.NET Core, ASP.NET Core, and ASP.NET Core MVC

brave new world
Outline

• Motivation
• .NET Core
• ASP.NET Core
• ASP.NET Core MVC
Motivation

- .NET has a strong history
  - Very popular
  - Lots of investments

- There is more than just Windows
  - Many more platforms, devices, and clouds

- .NET is evolving
  - .NET needs to learn to run in more places
  - .NET needs modern tooling
.NET runtimes target a platform

- .NET Framework
  - Windows-only
- .NET Core
  - Cross-platform runtime
- .NET Native (UWP)
- Mono
- Windows Phone
- More...
.NET Core: next gen .NET for server apps

- Cross platform
  - Windows, Linux, Mac, FreeBSD

- Portable
  - Can be ~/bin deployed
  - Can be user or machine installed as well

- Open source
  - https://github.com/dotnet/coreclr
  - Contains core runtime and mscorlib (e.g. GC, JIT, BCL)
  - Does not contain many frameworks (e.g. WCF, WPF)
Development ecosystem

• SDK
  • Command-line tooling (*dotnet*)

• Project system
  • File-system based project system (*project.json*)

• Runtime, libraries, and packaging
  • NuGet-focused

• Editors/IDEs
  • Any text editor (VS Code, Emacs, Sublime, etc) and OmniSharp (OSS)
  • Visual Studio (Microsoft)
  • Project Rider (JetBrains)
Installing .NET SDK

• Use nightly builds (until RC2 is released)
  • https://github.com/dotnet/cli
dotnet : command line tool

- Create new project
- Install NuGet dependencies
- Build application
- Load .NET and run application
- Package library
- Publish application
dotnet new

• Creates new project
  • program.cs
  • project.json
• Console-based application

```csharp
using System;
namespace ConsoleApplication
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```
project.json

- Project type
  - Application or library
- Dependencies
  - Primarily from NuGet
- Target framework(s)
  - Target framework moniker (TFM)
.NET platform standard

- Identifier (TFM) for required framework
  - Replacement for PCL platform versioning nightmare
- Libraries target an expected API from framework
  - "netstandard1.0", "netstandard1.1", ..., "netstandard1.5"
  - Can use libraries from earlier .NET Standard version
- Applications target a specific platform (and thus framework)
  - "net451", "net452", "net46", "net461", "netcoreapp1.0", etc...
  - Platforms support a specific .NET Standard version

### Platform support for .NET Standard

<table>
<thead>
<tr>
<th>Target Platform Name</th>
<th>Alias</th>
<th>1.0</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>1.4</th>
<th>1.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET Platform Standard</td>
<td>netstandard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.NET Core</td>
<td>netcoreapp</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>1.0</td>
</tr>
<tr>
<td>.NET Framework</td>
<td>net</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>4.6.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>4.6.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>4.5.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>4.5.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→</td>
<td></td>
<td></td>
<td>→</td>
<td>→</td>
<td>4.5</td>
</tr>
<tr>
<td>Universal Windows Platform</td>
<td>uap</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>10.0</td>
</tr>
<tr>
<td>Windows</td>
<td>win</td>
<td>→</td>
<td>→</td>
<td></td>
<td></td>
<td></td>
<td>8.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.0</td>
</tr>
<tr>
<td>Windows Phone</td>
<td>wpa</td>
<td>→</td>
<td>→</td>
<td></td>
<td></td>
<td></td>
<td>8.1</td>
</tr>
<tr>
<td>Windows Phone Silverlight</td>
<td>wp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.0</td>
</tr>
<tr>
<td>Mono/Xamarin Platforms</td>
<td></td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>*</td>
</tr>
<tr>
<td>Mono</td>
<td></td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>*</td>
</tr>
</tbody>
</table>
dotnet restore

• Downloads NuGet dependencies
  • Might need a local nuget.config to target nightly builds from myget.org

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <!--To inherit the global NuGet package sources remove the <clear/> line below -->
    <clear/>
    <add key="dotnet-core" value="https://dotnet.myget.org/F/dotnet-core/api/v3/index.json"/>
    <add key="AspNetCI" value="https://www.myget.org/F/aspnetcirelease/api/v3/index.json"/>
  </packageSources>
</configuration>
```

• Builds project.json.lock
  • Snapshot of dependency versions
  • Needed to load application
**dotnet build / dotnet run / dotnet app.dll**

- Builds project, or builds and runs application
  - -c indicates configuration (release/debug)
  - -f indicates framework to target
  - -v emits verbose log output
- Binaries output to ~/bin/<configuration>/<framework> folder
ASP.NET Core

- The new pipeline
- Middleware
- Dependency Injection
- Configuration
Motivation

• Modern web stack
  • Modern package system (NuGet)
  • Lightweight/composable runtime
  • Dependency injection
  • Flexible configuration/deployment
<globalModules>
  <add name="UriCacheModule" image="%windir%\System32\inetsrv\cachuri.dll"/>
  <add name="FileCacheModule" image="%windir%\System32\inetsrv\cachfile.dll"/>
  <add name="TokenCacheModule" image="n"/>
  <add name="HttpCacheModule" image="n"/>
  <add name="StaticCompressionModule" image="n"/>
  <add name="DefaultDocumentModule" image="n"/>
  <add name="DirectoryListingModule" image="n"/>
  <add name="ProtocolSupportModule" image="n"/>
  <add name="StaticFileModule" image="n"/>
  <add name="AnonymousAuthenticationModule" image="n"/>
  <add name="CertificateMappingAuthenticationModule" image="n"/>
  <add name="BasicAuthenticationModule" image="n"/>
  <add name="WindowsAuthenticationModule" image="n"/>
  <add name="ISICertificateMappingAuthenticationModule" image="n"/>
  <add name="RequestFilteringModule" image="n"/>
  <add name="CustomErrorModule" image="n"/>
  <add name="IsapiModule" image="n"/>
  <add name="IisapiModule" image="n"/>
  <add name="ApplicationInitializationModule" image="n"/>
  <add name="WebSocketModule" image="n"/>
  <add name="ManagedEngineV4.0_32bit" image="n"/>
  <add name="ManagedEngineV4.0_64bit" image="n"/>
</globalModules>
"Empty Web Application"

An example of a web server written with Node which responds with 'Hello World':

```javascript
var http = require('http);

http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(8124);

console.log('Server running at http://127.0.0.1:8124/');
```
About

OWIN defines a standard interface between .NET web servers and web applications. The goal of the OWIN interface is to decouple server and application, encourage the development of simple modules for .NET web development, and, by being an open standard, stimulate the open source ecosystem of .NET web development tools.
OWIN Middleware Architecture

- Middleware are linked components that process requests
- Application code targeting a framework
ASP.NET Core

• ASP.NET Core is HTTP pipeline implementation
  • sits on top of .NET Core
  • uses the middleware concept (but at a higher abstraction level than OWIN)
  • comes with its own server (Kestrel)
  • adds DI to provide services

• ASP.NET Core MVC is Microsoft's application framework
How ASP.NET Core Applications start

IIS → ASP.NET Core Module

start process → Application.dll

dotnet run

Startup.cs

ConfigureServices(...) → Configure(...)
Loading ASP.NET Core

```csharp
public class Program
{
    public static void Main()
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseIISIntegration()
            .UseStartup<Startup>()
            .Build();

        host.Run();
    }
}

public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        ...
    }
}
```
Pipeline primitives

```
app.Use(context, next)
app.Map("/path")
app.Use(context, next)
app.Run(context)
```
namespace Microsoft.AspNetCore.Builder
{
    public delegate Task RequestDelegate(HttpContext context);
}

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello ASP.NET Core");
});
Map

```csharp
app.Map("/hello", helloApp =>
{
    helloApp.Run(async (HttpContext context) =>
    {
        await context.Response.WriteAsync("Hello ASP.NET Core");
    });
});
```
Use

```csharp
app.Use(async (context, next) =>
{
    if (!context.Request.Path.Value.EndsWith("/favicon.ico"))
    {
        Console.WriteLine("pre");
        Console.WriteLine(context.Request.Path);

        await next();

        Console.WriteLine("post");
        Console.WriteLine(context.Response.StatusCode);
    }
    else
    {
        await next();
    }
});
```
Middleware classes

```csharp
app.UseMiddleware<InspectionMiddleware>();

public class InspectionMiddleware
{
    private readonly RequestDelegate _next;

    public InspectionMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext context)
    {
        Console.WriteLine($"request: {context.Request.Path}");
        await _next(context);
    }
}
```
Dependency Injection

• Various places
  • Configure
  • Middleware classes
  • Higher-level frameworks (e.g. MVC controller)

• Host provided dependencies (e.g. IHostingEnvironment, ILoggerFactory)

• Dependencies set up in ConfigureServices
public class Startup
{
    public Startup(IHostingEnvironment environment)
    {
        /* stuff */
    }

    public void ConfigureServices(IServiceCollection services)
    {
        /* register more stuff */
    }

    public void Configure(IApplicationBuilder app, ILoggerFactory loggerFactory)
    {
        /* add middleware */
    }
}
Registering dependencies

- New instance "per call"

```csharp
services.AddTransient<IMyCustomService, MyCustomService>();
```

- New instance per HTTP request

```csharp
services.AddScoped<IMyCustomService, MyCustomService>();
```

- Singleton

```csharp
services.AddSingleton<IMyCustomService, MyCustomService>();
```
Configuration

• web.config is no more

• New configuration system based on key/value pairs
  • command line
  • environment variables
  • JSON files
  • INI files

• Configuration can come from multiple sources
  • last source wins
public class Startup
{
    public IConfiguration Configuration { get; set; }

    public Startup(IHostingEnvironment env)
    {
        Configuration = new ConfigurationBuilder()
            .SetBasePath(env.ContentRootPath)
            .AddJsonFile("config.json")
            .AddJsonFile($"config.{env.EnvironmentName}.json", optional: true)
            .AddEnvironmentVariables()
            .Build();
    }

    // more
}
Using configuration

```csharp
public class Startup
{
    IConfiguration _configuration;

    public Startup()
    {
        _configuration = new ConfigurationBuilder()
            ...
            .Build();
    }

    public void Configure(IApplicationBuilder app)
    {
        var copyright = new Copyright
        {
            Company = _configuration.Get("copyright_company"),
            Year = _configuration.Get("copyright_year")
        };

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync($"Copyright {copyright.Year}, {copyright.Company}");
        });
    }
}
```
ASP.NET Core MVC

• Packaging
• Middleware
• Routing and action selection
• Controller initialization
• Model binding changes
• Razor
• Filters
• APIs
• Error handling
Packaging

• MVC is packaged entirely as a NuGet
  • Microsoft.AspNetCore.Mvc

```json
{
  "dependencies": {
    "Microsoft.AspNetCore.Mvc": "1.0.0-rc2-*,
    "Microsoft.AspNetCore.Server.Kestrel": "1.0.0-rc2-"
  }
}
```
Middleware

• MVC is configured as middleware
  • In ConfigureServices via AddMvc
  • In Configure via UseMvc

```csharp
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app)
    {
        app.UseMvc();
    }
}
```
Overriding default settings

- Delegate callback param used to override defaults
- Also fluent API on result from AddMvc()

```csharp
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc(mvc =>
        {
            mvc.Filters.Add(...);
            mvc.ViewEngines.Add(...);
            mvc.InputFormatters.Add(...);
            mvc.OutputFormatters.Add(...);
        });
    }
}
```
Routing

- Routes configured via UseMvc
  - RouteParameters.Optional from MVC 5 removed

```csharp
public void Configure(IApplicationBuilder app)
{
    app.UseMvc(routes =>
    {
        routes.MapRoute("old_default",
            "{controller}/{action}",
            new {
                controller = "Home", action="Index"
            });

        routes.MapRoute("new_default",
            "{controller=Home}/{action=Index}/{id?}");
    });
}
```
Controllers

- Controller base class still provided
  - Action results now implement IActionResult
  - Controller base provides many helpers to create action results
    - View(), PartialView(), Content(), Json(), Ok(), Created(), HttpNotFound(), HttpUnauthorized(), HttpBadRequest(), File(), PhysicalFile(), Redirect(), RedirectPermanent()

```csharp
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```
Attribute routing

• Attribute routing enabled by default

```csharp
public class HomeController : Controller
{
    // ~/ or ~/hello-world
    [Route("/")]
    [Route("/hello-world")]  
    public IActionResult Index()
    {
        return View();
    }
}
```
Attribute routing

- Attribute routing can be applied to class
- [controller] and [action] act as tokens

```csharp
[Route("/[controller]/[action]/")]
public class HomeController : Controller
{
    // ~/Home/Index
    public IActionResult Index()
    {
        return View();
    }
}
```
Combining Route attributes

- Route attributes inherit path
  - RoutePrefix from MVC 5 removed
- Can replace inherited path
  - If template starts with "/" or "~/"

```csharp
[Route("[controller]")]
public class HomeController : Controller
{
    // ~/Home/hello
    [Route("hello")]  
    public IActionResult Index()
    {
        return View();
    }

    // ~/hello
    [Route("/hello")]  
    public IActionResult Index2()
    {
        return View();
    }
}
```
Route parameters

- [Route] allows parameters
  - With {param} syntax
- Supports filters
  - With {param:filter} syntax

```csharp
[Route("[controller]/[action]")]
public class HomeController : Controller
{
    // GET ~/Home/Index
    public IActionResult Index()
    {
        return View();
    }

    // GET ~/Home/Index/5
    [Route("{id:int}\")]
    public IActionResult Index(int id)
    {
        return View();
    }
}
```
HTTP method based routes

- **HttpGet, HttpPost, HttpPut, HttpDelete, HttpPatch**
  - Filter action method on request method
  - Build on [Route] semantics

```csharp
[Route("[controller]/[action]")]
public class HomeController : Controller
{
    // GET ~/Home/Index
    [HttpGet]
    public IActionResult Index()
    {
        return View();
    }

    // ~/Submit
    [HttpPost("/Submit")]
    public IActionResult Submit()
    {
        return View();
    }
}
```
Areas

• Areas defined with the [Area] attribute
  • Used to match an {area} route param
  • Attribute routing allows [area] route token

• Views must still reside under ~/Areas/<area>/Views/<controller>

```csharp
public void Configure(IApplicationBuilder app)
{
    app.UseMvc(routes =>
    {
        routes.MapRoute("new_default",
                        "{area}/{controller=Home}/{action=Index}/{id?}");
    });
}
```

```csharp
[Area("account")]
public class HomeController : Controller
{
    // ...
}
```
POCO controllers

• Controller classes can be POCO
  • Discovered in projects that reference Microsoft.AspNetCore.Mvc.*
  • Identified by "Controller" class name suffix
  • [NonController] disables
Dependency injection

• Can inject dependencies into controller ctor
• Special per-request types can be property injected with decorator attribute
  • ActionContext
  • ControllerContext

```csharp
public class HomeController
{
    IHttpContextAccessor _accessor;

    public HomeController(IHttpContextAccessor accessor)
    {
        _accessor = accessor;
    }

    [ControllerContext]
    public ControllerContext ControllerContext { get; set; }

    // ...
}
```
Razor

• Shared config
  • _ViewStart and _ViewImports

• Chunks
  • @ directives

• TagHelpers
  • Like WebForms custom controls

• ViewComponents
  • Child action replacements
Shared razor configuration

• `_ViewStart.cshtml` still exists
  • Can now easily be put in application root
  • Layout assignment no longer is full path

• `_ViewImports.cshtml` is new
  • Allows for sharing `@using`, `@addTagHelper` chunks across views
  • Can be located in the same places as `_ViewStart.cshtml`
Razor directives (aka chunks)

• @model, @using, @section, @functions still exist
• @helper is gone
• @inject, @addTagHelper are new

• Also, @await Html.PartialAsync() is new
@inject

- Allows dependency injection into view
  - @inject <type> <property>

```csharp
@using Microsoft.Framework.OptionsModel
@inject IOptions<MyConfig> Config

<h2>@Config.Options.SiteName</h2>
```
Tag helpers

• Like custom controls for MVC
  • Allow server-side code to inspect the element
  • Can modify attributes, tag, and/or contents

• @addTagHelper Namespace.ClassName, Assembly
  • Or @addTagHelper *, Assembly

```csharp
@addTagHelper SpanTagHelper, YourProjectName
<span emoji="smile" />
```
Tag helper implementation

- TagHelper base class
  - Class name used to match element
- Override Process or ProcessAsync
  - Inspect element via TagHelperContext
  - Alter output via TagHelperOutput

```csharp
public class SpanTagHelper : TagHelper
{
    public override void Process(
        TagHelperContext context, TagHelperOutput output)
    {
        if (context.AllAttributes.ContainsKey("emoji") &&
            "smile" == context.AllAttributes["emoji"].ToString())
        {
            output.Attributes.Add("title", "smile");
            output.Content.SetContent(" :) ");
            output.SelfClosing = false;
        }
    }
}
```
Tag helper implementation

- `[TargetElement]` can be used to match element
  - Attributes can be used to filter
- `[HtmlAttributeName]` will read incoming attribute
  - Will remove from output

```csharp
[TargetElement("span", Attributes = "emoji")]
public class EmojiTagHelper : TagHelper
{
    [HtmlAttributeName("emoji")]
    public string Emoji { get; set; }

    public override void Process(
        TagHelperContext context,
        TagHelperOutput output)
    {
        if ("smile" == Emoji)
        {
            output.Attributes.Add("title", "smile");
            output.Content.SetContent(" :) ");
            output.SelfClosing = false;
        }
    }
}
```
MVC tag helpers

```html
<a asp-controller="Manage" asp-action="Index">Manage Your Account</a>

<form asp-controller="Account" asp-action="LogOff" method="post"></form>

<environment names="Staging,Production">
    <h1>You're in production!</h1>
</environment>

<link rel="stylesheet"
    href="//ajax.aspnetcdn.com/ajax/bootstrap/3.0.0/css/bootstrap.min.css"
    asp-fallback-href="~/lib/bootstrap/css/bootstrap.min.css"
    asp-fallback-test-class="hidden"
    asp-fallback-test-property="visibility"
    asp-fallback-test-value="hidden" />

<script src="//ajax.aspnetcdn.com/ajax/jquery.validation/1.11.1/jquery.validate.min.js"
    asp-fallback-src="~/lib/jquery-validation/jquery.validate.js"
    asp-fallback-test="window.jquery && window.jquery.validator">" />
</script>
```
Validation tag helpers

```html
<form asp-controller="Account" asp-action="ForgotPassword" method="post">
    <h4>Enter your email.</h4>

    <div asp-validation-summary="ValidationSummary.All" class="text-danger"></div>

    <div>
        <label asp-for="Email"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>

</form>
```
View components

• Replacement for child actions
  • Partial views still exist

• Allow for a partial view that runs controller-like code
  • Supports dependency injection

```csharp
@Component.Invoke("Menu", 3)
```

Or

```csharp
@await Component.InvokeAsync("Menu", 3)
```
View components

- **ViewComponent base class**
  - Matched by class prefix
  - Or can use `[ViewComponent]` on POCO
- **Implement Invoke or InvokeAsync**
  - Returns `IViewComponentResult` or `Task<IViewComponentResult>`

```csharp
public class MenuViewComponent : ViewComponent
{
    ICustomMenuService _menu;

    public MenuViewComponent(ICustomMenuService menu)
    {
        _menu = menu;
    }

    public IViewComponentResult Invoke(int depth)
    {
        var menuModel = _menu.GetMenu(depth);
        return View("Index", menuModel);
    }
}
```
View components

- View component views are under:
  - ~/Views/<controller>/Components/<component>
  - ~/Views/Shared/Components/<component>
Filters

- Dependency injection
- Resource filter
- Async support
TypeFilter

- Allows for filters that require dependency injection
  - Implemented via IFilterFactory

```csharp
public class MyActionFilter : Attribute, IActionFilter
{
    private IHakingEnvironment _env;

    public MyActionFilter(IHostingEnvironment env)
    {
        _env = env;
    }

    // ...
}
```

```csharp
[TypeFilter(typeof(MyFilter))]
public IActionResult Index()
{
    // ...
}
```
IResourceFilter

• Surrounds model binding, action, and result (including those filters)

```csharp
public interface IResourceFilter : IFilter
{
    void OnResourceExecuting(ResourceExecutingContext context);
    void OnResourceExecuted(ResourceExecutedContext context);
}
```

• ResourceExecutingContext
  • Value providers, model binders, input formatters, validation providers
  • Can alter these on each request
Async filters

- All filters now have IAsync<Filter> support
  - Authorization, Resource, Action, Result, Exception
  - Pattern similar to middleware pipeline

```csharp
public class MyResourceFilter : Attribute, IAsyncResourceFilter
{
    public async Task OnResourceExecutionAsync(
        ResourceExecutingContext context, ResourceExecutionDelegate next)
    {
        // pre
        var resourceExecutedContext = await next();
        // post
    }
}
```
Web API

• Formatters
  • Content negotiation
  • Format filters
  • XML support
Formatters

• Formatters have been split into two groups
  • Input formatters triggered via [FromBody]
  • Output formatters triggered via ObjectResult
Input formatters

- InputFormatter base class provides starting point
  - SupportedMediaTypes property used to match Content-Type header
  - [Consumes] filter can be used to limit formatters

<table>
<thead>
<tr>
<th>Formatter</th>
<th>Content type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringInputFormatter</td>
<td>text/plain</td>
<td></td>
</tr>
<tr>
<td>JsonInputFormatter</td>
<td>application/json, text/json</td>
<td></td>
</tr>
<tr>
<td>XmlSerializerInputFormatter</td>
<td>application/xml, text/xml</td>
<td>Not registered by default</td>
</tr>
<tr>
<td>XmlDataContractSerializerInputFormatter</td>
<td>application/xml, text/xml</td>
<td>Not registered by default</td>
</tr>
</tbody>
</table>
Output formatters

- ObjectResult chooses formatter from Accept header
  - OutputFormatter base class has SupportedMediaTypes property
  - ContentTypes property or [Produces] filter can be set explicitly to limit formatters
  - If Accept contains "*/*", then rest of Accept values ignored
    - RespectBrowserAcceptHeader onMvcOptions can change behavior

<table>
<thead>
<tr>
<th>Formatter</th>
<th>Accept type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringOutputFormatter</td>
<td>text/plain</td>
<td></td>
</tr>
<tr>
<td>JsonOutputFormatter</td>
<td>application/json, text/json</td>
<td></td>
</tr>
<tr>
<td>XmlSerializerOutputFormatter</td>
<td>application/xml, text/xml</td>
<td>Not registered by default</td>
</tr>
<tr>
<td>XmlDataContractSerializerOutputFormatter</td>
<td>application/xml, text/xml</td>
<td>Not registered by default</td>
</tr>
</tbody>
</table>
FormatFilter & FormatFilterAttribute

• Allows overriding of Accept header
  • Looks for "format" route or query param
  • FormatterMappings onMvcOptions indicates format to media type mapping
• Sets ContentTypes on ObjectResult

```csharp
public void Configure(IApplicationBuilder app)
{
    app.UseMvc(routes =>
    {
        routes.MapRoute("default",
            "api/{controller} ");

        routes.MapRoute("formatted",
            "api/{controller}.{format}";
    });
}
```
[Produce] & [Consumes] attributes

- Used to control what formatters used on request/response

```csharp
[HttpGet]
[Consumes("application/xml")]
[Produce("application/json")]
public object Get()
{
    return new {...};
}
```
XML compatibility shim

• Library to help migrate from Web API to Core MVC
  • Microsoft.AspNetCore.Mvc.WebApiCompatShim
• Provides old classes that map to the new framework
  • ApiController
  • FromUriAttribute
  • HttpRequestMessage helpers/extensions/model binder
  • HttpResponseMessage helpers/extensions/formatter
  • HttpResponseException
  • HttpRequestException
Error handling

- HandleError from MVC 5 has been removed
- Resource filter's post processing runs after exception filters
  - Last chance place to "handle" exceptions with a result
  - Or just can log exceptions
Error pages

• Diagnostics middleware
  • Microsoft.AspNetCore.Diagnostics
  • UseDeveloperExceptionPage useful for development/debugging error info
  • UseExceptionHandler useful for production error pages
    • Logs error information
    • Invokes error path
Summary

• Brave new (yet somewhat familiar) world
• .NET Core is a cross-platform framework
• ASP.NET Core is a flexible HTTP pipeline architecture
• MVC and Web API have had a quite a make over