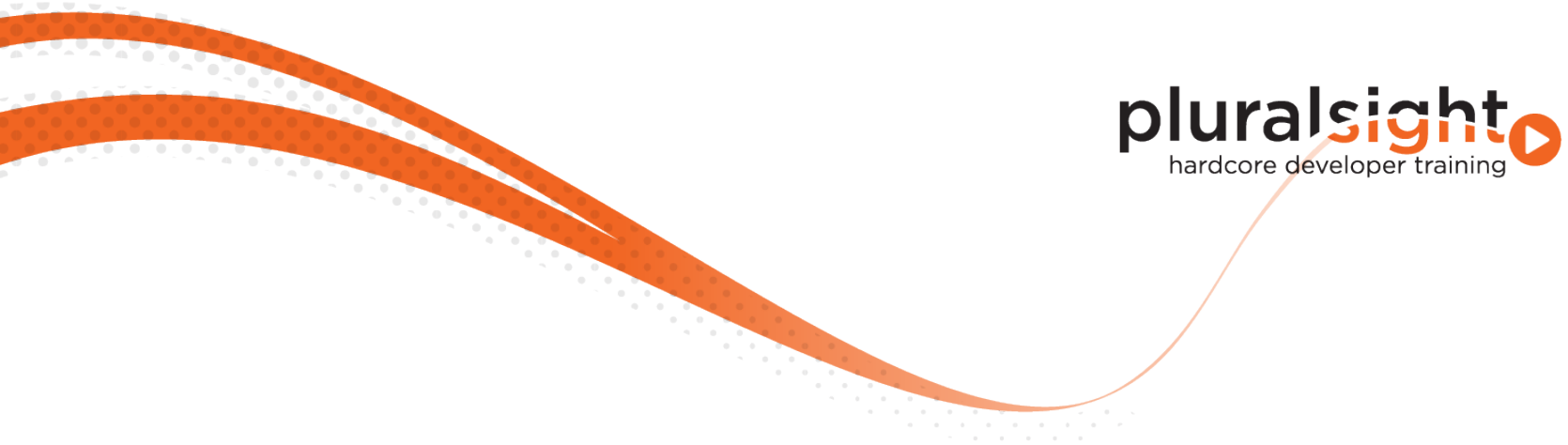
Introducing ASP.NET SignalR - Push Services with Hubs

SignalR Hubs

Christian Weyer
christian.weyer@thinktecture.com
http://www.thinktecture.com
@christianweyer



pluralsight
hardcore developer training

Outline

- Concepts & programming model
- Protocol
- Clients, Groups
- Lifecycle

Hubs concept

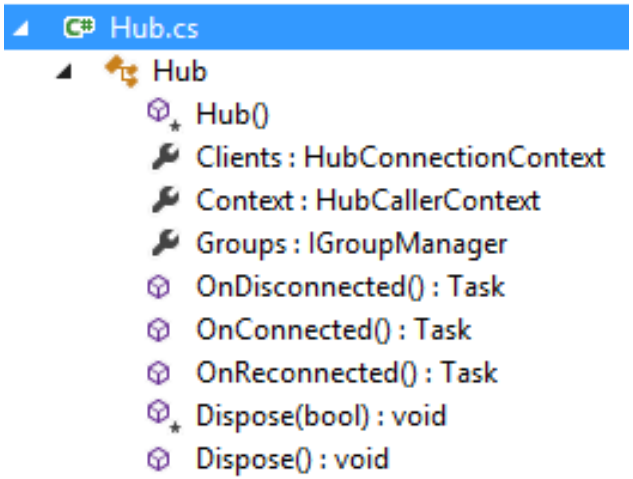
- **Hubs are classes to implement push services in SignalR**
  - Abstraction on top of persistent connection
  - Convention-over-configuration
- **Hubs provide a higher level RPC framework**
  - Perfect for different types of messages to send between server and client

- **Hub conventions**
  - Public methods are callable from the outside
  - Send messages to clients by invoking client-side methods

- **Simple steps to get going**
  1. Write hub class
  2. Add route in host process
  3. Done

Programming model

- **Hub name reflected into external API**
  - `[HubName]` can alias name



- **Return simple type, complex type or Task**
- **Complex objects and arrays of objects automatically serialized/deserialized to/from JSON**
  - Default is JSON.NET

- **`Context` holds connection- and request-specific information**
  - `ConnectionId`
  - `Request`
  - `Headers`
  - `RequestCookies`
  - `QueryString`
  - `User`

DEMO

Get going with a Hub

Hubs protocol

- **Base endpoint is `/signalr`**
- ***Optional*: get JS metadata from `/signalr/hubs`**

- **Basically two protocol steps (details vary by transport)**
  - `/negotiate`: which transport do you support?
  - (*optional*: `/ping` server)
  - `/connect`: OK, here is my persistent connection

- **'Best' transport negotiation**
  - Web Sockets ⇨ SSE ⇨ Forever frame ⇨ Long polling

- **Pre-defined payload**
- **Any data is JSON encoded**

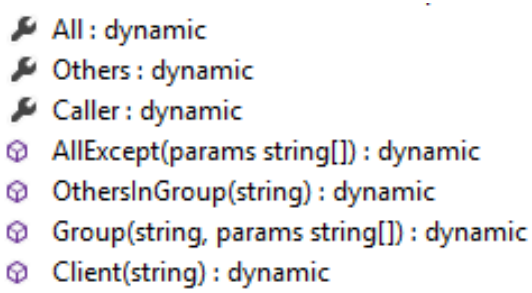
```
C: Cursor
M: Messages
   H: Hub name
   M: Method name
   A: Method args
T: Timeout
D: Disconnect
```

DEMO

Protocol interaction

Pushing data: Clients

- `Clients` property as dispatching point to send messages to clients
- Holds `dynamic` properties and methods



- Target method with parameters is dynamically 'injected' into code path, serialized, and embedded into protocol response

```
public void SendMessage(string message)
{
    var msg = string.Format("{0}: {1}", Context.ConnectionId, message);
    Clients.All.newMessage(msg);
}
```

DEMO

Pushing data to clients

Pushing data: Groups

- **Groups**
  - Typical base pattern in push scenarios
  - Can add connections to groups and send messages to particular groups

```
public void JoinRoom(string room)
{
    Groups.Add(Context.ConnectionId, room);
}
public void SendMessageForRoom(string room, string message)
{
    var msg = string.Format("{0}: {1}", Context.ConnectionId, message);
    Clients.Group(room).newMessage(msg);
}
```

- **Groups are not persisted on the server**
  - We need to keep track of what connections are in what groups
  - No automatic group count

DEMO

Pushing data to groups

Hub lifecycle

- **Hub connections have a lifecycle**
  - Override async pipeline event handlers

```
public override Task OnConnected()
{
   …
}
public override Task OnDisconnected()
{
   …
}
public override Task OnReconnected()
{
   …
}
```

- **Sending data from outside a hub**
  - Retrieve hub context via dependency resolver

```
private void SendMonitorData(string eventType, string connection)
{
   var context = GlobalHost.ConnectionManager.GetHubContext<MonitorHub>();
   context.Clients.All.newEvent(eventType, connection);
}
```

DEMO

Hub lifecycle

Summary

- Simple server-side programming model
- JSON-serialized data over Hubs protocol
- Pushing data to clients and/or groups
- Lifecycle allows to hook into events to react accordingly