalR Server from Java code, including Android apps.

The API for the Java client is very similar to that of the already existing .NET and JavaScript clients but there are some important differences to note.

The HubConnection is initialized the same way, with the `HubConnectionBuilder` type.

<https://blogs.msdn.microsoft.com/webdev/2018/08/23/asp-net-core-2-2-0-preview1-signalr-java-client/>



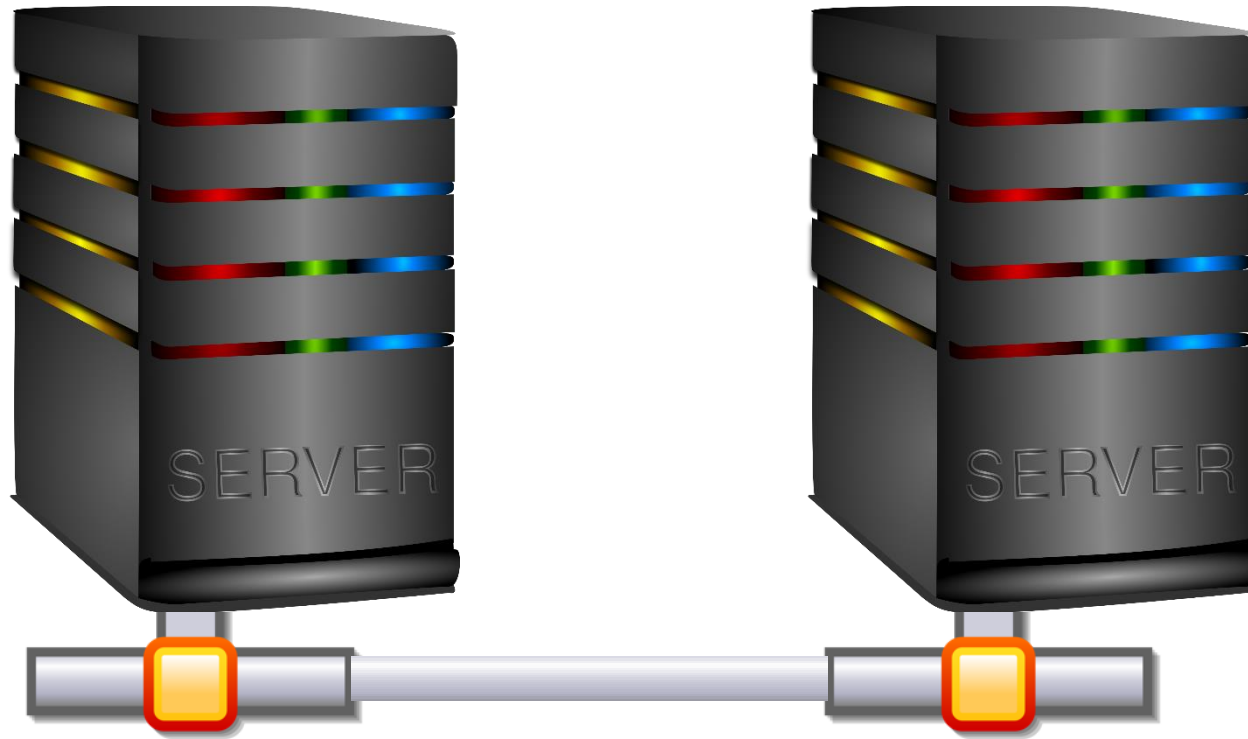
최신 버전의 Unity는 .NET Core는 지원하지 않고, .NET Framework 4.x 버전만 지원.

Unity 혹은 SignalR 팀에서 지원하지 않으면 Unity에서 SignalR C# 클라이언트를 사용하지 못함.

참고: [2018.1.0b10: guide for net core 2.0 dll integration](#)

올해 말에 C++ 지원 클라이언트 라이브러리가 나오면 이것을 Unity에서 사용하도록 **플러그인**을 만들면 사용 할 수 있을 듯

MS 혹은 개발사에서 유니티용 플러그인을 만들어서 애셋 스토어에 등록될 것으로 예상합니다.
(SignalR이 C#을 사용하는 유니티와 궁합이 좋으니)



서버간 통신에 사용할 수도 있음



HTML5 게임(MMO 등)에서 사용



Azure SignalR Service

미리 보기

편리하게 응용 프로그램에 실시간 웹 기능 추가

무료 체험 시작하기 >

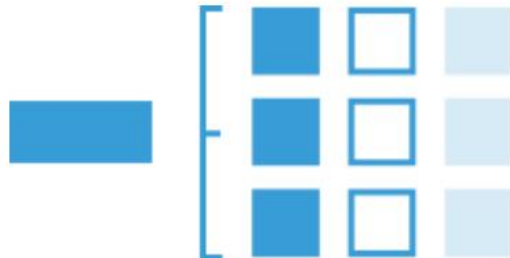
이미 Azure를 사용하고 있습니까? 지금 SignalR Service 사용해 보기 >

Azure SignalR Service 살펴보기: [가격 정보](#) [설명서](#) [샘플 코드](#)

완전히 관리되는 서비스

SignalR Service는 완전히 관리되는 서비스이므로 호스팅, 확장성, 부하 분산 또는 인증에 대해 염려하지 않고 다중 서버 환경에 배포할 수 있습니다.

<https://azure.microsoft.com/ko-kr/services/signalr-service/>



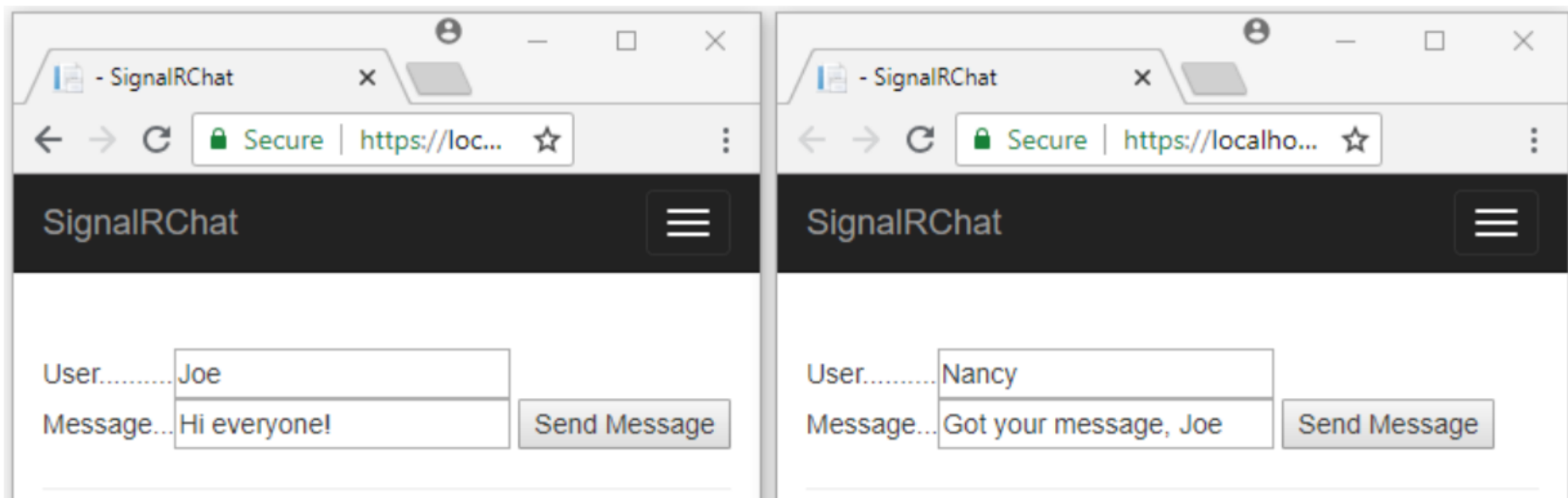
자습서: ASP.NET Core에서 SignalR 시작

📅 2018. 08. 20. • ⌚ 읽는 데 9분 • 참가자 🧑🏻 🇺🇸

이 자습서에서는 SignalR을 사용하여 실시간 앱을 빌드하는 방법에 대한 기본 사항을 설명합니다. 여기에서는 다음과 같은 작업을 수행하는 방법에 대해 배우게 됩니다.

- ✓ ASP.NET Core에서 SignalR을 사용하는 웹앱을 만듭니다.
- ✓ 서버에서 SignalR 허브를 만듭니다.
- ✓ JavaScript 클라이언트에서 SignalR 허브에 연결합니다.
- ✓ 허브를 사용하여 모든 클라이언트에서 연결된 모든 클라이언트에 메시지를 보냅니다.

작동하는 채팅 앱이 만들어집니다.



<https://docs.microsoft.com/ko-kr/aspnet/core/tutorials/signalr?view=aspnetcore-2.1&tabs=visual-studio>

코드 다운로드 <https://github.com/aspnet/Docs>

<> Code

! Issues 8

🔗 Pull requests 1

📁 Projects 0

📖 Wiki

📊 Insights

Samples for ASP.NET Core SignalR

🕒 50 commits

🌿 22 branches

📦 0 releases

👤 8 contributors

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾



mikaelm12 Update stock prices in sample to resemble reality (for now) (#46)

Latest commit b9d6055 16 days ago

📁 AndroidJavaClient	Android Java client sample (#44)	17 days ago
📁 ChatSample	Fix minor whitespace issue	2 months ago
📁 MoveShape	More clean up	4 months ago
📁 StockTickR	Update stock prices in sample to resemble reality (for now) (#46)	16 days ago
📁 WhiteBoard	More clean up	4 months ago
📁 WindowsFormsSample	Removed comment	4 months ago
📁 WindowsUniversal	Add UWP client sample (#35)	2 months ago
📁 Xamarin	update to SignalR 1.0.1 (#38)	a month ago
📄 .gitattributes	Adding StockTickR App	a year ago
📄 .gitignore	Adding StockTickR App	a year ago

<https://github.com/aspnet/SignalR-samples>



SignalR 좀 더 안을 보자...

Using SignalR with Bare Websockets

The process of connecting to SignalR has been simplified to the point where, when using websockets, it is now possible to connect to the server without any client with a single request.

<https://blogs.msdn.microsoft.com/webdev/2017/09/14/announcing-signalr-for-asp-net-core-2-0/>

Simplified Connection Model

In the existing version of SignalR the client would try starting a connection to the server, and if it failed it would try using a different transport. The client would fail starting the connection when it could not connect to the server with any of the available transports. This feature is no longer supported with the new SignalR.

Another functionality that is no longer supported is automatic reconnects. Previously SignalR would try to reconnect to the server if the connection was dropped. Now, if the client is disconnected the user must explicitly start a new connection if they want to reconnect. Note, that it was required even before – the client would stop its reconnect attempts if it could not reconnect successfully within the reconnect timeout. One more reason to remove automatic reconnects was a very high cost of storing messages sent to clients. The server would by default remember the last 1000 messages sent to a client so that it could replay messages the client missed when it was offline. Since each connection had its own buffer the memory footprint of storing these messages was very high.

<https://blogs.msdn.microsoft.com/webdev/2017/09/14/announcing-signalr-for-asp-net-core-2-0/>

- 기존에는 연결할 수 있는 모든 방법을 자동으로 시도했지만 이젠 지원하지 않음.
- 자동 재 접속 지원하지 않음. 이유는 메시지 누락을 막기 위해 메시지를 버퍼에 최대 1000개까지 저장하는데 많은 메모리 공간 소모

Single Hub per Connection 단일 허브만 지원

The new version of SignalR does not support having more than one Hub per connection. This results in a simplified client API, and makes it easier to apply Authentication policies and other Middleware to Hub connections. In addition subscribing to hub methods before the connection starts is no longer required.

<https://blogs.msdn.microsoft.com/webdev/2017/09/14/announcing-signalr-for-asp-net-core-2-0/>

What's under Hubs?

Original SignalR

Hub

HubDispatcher

PersistentConnectionMiddleware

OWIN

System.Web HttpHandler

ASP.NET Core SignalR

Hub

HubConnectionHandler

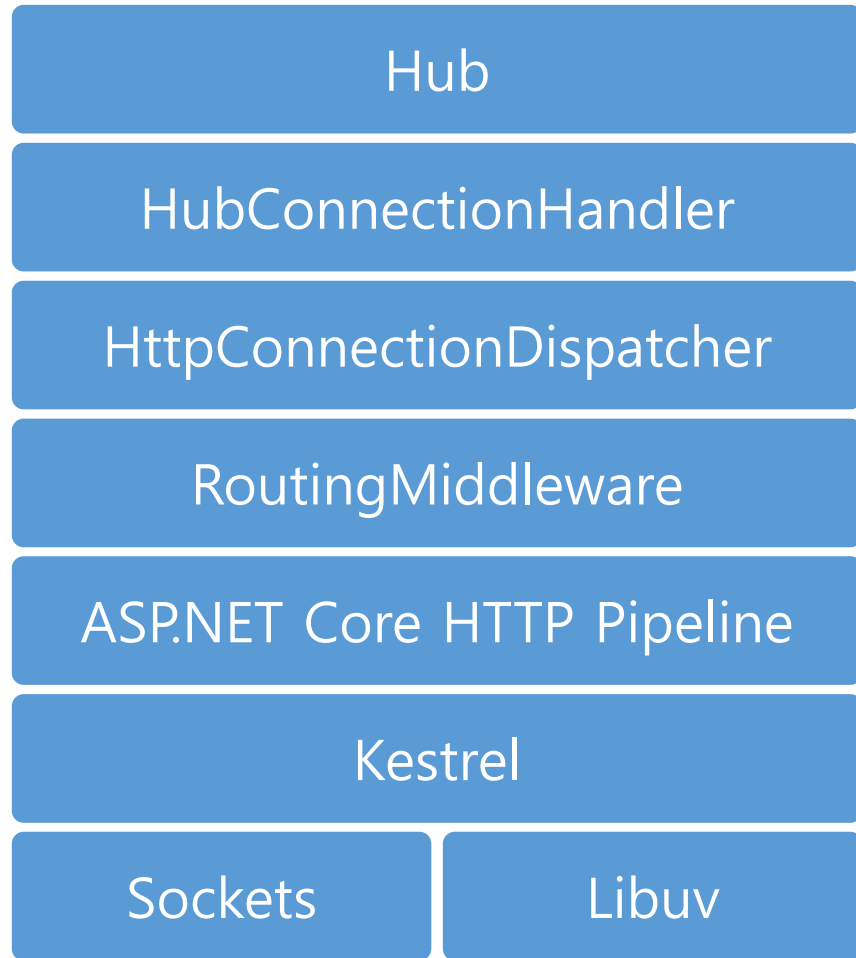
HttpConnectionDispatcher

RoutingMiddleware

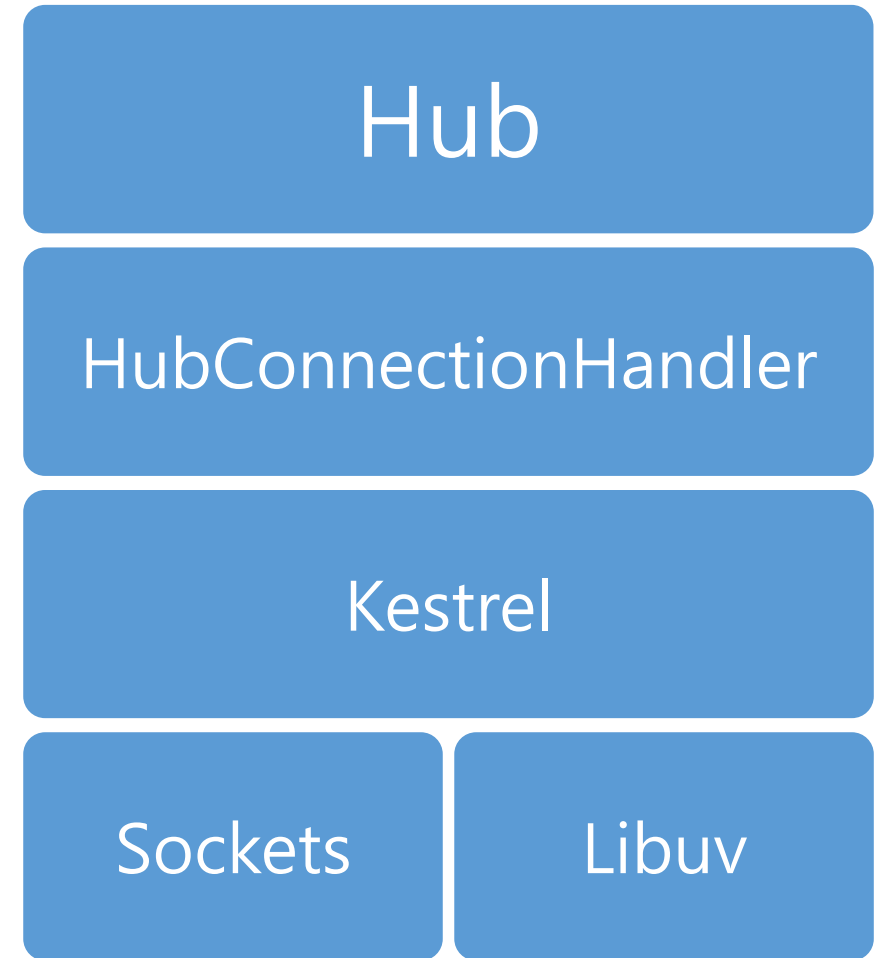
ASP.NET Core HTTP Pipeline

Server & transport agnostic

Over HTTP



Over TCP/IP



Hub 클래스

SignalR의 핵심 중의 핵심이 Hub 클래스이다.

여기를 통해 접속, 접속해제, Rpc 함수 호출 등의 이벤트가 발생한다.

SignalR 응용 프로그램의 허브 클래스는 기본 Hub 클래스를 상속한다.

이 기본 클래스의 유일한 목적은 개발자가 같은 형태의 코드를 반복해서 작성하지 않도록 하는 것이다.

기본 클래스는 인프라의 일부만을 제공하지만, 미리 정의된 동작은 제공하지 않는다.

member	설명
Clients	허브가 현재 관리하는 클라이언트 목록을 공개하는 속성.
Context	호출자의 현재 상황을 공개하는 속성. 사용 가능하다면 연결 ID 나 사용자의 요구 등의 정보를 포함한다.
Groups	클라이언트의 다양한 하위 집합을 공개하는 속성. 클라이언트 전체 목록의 그룹으로서 프로그램에서 정의할 수 있다. 그룹은 일반적으로 특정 메시지를 선택한 대상에 브로드 캐스트 하는 수단으로 만든다.
OnConnectedAsync	새로운 클라이언트가 허브에 연결할 때마다 호출되는 가상 메소드.
OnDisconnectedAsync	새로운 클라이언트가 허브에서 끊어질 될 때마다 호출되는 가상 메소드.


```
public class ChatHub : Hub
{
    참조 0개 | 0 예외
    public void Send(string name, string message)
    {
        // Call the broadcastMessage method to update clients.
        Clients.All.SendAsync("broadcastMessage", name, message);
    }
}
```

ASP.NET Core의 Controller에서 Hub 조작

```
public class TaskController : Controller
{
    private readonly IHubContext<ProgressHub> _progressHubContext;

    참조 0개 | 0 예외
    public TaskController(IHubContext<ProgressHub> progressHubContext)
    {
        _progressHubContext = progressHubContext;
    }

    //
    참조 0개 | 0 원 0개 | 0 예외
}
```

예) 특정 API가 호출 되었을 때 실시간으로 접속된 모든 클라이언트에 통보하기.

서버에 특정 이벤트가 발생했을 때 진행 상황을 실시간으로 접속된 모든 클라이언트에 통보하기.

```
public void Lengthy([Bind(Prefix="id")] string connId)
{
    var steps = new Random().Next(3, 20);
    var increase = (int) 100/steps;

    // NOTIFY START
    _progressHubContext.Clients.Client(connId).SendAsync("initProgressBar");
    //_progressHubContext.Clients.All.SendAsync("initProgressBar");
    var total = 0;

    for (var i = 0; i < steps; i++)
    {
        Thread.Sleep(2000);
        total += increase;

        // PROGRESS
        _progressHubContext.Clients.Client(connId).SendAsync("updateProgressBar", total);
        //_progressHubContext.Clients.All.SendAsync("updateProgressBar", total);
    }

    // NOTIFY END
    _progressHubContext.Clients.Client(connId).SendAsync("clearProgressBar");
    //_progressHubContext.Clients.All.SendAsync("clearProgressBar");
}
```

버퍼 관리

- 세션 당 받기/보내기 메시지를 관리하는 버퍼를 가지고 있다.
- 이 버퍼의 최대 크기는 32KB.
- 즉 1 메시지 당 최대 크기는 32KB 이다.
- `HttpConnectionDispatcherOptions` 설정을 통해서 크기를 줄이고 늘릴 수 있다.
- **ApplicationMaxBufferSize** 서버의 클라이언트 버퍼의 최대 바이트 수를 나타낸다. 클라이언트에서 이 제한 보다 큰 메시지를 전송 하려고 하는 경우 연결이 닫힐 수 있다.

- TransportMaxBufferSize 서버에서 클라이언트로 보낼 수 있는 최대 바이트 수를 나타낸다. 이 크기보다 큰 메시지를 서버에서 보내려고 하는 경우(허브 메시드의 반환 값 포함) 예외가 throw 된다.
- 제한 설정을 0으로 하여 한계를 정하지 않을 수 있다. 그러나 이것은 사용을 주의해야 한다. 악의적인 클라이언트가 큰 메시지를 보내는 경우 과도한 메모리 할당으로 동시 연결 수가 줄어 들 수 있다.

Configuration

Configure server options

The following table describes options for configuring SignalR hubs: 허브 구성 옵션

Option	Default Value	Description
HandshakeTimeout	15 seconds	If the client doesn't send an initial handshake message within this time interval, the connection is closed. This is an advanced setting that should only be modified if handshake timeout errors are occurring due to severe network latency. For more detail on the handshake process, see the SignalR Hub Protocol Specification .
KeepAliveInterval	15 seconds	If the server hasn't sent a message within this interval, a ping message is sent automatically to keep the connection open. When changing <code>KeepAliveInterval</code> , change the <code>ServerTimeout / serverTimeoutInMilliseconds</code> setting on the client. The recommended <code>ServerTimeout / serverTimeoutInMilliseconds</code> value is double the <code>KeepAliveInterval</code> value.
SupportedProtocols	All installed protocols	Protocols supported by this hub. By default, all protocols registered on the server are allowed, but protocols can be removed from this list to disable specific protocols for individual hubs.
EnableDetailedErrors	false	If <code>true</code> , detailed exception messages are returned to clients when an exception is thrown in a Hub method. The default is <code>false</code> , as these exception messages can contain sensitive information. true로 하면 예외가 발생한 경우 예외 정보가 클라이언트로 전달. 개발할 때만 true로

<https://docs.microsoft.com/en-us/aspnet/core/signalr/configuration?view=aspnetcore-2.1#configure-server-options>

Option	Default Value	Description
<code>ApplicationMaxBufferSize</code>	32 KB	The maximum number of bytes received from the client that the server buffers. Increasing this value allows the server to receive larger messages, but can negatively impact memory consumption.
<code>AuthorizationData</code>	Data automatically gathered from the <code>Authorize</code> attributes applied to the Hub class.	A list of <code>IAuthorizeData</code> objects used to determine if a client is authorized to connect to the hub.
<code>TransportMaxBufferSize</code>	32 KB	The maximum number of bytes sent by the app that the server buffers. Increasing this value allows the server to send larger messages, but can negatively impact memory consumption.
<code>Transports</code>	All Transports are enabled.	A bitmask of <code>HttpTransportType</code> values that can restrict the transports a client can use to connect.
<code>LongPolling</code>	See below.	Additional options specific to the Long Polling transport.
<code>WebSockets</code>	See below.	Additional options specific to the WebSockets transport.

The Long Polling transport has additional options that can be configured using the `LongPolling` property:

Option	Default Value	Description
<code>PollTimeout</code>	90 seconds	The maximum amount of time the server waits for a message to send to the client before terminating a single poll request. Decreasing this value causes the client to issue new poll requests more frequently.

The WebSocket transport has additional options that can be configured using the `WebSockets` property:

Option	Default Value	Description
<code>CloseTimeout</code>	5 seconds	After the server closes, if the client fails to close within this time interval, the connection is terminated.
<code>SubProtocolSelector</code>	<code>null</code>	A delegate that can be used to set the <code>Sec-WebSocket-Protocol</code> header to a custom value. The delegate receives the values requested by the client as input and is expected to return the desired value.

Streaming

```
public class StreamHub : Hub
{
    public ChannelReader<int> Counter(int count, int delay)
    {
        var channel = Channel.CreateUnbounded<int>();

        // We don't want to await WriteItems, otherwise we'd end up waiting
        // for all the items to be written before returning the channel back to
        // the client.
        _ = WriteItems(channel.Writer, count, delay);

        return channel.Reader;
    }

    private async Task WriteItems(ChannelWriter<int> writer, int count, int delay)
    {
        for (var i = 0; i < count; i++)
        {
            await writer.WriteAsync(i);
            await Task.Delay(delay);
        }

        writer.TryComplete();
    }
}
```

MessagePack 프로토콜 사용하기

클라이언트: JavaScript

```
var protocol = new signalR.protocols.msgpack.MessagePackHubProtocol();
var progressConnection = new signalR.HubConnection(
    "/progressDemo",
    {
        transport : signalR.TransportType.WebSocket,
        protocol : protocol
    }
);
```

서버

```
public void ConfigureServices(IServiceCollection services)
{
    // SignalR already configured. Just add this:
    services.AddMessagePackProtocol();
}
```

```
private static HubConnection _connection;
private async void Form1_Load(object sender, EventArgs e)
{
    _connection = new HubConnectionBuilder()
        .WithUrl("http://localhost:60000/progressdemo")
        .Build();
    await _connection.StartAsync();
}
```

클라이언트 - JSON

```
{
    _connection = new HubConnectionBuilder()
        .WithUrl("http://localhost:60000/progressdemo")
        .WithMessagePackProtocol()
        .Build();
    await _connection.StartAsync();
}
```

클라이언트 - MessagePack

크로스 도메인

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(); //CORS 미들웨어를 사용해야 한다
}
```

Introducing Project Bedrock

New high performance networking abstraction for .NET

Separates application layers from transport layers

High performance transport implementations: System.Net.Sockets, libuv

New transports can come online without rewriting application layers

Middleware abstraction for cross-cutting concerns: TLS, logging, etc.

Extending ASP.NET Core beyond HTTP

Built on the new System.IO.Pipelines API

Client and server

Bedrock layers

Application framework

ConnectionHandler

Middleware

TLS

Logging

Other

Server

Kestrel

Transports


Sockets

libuv

RIO

Magma

Project Bedrock #1980

 Open

davidfowl opened this issue on 4 Aug 2017 · 101 comments



davidfowl commented on 4 Aug 2017 • edited by shirhatti ▾

Member



Project Bedrock

Project bedrock is about further decoupling the components of Kestrel so that we can use it as the foundation for our non-http networking stack.

We want to build on the primitives, patterns and cross cutting concerns that exist today in ASP.NET Core applications. The goal is to enable higher level frameworks (like SignalR or WCF and even ASP.NET Core itself) to build on top of abstractions that don't tie them to a specific connection implementation (OWIN for Connections). As an example, it allows SignalR to run both on top of TCP or Websockets without having to understand what the underlying transport is. We also want to enable building raw low level protocol servers to handle things like MQTT for IOT scenarios.

There are 3 main actors in this server side programming model:

- **Applications/Middleware/Frameworks** - The application code that handles connections and implement protocol parsing logic or other logic that modifies the stream of data (http, TLS as an example)

<https://github.com/aspnet/KestrelHttpServer/issues/1980>

[illegible]




Visual
Studio



Demo: Echo

- Server: .NET Core 2.1
- Client: .NET Framework 4.6

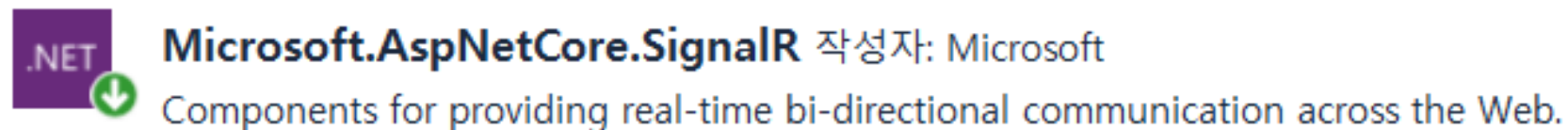
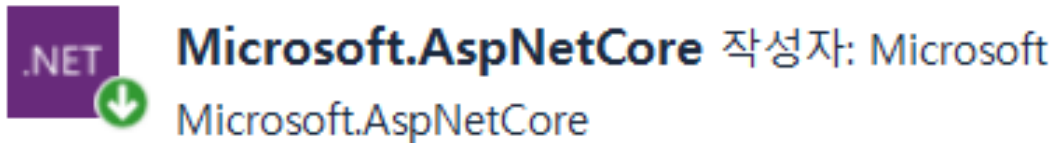
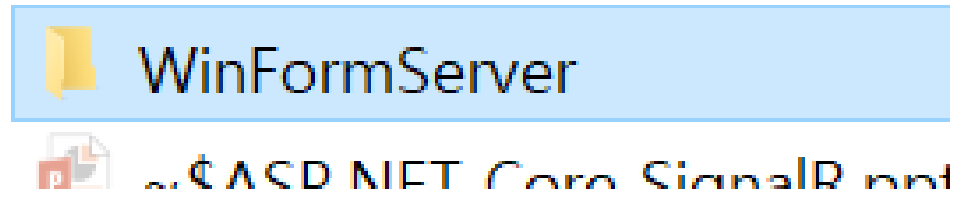
 EchoClient

 EchoServer



Demo: SignalR Server - WinForm

- Server: .NET Framework 4.6
- Client: .NET Framework 4.6



Demo: 서버 이벤트



WinFormServer



~\C#\NET Core SignalR\nt

```
public class EchoHub : Hub
{
    static int Count2 = 0;
    int Count = 0;

    참조 0개
    public override Task OnConnectedAsync(...)

    참조 0개
    public override Task OnDisconnectedAsync(Exception exception)

    참조 0개
    public Task Broadcast(string message)
    {
        var timestamp = DateTime.Now.ToString();
        return this.Clients.All.SendAsync("Receive", message, timestamp);
    }
}
```

Hub 클래스 생성 주기



WinFormServer



ASD NET Core SignalR.net

```
public class EchoHub : Hub
```

```
{
```

```
    static int Count2 = 0;
```

```
    int Count = 0;
```

참조 0개

```
    public override Task OnConnectedAsync()
```

```
    {
```

```
        base.OnConnectedAsync();
```

```
        Interlocked.Increment(ref Count);
```

```
        DevLog.Write($"Client: {Context.ConnectionId}, {Context.User.Identity.Name}, [Total: {Count}]", LOG_LEVEL.INFO);
```

```
        Interlocked.Increment(ref Count2);
```

```
        DevLog.Write($"Client: {Context.ConnectionId}, {Context.User.Identity.Name}, [Total: {Count2}]", LOG_LEVEL.INFO);
```

```
        return Task.CompletedTask;
```

```
    }
```

참조 0개

```
    public override Task OnDisconnectedAsync(Exception exception)
```

```
    {
```

```
        base.OnDisconnectedAsync(exception);
```

```
        Interlocked.Decrement(ref Count);
```

```
        DevLog.Write($"Client: {Context.ConnectionId}, Total: {Count}", LOG_LEVEL.INFO);
```

```
        Interlocked.Decrement(ref Count2);
```

```
        DevLog.Write($"Client: {Context.ConnectionId}, {Context.User.Identity.Name},
```

Port

19000

서버 시작

```
2018-09-13 오후 1:40:39:OnConnectedAsync| Client: wLY3xV9IYSbqvGPNnCDN-g, [Total: 1]
2018-09-13 오후 1:40:39:OnConnectedAsync| Client: wLY3xV9IYSbqvGPNnCDN-g, [Total: 1]
2018-09-13 오후 1:40:44:OnDisconnectedAsync| Client: wLY3xV9IYSbqvGPNnCDN-g, Total: -1
2018-09-13 오후 1:40:44:OnDisconnectedAsync| Client: wLY3xV9IYSbqvGPNnCDN-g, [Total: 0]
```

Demo: Hub 호출은 Multi-Thread ?



WinFormServer



ASD NET Core SignalR not

```
public Task DelayTast1()
{
    Thread.Sleep(5000);

    var timestamp = DateTime.Now.ToString();
    return this.Clients.Caller.SendAsync("Receiv
}
```

참조 0개

```
public async Task DelayTast2()
{
    await Task.Delay(5000);

    var timestamp = DateTime.Now.ToString();
    await this.Clients.Caller.SendAsync("Receive
}
```


클라이언트 A가 요청한 것이 많은 시간이 걸리는 작업이 이라면 다음 요청을 서버로 보내도 앞에 요청한 것을 다 처리하기 전까지는 처리하지 않음.

클라이언트 A가 요청한 것이 많은 시간이 걸리는 작업이라도, 클라이언트가 B가 보낸 요청은 지연이 즉시 처리 해준다.

각 클라이언트 별로 요청을 순차적으로 처리해 준다.

Demo: 서버에서 클라이언트 짜르기

서버에서 접속 중인 클라이언트를 강제로 끊고 싶은 경우 **Abort()** 함수를 호출한다.
현재 WebSocket 사양에는 서버가 강제로 클라이언트를 끊는 방법은 지원하지 않음.

 WinFormServer

 ~\ASD.NET Core SignalR\nt

```
public Task SelfKickOff()
{
    DevLog.Write($"SelfKickOff, Client: {Context.ConnectionId}", LOG_LEVEL.INFO);

    Context.Abort();
    return Task.CompletedTask;
}
```


Demo: 클라이언트의 콜백 메소드

📁 CallbackTestClient

📁 CallbackTestServer

```
public Task Broadcast(string message)
{
    var timestamp = DateTime.Now.ToString();
    //return this.Clients.All.SendAsync("Receive", message, timestamp);
    return this.Clients.All.EchoReceive(Context.ConnectionId, message, timestamp);
}
```

Demo: Chat Server

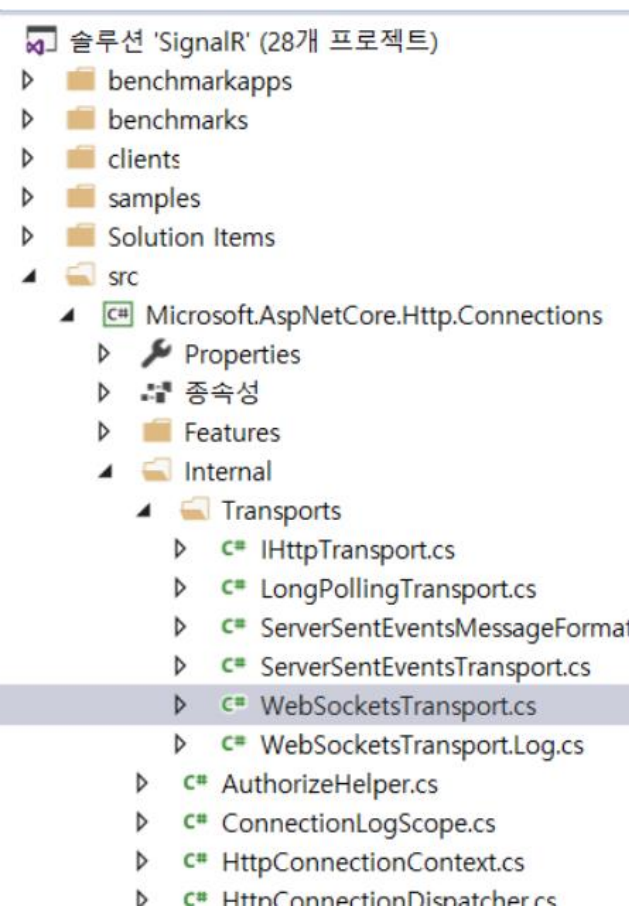


ChatClient



ChatServer





```
public async Task ProcessRequestAsync(HttpContext context, CancellationToken token)
{
    Debug.Assert(context.WebSockets.IsWebSocketRequest, "Not a websocket request");

    var subProtocol = _options.SubProtocolSelector?.Invoke(context.WebSockets.WebSocketRequestedProtocols);

    using (var ws = await context.WebSockets.AcceptWebSocketAsync(subProtocol))
    {
        Log.SocketOpened(_logger, subProtocol);

        try
        {
            await ProcessSocketAsync(ws);
        }
        finally
        {
            Log.SocketClosed(_logger);
        }
    }
}
```

참조 9개 | 0 예외

```
public async Task ProcessSocketAsync(WebSocket socket)
{
    // Begin sending and receiving. Receiving must be started first because ExecuteAsync enables SendAsync.
    var receiving = StartReceiving(socket);
    var sending = StartSending(socket);

    // Wait for send or receive to complete
    var trigger = await Task.WhenAny(receiving, sending);

    if (trigger == receiving)
    {

```