Конфигурация в ASP.Net Core

Дмитриев Дмитрий, SmartHead

О чем этот доклад

- Что такое `IConfiguration`
- Поставщики конфигураций
- Строго типизированная конфигурация
- Валидация объектов конфигурации
- Сценарии использования

Плохие примеры

```
public Startup(IConfiguration configuration,
    IHostingEnvironment env)
{
  _environment = env;
  Configuration = new ConfigurationBuilder()
        .AddJsonFile("appsettings.json")
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json")
        .Build();
}
```

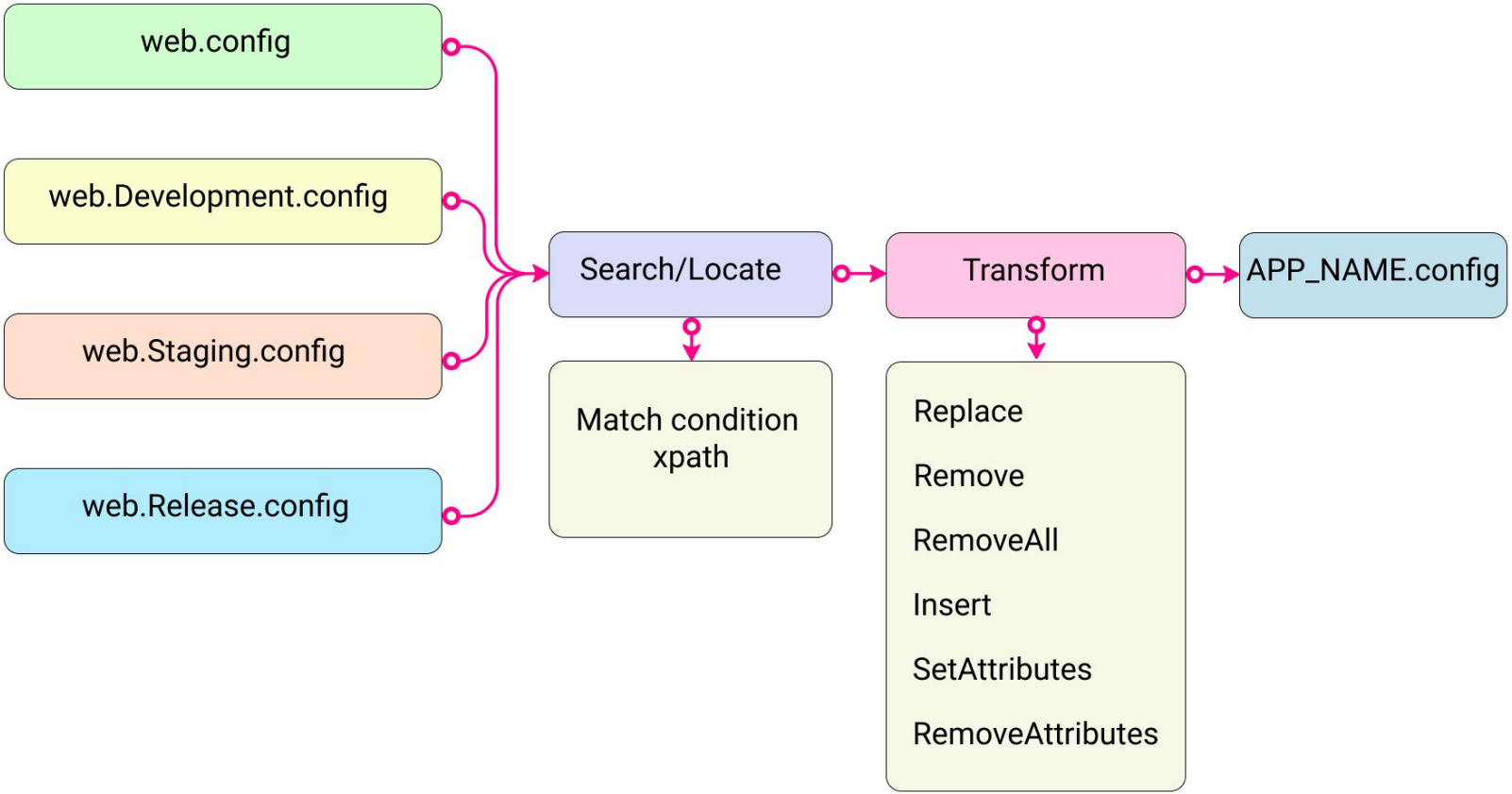
Плохие примеры

```
public Startup(IConfiguration configuration,
    IHostingEnvironment env)
{
  _environment = env;
  Configuration = new ConfigurationBuilder()
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json")
        .Build();
}
```

Плохие примеры

```
public class SampleScopedService
{
    private readonly IConfiguration _configuration;
    public SampleScopedService(IConfiguration configuration)
    {
        _configuration = configuration;
    }
    /* other */
}
```

Трансформации


web.config
web.Development.config
web.Staging.config
web.Release.config
Search/Locate
Match condition
xpath
Transform
Replace
Remove
RemoveAll
Insert
SetAttributes
RemoveAttributes
APP_NAME.config

Секреты

- Windows: `%APPDATA%\microsoft\UserSecrets\<userSecretsId>\secrets.json`
- Linux: `~/.microsoft/usersecrets/<userSecretsId>/secrets.json`
- Mac: `~/.microsoft/usersecrets/<userSecretsId>/secrets.json`

Интерфейс `IConfiguration`

```
public interface IConfiguration
{
  string this[string key] { get; set; }

  IConfigurationSection GetSection(string key);

  IEnumerable<IConfigurationSection> GetChildren();

  IChangeToken GetReloadToken();
}
```

Соглашения

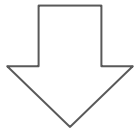
- Key/Value

Соглашения

- Key/Value
- Иерархические данные преобразуются в плоскую структуру

Иерархические данные

```
{
  "Settings": {
   "Key": "I am options"
  }
}
```



Плоская структура

```
"Settings:Key" = "I am options"
```

Соглашения

- Key/Value
- Иерархические данные преобразуются в плоскую структуру
- Нельзя задать значение null

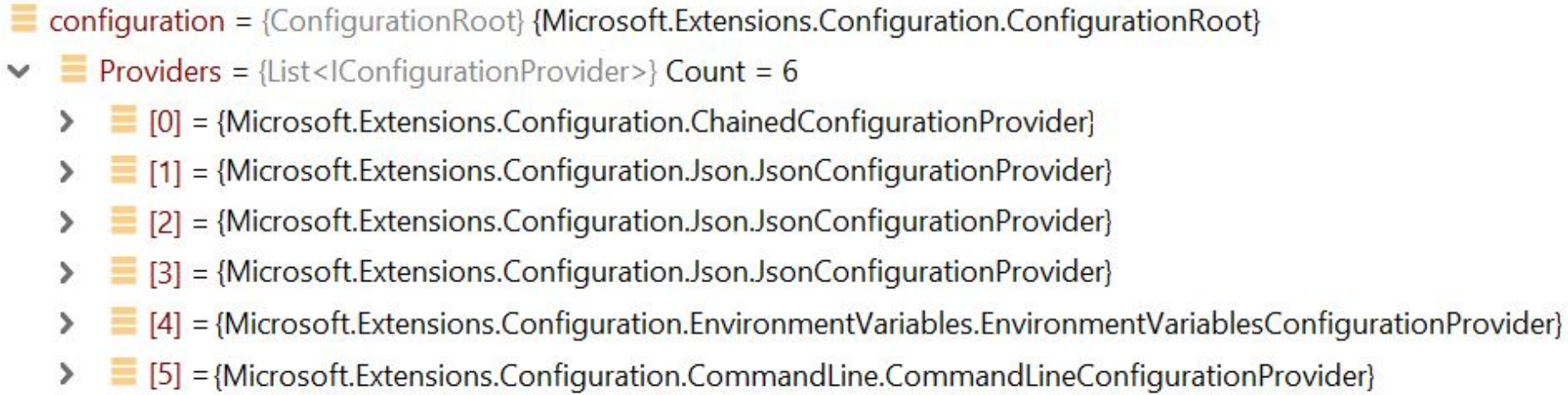
Поставщики конфигураций

Из коробки

- In memory
- Json
- CommandLine
- EnvironmentVariables
- Ini
- Xml
- Azure

Соглашения

- Источники конфигурации считываются в том порядке, в котором они были указаны


configuration = {ConfigurationRoot} {Microsoft.Extensions.Configuration.ConfigurationRoot}
Providers = {List<IConfigurationProvider>} Count = 6
[0] = {Microsoft.Extensions.Configuration.ChainedConfigurationProvider}
[1] = {Microsoft.Extensions.Configuration.Json.JsonConfigurationProvider}
[2] = {Microsoft.Extensions.Configuration.Json.JsonConfigurationProvider}
[3] = {Microsoft.Extensions.Configuration.Json.JsonConfigurationProvider}
[4] = {Microsoft.Extensions.Configuration.EnvironmentVariables.EnvironmentVariablesConfigurationProvider}
[5] = {Microsoft.Extensions.Configuration.CommandLine.CommandLineConfigurationProvider}

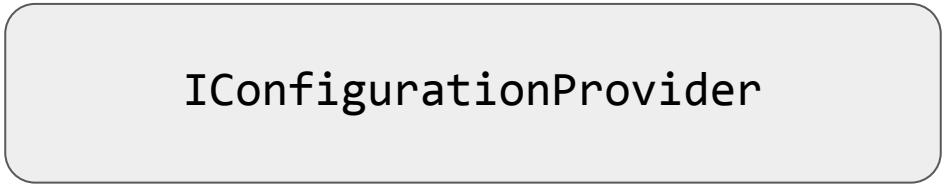
```
public Startup(IConfiguration configuration,
    IHostingEnvironment env)
{
  _environment = env;
  Configuration = new ConfigurationBuilder()
        .AddJsonFile("appsettings.json")
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json")
        .Build();
}
```

Соглашения

- Источники конфигурации считываются в том порядке, в котором они были указаны
- Если в разных источниках конфигурации присутствуют одинаковые ключи (без учета регистра), то используется значение, которое было добавлено последним.

```
public interface IConfigurationProvider
{
    bool TryGet(string key, out string value);
    void Set(string key, string value);
    IChangeToken GetReloadToken();
    void Load();
    IEnumerable<string> GetChildKeys(
            IEnumerable<string> earlierKeys,
            string parentPath);
}
```

```
public interface IConfigurationSource
{
    IConfigurationProvider Build(IConfigurationBuilder builder);
}
```


IConfigurationProvider

IConfigurationProvider

IConfigurationSource

IConfigurationProvider

IConfigurationSource

Extension for

IConfigurationBuilder*

```
public class MyConfigProvider : ConfigurationProvider
{
}

public class MyFileConfigProvider : FileConfigurationProvider
{
}
```

Отслеживание изменений

- Поставщик конфигурации отслеживает изменение источника конфигурации (файл, менеджер секретов)
- Если произошло изменение конфигурации, создается новый IChangeToken
- При изменении токена вызывается перезагрузка конфигурации

```
_changeTokenRegistration = ChangeToken.OnChange(
    () => Source.FileProvider.Watch(Source.Path),
    () => {
        Thread.Sleep(Source.ReloadDelay);
        Load(reload: true);
    });
```

```
_subscription = ChangeToken
    .OnChange(
        () => Configuration.GetReloadToken(),
        () =>  _logger.log("Configuration changed"));
```

IOptions<T>, IOptionsMonitor<T>, IOptionsSnapshot<T>

```
public interface IOptions<out TOptions>
  where TOptions : class, new()
{
    TOptions Value { get; }
}
```

```
public class Settings
{
    public string Key { get; set; }
}
```

```
{
  "Settings": {
   "Key": "I am options"
  }
}
/* ... */

services.Configure<Settings>(
    Configuration.GetSection("Settings"));
/* ... */

public SampleScopedService(IOptions<Settings> options) {}
```

```
public class SampleScopedService
{
    private readonly IConfiguration _configuration;
    public SampleScopedService(IConfiguration configuration)
    {
        _configuration = configuration;
    }
    /* other */
}
```

Как это было раньше?

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>
    <configSections>
        <section name="customSection"
                 type="CustomSection, Samples.CustomSection"/>
    </configSections>
    /* .... */
    <customSection>
        <settings name="Value" />
    </customSection>
</configuration>
```

```
<customSection>
    <settings name="Value" />
</customSection>
```

```
public class CustomSection : ConfigurationSection
{
    [ConfigurationProperty("settings", IsRequired = true)]
    public Settings Settings
    {
        get { return (Settings)this["settings"]; }
        set { this["settings"] = value; }
    }
}
```

```
<customSection>
    <settings name="Value" />
</customSection>
```

```
public class Settings: ConfigurationElement
{
    [ConfigurationProperty("name", IsRequired = true)]
    public string Name
    {
        get { return (string)this["name"]; }
        set { this["name"] = value; }
    }
}
```

```
var section = (CustomSection)ConfigurationManager
     .GetSection( "customSection" );

if ( section != null )
{
   /* .... */
}
```

```
public interface IOptionsMonitor<out TOptions>
{
    TOptions CurrentValue { get; }

    TOptions Get(string name);

    IDisposable OnChange(Action<TOptions, string> listener);
}
```

- IOptionsMonitor подписывается на изменение источника конфигурации
- При изменении - старое значение опций удаляется из кэша, создается новое значение (через IOptionsFactory) и добавляется в кэш
- Все слушатели уведомляются об изменении

```
public interface IOptionsSnapshot<out TOptions>
  : IOptions<TOptions>
     where TOptions : class, new()
{
    TOptions Get(string name);
}
```

- IOptionsSnapshot не подписывается на изменение источника конфигурации
- При каждом запросе создается новый IOptionsSnapshot, кэш очищается
- При обращении к опции создается новое значение (через IOptionsFactory) и добавляется в кэш

```
{
  "Main": {},
  "Reserve": {},
}
/* ... */

services.Configure<Settings>(
    "Main",
    Configuration.GetSection("Main"));
/* ... */

var value = IOptionsSnapshot<T>.Get("Main");
```

```
public class SettingsOptions
{
    [Required(AllowEmptyStrings = false)]
    public string Key { get; set; }
}
/* ... */

services
   .AddOptions<SettingsOptions>()
   .Bind(Configuration.GetSection("SettingsOptions"))
   .ValidateDataAnnotations()
   .Validate((options) => options.Key == "I am a teapot",
            "Error message");
```

Недостатки

- Валидация происходит непосредственно при обращении к объекту
- Вызов services.Configure<T> перезапишет настройку валидации

Фабрика опций и сценарии постконфигурации

```
public class OptionsFactory<TOptions>
  : IOptionsFactory<TOptions> where TOptions : class, new()
{
  IEnumerable<IConfigureOptions<TOptions>> _setups;
  IEnumerable<IPostConfigureOptions<TOptions>> _postConfigures;

  /* Other methods */
  public TOptions Create(string name)
  {
     /* Implementation */
  }
}
```

```
services.PostConfigure<SettingsOptions>(
      "Main",
      opt => {
          opt.Key = "Configure";
      });

services.PostConfigureAll<SettingsOptions>(
      opt => {
          opt.Key = "ConfigureAll";
      });
```

```
public class OptionsConfigurator
    : IPostConfigureOptions<NamedOptions>
{
    public void PostConfigure(string name, NamedOptions options)
    {
        throw new System.NotImplementedException();
    }
}
```

Сценарии использования

Трансформации

appsettings.json

```
SomeSettings:Key = "Value"
ConnectionStrings:Default = "Some Value"
```

secrets.json

```
NewSetting = "Value"
ConnectionStrings:Default = "Secret Value"
```

Трансформации

```
SomeSettings:Key = "Value"
NewSetting = "Value"
ConnectionStrings:Default = "Secret Value"
```

Переменные окружения

appsettings.json

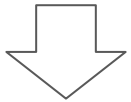
```
SomeSettings:Key = "Value"
ConnectionStrings:Default = "Some Value"
```

Environment variable

```
ConnectionStrings__Default: "Value"
```

Переменные окружения

```
ConnectionStrings__Default: "Value"
```



```
ConnectionStrings:Default: "Value"
```

Переменные окружения

```
SomeSettings:Key = "Value"
ConnectionStrings:Default = "Value"
```

Выводы

- 😀 Большая гибкость и лучшая расширяемость
- 😀 Строгая типизация объектов конфигурации
- 😀 Валидация конфигурации
- 😀 Возможность использовать в .Net Framework проектах
- 😀 Возможность получать конфигурацию из удаленных источников (Azure Key Vault, HashiCorp Key Vault)

- 😭 Отсутствие валидации при запуске приложения
- 😭 Сложность отладки
- 😭 Сложности с использованием опций в .Net Framework проектах

https://github.com/qdimka/netcore-configuration-sample


Типичный сингапурец смотрит
на нашу презентацию
Спасибо за внимание,
смотрите шире))