Mapping SignalR Users to Connections

12/30/2014 8 minutes to read Contributors all

In this article

Software versions used in this topic
Previous versions of this topic
Questions and comments
Introduction
IUserID provider
In-memory storage
Single-user groups
Permanent, external storage

by Tom FitzMacken

Note

This article refers to ASP.NET SignalR. If you're thinking about using SignalR to enable real-time scenarios with Java, Node.js, or in a serverless scenario, take a look at ASP.NET Core SignalR. If you've already used ASP.NET SignalR, take a look at the version differences page to understand the differences in the versions and the improvements in ASP.NET Core SignalR. Finally, if you know you'll be running your real-time apps in Microsoft Azure, take a look at the Azure SignalR Service, as it provides cloud-based scale-out once your apps need it.

This topic shows how to retain information about users and their connections.

Patrick Fletcher helped write this topic.

Software versions used in this topic

- Visual Studio 2013
- .NET 4.5
- SignalR version 2

Previous versions of this topic

For information about earlier versions of SignalR, see SignalR Older Versions.

Questions and comments

Please leave feedback on how you liked this tutorial and what we could improve in the comments at the bottom of the page. If you have questions that are not directly related to the tutorial, you can post them to the ASP.NET SignalR forum or StackOverflow.com.

Introduction

Each client connecting to a hub passes a unique connection id. You can retrieve this value in the `Context.ConnectionId` property of the hub context. If your application needs to map a user to the connection id and persist that mapping, you can use one of the following:

- The User ID Provider (SignalR 2)
- In-memory storage, such as a dictionary
- SignalR group for each user
- Permanent, external storage, such as a database table or Azure table storage

Each of these implementations is shown in this topic. You use the `OnConnected`, `OnDisconnected`, and `OnReconnected` methods of the `Hub` class to track the user connection status.

The best approach for your application depends on:

- The number of web servers hosting your application.
- Whether you need to get a list of the currently connected users.

- Whether you need to persist group and user information when the application or server restarts.
- Whether the latency of calling an external server is an issue.

The following table shows which approach works for these considerations.

	More than one server	Get list of currently connected users	Persist information after restarts	Optimal performance
UserID Provider	✓			✓
In-memory		✓		✓
Single-user groups	✓			✓
Permanent, external	✓	✓	✓	

IUserID provider

This feature allows users to specify what the userId is based on an IRequest via a new interface IUserIdProvider.

The IUserIdProvider

C# Copy

```
public interface IUserIdProvider
{
    string GetUserId(IRequest request);
}
```

By default, there will be an implementation that uses the user's `IPrincipal.Identity.Name` as the user name. To change this, register your implementation of `IUserIdProvider` with the global host when your application starts:

C# Copy

```
GlobalHost.DependencyResolver.Register(typeof(IUserIdProvider), () => new MyIdProvider());
```

From within a hub, you'll be able to send messages to these users via the following API:

Sending a message to a specific user

C# Copy

```
public class MyHub : Hub
{
    public void Send(string userId, string message)
    {
        Clients.User(userId).send(message);
    }
}
```

In-memory storage

The following examples show how to retain connection and user information in a dictionary that is stored in memory. The dictionary uses a `HashSet` to store the connection id. At any time a user could have more than one connection to the SignalR application. For example, a user who is connected through multiple devices or more than one browser tab would have more than one connection id.

If the application shuts down, all of the information is lost, but it will be re-populated as the users re-establish their connections. In-memory storage does not work if your environment includes more than one web server because each server would have a separate collection of connections.

The first example shows a class that manages the mapping of users to connections. The key for the HashSet will be the user's name.

C# Copy

```
using System.Collections.Generic;
using System.Linq;

namespace BasicChat
{
    public class ConnectionMapping<T>
    {
        private readonly Dictionary<T, HashSet<string>> _connections =
            new Dictionary<T, HashSet<string>>();

        public int Count
        {
            get
            {
                return _connections.Count;
            }
        }

        public void Add(T key, string connectionId)
        {
            lock (_connections)
            {
                HashSet<string> connections;
                if (!_connections.TryGetValue(key, out connections))
                {
                    connections = new HashSet<string>();
                    _connections.Add(key, connections);
                }

                lock (connections)
                {
                    connections.Add(connectionId);
                }
            }
        }

        public IEnumerable<string> GetConnections(T key)
        {
            HashSet<string> connections;
```

```
                if (_connections.TryGetValue(key, out connections))
                {
                    return connections;
                }

                return Enumerable.Empty<string>();
            }

            public void Remove(T key, string connectionId)
            {
                lock (_connections)
                {
                    HashSet<string> connections;
                    if (!_connections.TryGetValue(key, out connections))
                    {
                        return;
                    }

                    lock (connections)
                    {
                        connections.Remove(connectionId);

                        if (connections.Count == 0)
                        {
                            _connections.Remove(key);
                        }
                    }
                }
            }
        }
    }
```

The next example shows how to use the connection mapping class from a hub. The instance of the class is stored in a variable name `_connections`.

C# Copy

```
using System.Threading.Tasks;
using Microsoft.AspNet.SignalR;

namespace BasicChat
{
    [Authorize]
    public class ChatHub : Hub
    {
        private readonly static ConnectionMapping<string> _connections =
            new ConnectionMapping<string>();

        public void SendChatMessage(string who, string message)
        {
            string name = Context.User.Identity.Name;

            foreach (var connectionId in _connections.GetConnections(who))
            {
                Clients.Client(connectionId).addChatMessage(name + ": " + message);
            }
        }

        public override Task OnConnected()
        {
            string name = Context.User.Identity.Name;

            _connections.Add(name, Context.ConnectionId);

            return base.OnConnected();
        }

        public override Task OnDisconnected(bool stopCalled)
        {
            string name = Context.User.Identity.Name;

            _connections.Remove(name, Context.ConnectionId);

            return base.OnDisconnected(stopCalled);
        }
        public override Task OnReconnected()
        {
            string name = Context.User.Identity.Name;

            if (!_connections.GetConnections(name).Contains(Context.ConnectionId))
            {
                _connections.Add(name, Context.ConnectionId);
            }

            return base.OnReconnected();
        }
    }
}
```

Single-user groups

You can create a group for each user, and then send a message to that group when you want to reach only that user. The name of each group is the name of the user. If a user has more than one connection, each connection id is added to the user's group.

You should not manually remove the user from the group when the user disconnects. This action is automatically performed by the SignalR framework.

The following example shows how to implement single-user groups.

C# Copy

```
using Microsoft.AspNet.SignalR;
using System;
using System.Threading.Tasks;

namespace BasicChat
{
    [Authorize]
    public class ChatHub : Hub
    {
        public void SendChatMessage(string who, string message)
        {
            string name = Context.User.Identity.Name;

            Clients.Group(who).addChatMessage(name + ": " + message);
        }

        public override Task OnConnected()
        {
            string name = Context.User.Identity.Name;

            Groups.Add(Context.ConnectionId, name);

            return base.OnConnected();
        }
    }
}
```

Permanent, external storage

This topic shows how to use either a database or Azure table storage for storing connection information. This approach works when you have multiple web servers because each web server can interact with the same data repository. If your web servers stop working or the application restarts, the `OnDisconnected` method is not called. Therefore, it is possible that your data repository will have records for connection ids that are no longer valid. To clean up these orphaned records, you may wish to invalidate any connection that was created outside of a timeframe that is relevant to your application. The examples in this section include a value for tracking when the connection was created, but do not show how to clean up old records because you may want to do that as background process.

Database

The following examples show how to retain connection and user information in a database. You can use any data access technology; however, the example below shows how to define models using Entity Framework. These entity models correspond to database tables and fields. Your data structure could vary considerably depending on the requirements of your application.

The first example shows how to define a user entity that can be associated with many connection entities.

C# Copy

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace MapUsersSample
{
    public class UserContext : DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<Connection> Connections { get; set; }
    }

    public class User
    {
        [Key]
        public string UserName { get; set; }
        public ICollection<Connection> Connections { get; set; }
    }

    public class Connection
    {
        public string ConnectionID { get; set; }
        public string UserAgent { get; set; }
        public bool Connected { get; set; }
    }
}
```

Then, from the hub, you can track the state of each connection with the code shown below.

C# Copy

```
using System;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;
using System.Collections.Concurrent;
using Microsoft.AspNet.SignalR;

namespace MapUsersSample
{
    [Authorize]
    public class ChatHub : Hub
    {
        public void SendChatMessage(string who, string message)
        {
            var name = Context.User.Identity.Name;
            using (var db = new UserContext())
            {
                var user = db.Users.Find(who);
                if (user == null)
                {
                    Clients.Caller.showErrorMessage("Could not find that user.");
                }
                else
                {
                    db.Entry(user)
                        .Collection(u => u.Connections)
                        .Query()
                        .Where(c => c.Connected == true)
                        .Load();

                    if (user.Connections == null)
                    {
                        Clients.Caller.showErrorMessage("The user is no longer connected.");
                    }
                    else
                    {
                        foreach (var connection in user.Connections)
                        {
                            Clients.Client(connection.ConnectionID)
                            .addChatMessage(name + ": " + message);
                    }
                }
            }
        }
    }

    public override Task OnConnected()
    {
        var name = Context.User.Identity.Name;
        using (var db = new UserContext())
        {
            var user = db.Users
                .Include(u => u.Connections)
                .SingleOrDefault(u => u.UserName == name);

            if (user == null)
            {
                user = new User
                {
                    UserName = name,
                    Connections = new List<Connection>()
                };
                db.Users.Add(user);
            }

            user.Connections.Add(new Connection
            {
                ConnectionID = Context.ConnectionId,
                UserAgent = Context.Request.Headers["User-Agent"],
                Connected = true
            });
            db.SaveChanges();
        }
        return base.OnConnected();
    }

    public override Task OnDisconnected(bool stopCalled)
    {
            using (var db = new UserContext())
            {
                var connection = db.Connections.Find(Context.ConnectionId);
                connection.Connected = false;
                db.SaveChanges();
            }
            return base.OnDisconnected(stopCalled);
        }
    }
}
```

Azure table storage

The following Azure table storage example is similar to the database example. It does not include all of the information that you would need to get started with Azure Table Storage Service. For information, see How to use Table storage from .NET.

The following example shows a table entity for storing connection information. It partitions the data by user name, and identifies each entity by the connection id, so a user can have multiple connections at any time.

C# Copy

```
using Microsoft.WindowsAzure.Storage.Table;
using System;

namespace MapUsersSample
{
    public class ConnectionEntity : TableEntity
    {
        public ConnectionEntity() { }

        public ConnectionEntity(string userName, string connectionID)
        {
            this.PartitionKey = userName;
            this.RowKey = connectionID;
        }
        }
    }
```

In the hub, you track the status of each user's connection.

C# Copy

```
using Microsoft.AspNet.SignalR;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Table;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace MapUsersSample
{
    public class ChatHub : Hub
    {
        public void SendChatMessage(string who, string message)
        {
            var name = Context.User.Identity.Name;

            var table = GetConnectionTable();

            var query = new TableQuery<ConnectionEntity>()
                .Where(TableQuery.GenerateFilterCondition(
                "PartitionKey",
                QueryComparisons.Equal,
                who));

            var queryResult = table.ExecuteQuery(query).ToList();
            if (queryResult.Count == 0)
            {
                Clients.Caller.showErrorMessage("The user is no longer connected.");
            }
            else
            {
                foreach (var entity in queryResult)
                {
                    Clients.Client(entity.RowKey).addChatMessage(name + ": " + message);
                }
            }
        }

        public override Task OnConnected()
        {
            var name = Context.User.Identity.Name;
            var table = GetConnectionTable();
            table.CreateIfNotExists();

            var entity = new ConnectionEntity(
                name.ToLower(),
                Context.ConnectionId);
            var insertOperation = TableOperation.InsertOrReplace(entity);
            table.Execute(insertOperation);

            return base.OnConnected();
        }

        public override Task OnDisconnected(bool stopCalled)
        {
            var name = Context.User.Identity.Name;
            var table = GetConnectionTable();

            var deleteOperation = TableOperation.Delete(
                new ConnectionEntity(name, Context.ConnectionId) { ETag = "*" });
            table.Execute(deleteOperation);

            return base.OnDisconnected(stopCalled);
        }

        private CloudTable GetConnectionTable()
        {
            var storageAccount =
                CloudStorageAccount.Parse(
                CloudConfigurationManager.GetSetting("StorageConnectionString"));
            var tableClient = storageAccount.CreateCloudTableClient();
            return tableClient.GetTableReference("connection");
        }
    }
}
```