riganti

Přechod z Newtonsoft.Json na System.Text.Json

Tomas Herceg

CEO @ RIGANTI
Microsoft MVP
Founder of Update Conference
Open-source maintainer

Kdy je to potřeba?

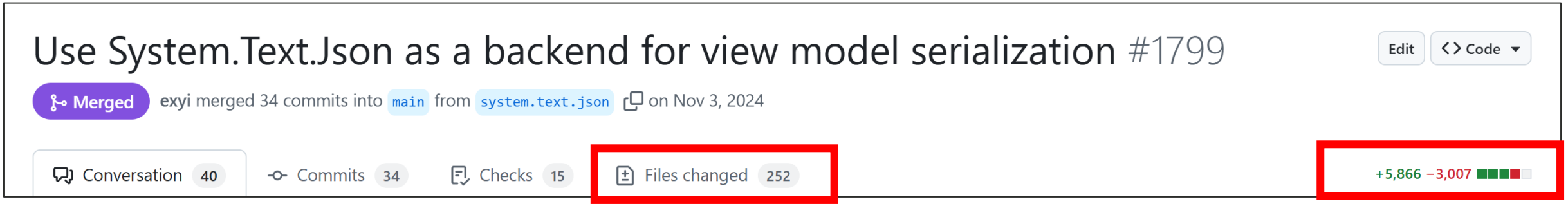
- **Přechod na ASP.NET Core**
  - API projekty
- **Chcete latest & greatest**
- **Jste autoři knihovny**, která potřebuje JSON

Nestačí tohle?

```
JsonConvert.SerializeObject(obj)
```

```
JsonSerializer.Serialize(obj)
```

Nestačí tohle?



- Nebyl to jediný pull request
  - Následovaly další funkce a bug fixy

Co dělá DotVVM s JSONem

```
<div class="form-group">

    <dot:TextBox Text="{value: Number}" />

    <dot:Button Text="Increment"
        Click="{command: Increment()}" />

</div>
```

```
public class IncrementViewModel
{
    public int Number { get; set; }

    public void Increment()
    {
        Number++;
    }
}
```

Co dělá DotVVM s JSONem

```
public class OrderViewModel(OrderService orderService)
{
    [Bind(Direction.ServerToClient)]
    public string OrderId { get; set; }

    [Bind(Direction.ServerToClientFirstRequest)]
    public List<CountryModel> Countries { get; set; }

    [Protect(ProtectMode.Sign)]
    public double TotalPrice { get; set; }

}
```

Co dělá DotVVM s JSONem

- Serializace a deserializace
  - Včetně naplnění (populate) existujícího objektu
  - Speciální zacházení s konstruktory – dependency injection
- Některé nody se přenášejí jen jedním směrem
- Některé nody se šifrují nebo podepisují
- JSON diffing & patching
- Chceme využít UTF-8 tam, kde to jde

Zajímavosti System.Text.Json

- Pracuje nad UTF-8 streamem
  - Nemusí ho celý držet v paměti
  - Až 50% úspora místa oproti použití `string`
- Snaha o co nejméně alokací
  - Interně využívá `Span<T>`, aby nemusel vytvářet substringy
    - „Pohled“ na určitý segment vstupního bufferu
- Dvě DOM reprezentace JSONu
  - `JsonDocument` / `JsonElement`... – read-only
  - `JsonNode` / `JsonObject`... - mutable

JsonConverter

- Změnila se bázová třída
  - Generický `JsonConverter<T>` nebo `JsonConverterFactory`

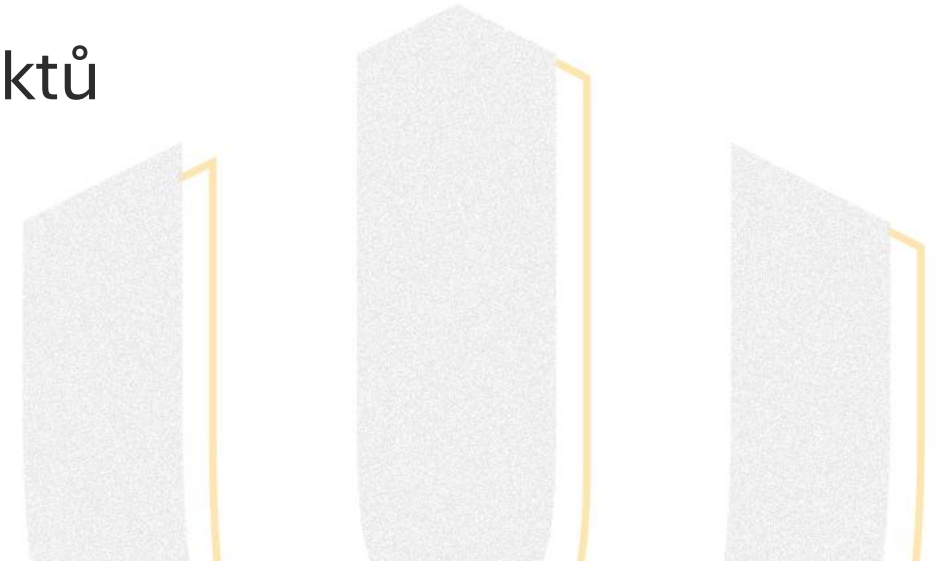
- `Utf8JsonReader` a `Utf8JsonWriter`

```
if (reader.ValueTextEquals("key"u8)) …
```

- Změny v namespaces a API
  - `JsonToken` ➔ `JsonTokenType`
  - `ReadAsString` ➔ `GetString`

JsonConverters v DotVVM

- Serializace konfigurace pro účely IntelliSense ve Visual Studiu
- Vlastní reprezentace `Dictionary<TKey, TValue>`
  - Serializujeme jako pole, aby šlo používat ne-stringové klíče
- Doplňování `$type` anotací do DotVVM objektů
  - DotVVM provádí typovou kontrolu na klientovi



JsonConverters v DotVVM

- Podepsání nebo šifrování části JSON
  - Ukládají se do speciálního pole `$encryptedValues` v rootu
  - Šifrované hodnoty se z JSONu vynechávají
- `[Bind]` atribut ovlivňuje, jestli se hodnota posílá ze serveru na klienta, z klienta na server, nebo oběma směry
- Vlastní handling UTC u přenášení data a času
  - `DateTime.Kind` může způsobit nechtěné konverze

Enumy

- Newtonsoft.Json podporoval atribut EnumValue
- System.Text.Json 9.0 přidal JsonStringEnumMemberName

```
[Flags]
enum CustomEnum
{
    [JsonStringEnumMemberName("custom_zero")]
    Zero = 0,

    [JsonStringEnumMemberName("custom_one")]
    One = 1,
```

ShouldSerialize* pattern

- `Newtonsoft.Json` respektoval metodu s názvem `ShouldSerialize`*`Property`*`()`
  - Pokud vrátila `false`, vlastnost se neserializovala

```
class Customer
{
    public string Name { get; set; }

    public bool IsCompany { get; set; }

    public string CompanyNumber { get; set; }
    public bool ShouldSerializeCompanyNumber() => IsCompany;
}
```

Polymorfismus

```
BaseType value = new DerivedType();
var result1 = JsonConvert.SerializeObject(value);
var result2 = JsonSerializer.Serialize(value);
```

- `Newtonsoft.Json` serializuje vlastnosti z **DerivedType**
- `System.Text.Json` serializuje jen vlastnosti z **BaseType**

- Newtonsoft umí při deserializaci použít typ z `$type` atributu
  - Nebezpečné, nedoporučuje se u untrusted vstupů
  - V DotVVM tuhle vlastnost používáme jinak

Polymorfismus

- `System.Text.Json` vyžaduje explicitní seznam povolených typů

```
[JsonDerivedType(typeof(WeatherForecastWithCity))]
public class WeatherForecastBase
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string? Summary { get; set; }
}
public class WeatherForecastWithCity : WeatherForecastBase
{
    public string? City { get; set; }
}
```

Defaultní property casing

- `Newtonsoft.Json`
  - Case-insensitive
  - Pascal Case
- `System.Text.Json`
  - Case-sensitive
  - Pascal Case
    - V ASP.NET Core je default camel case

Privátní settery

- `Newtonsoft.Json` používá privátní setter, pokud je explicitně uveden atribut `[JsonProperty]`

- `System.Text.Json` vyžaduje ještě `[JsonInclude]`
  - `[JsonPropertyName]` nestačí

```
[JsonInclude]
[JsonPropertyName("Name")]
public string Name { get; private set; }
```

Co nám to přineslo v DotVVM

- DotVVM 4.3.9

Method	Mean	Allocated
InitialGet	9.501 ms	6.88 MB
Postback	45.790 ms	20.93 MB

- DotVVM 5.0.0-preview07-final

Method	Mean	Allocated
InitialGet	3.092 ms	1.97 MB
Postback	7.612 ms	3.58 MB

~380kB viewmodel z reálné aplikace

3x rychlejší serializace

6x rychlejší deserializace

3,5x méně paměti na serializaci

5,5x méně paměti na deserializaci

Co nám to přineslo v DotVVM

- Není to jen změnou knihovny
  - Provedli jsme několik optimalizací algoritmu
  - Přepis stavby JSON DOMu na přímé použití `Utf8JsonWriter`
- Našli jsme bug v knihovně `FastExpressionCompiler`
  - V DotVVM se neprojevoval, targetovali jsme starou verzi
  - V projektu, kde se chyba našla, se používala novější verze
  - https://github.com/dadhi/FastExpressionCompiler/issues/490
- Pak jsme objevili ještě jiný bug, který byl pro změnu náš

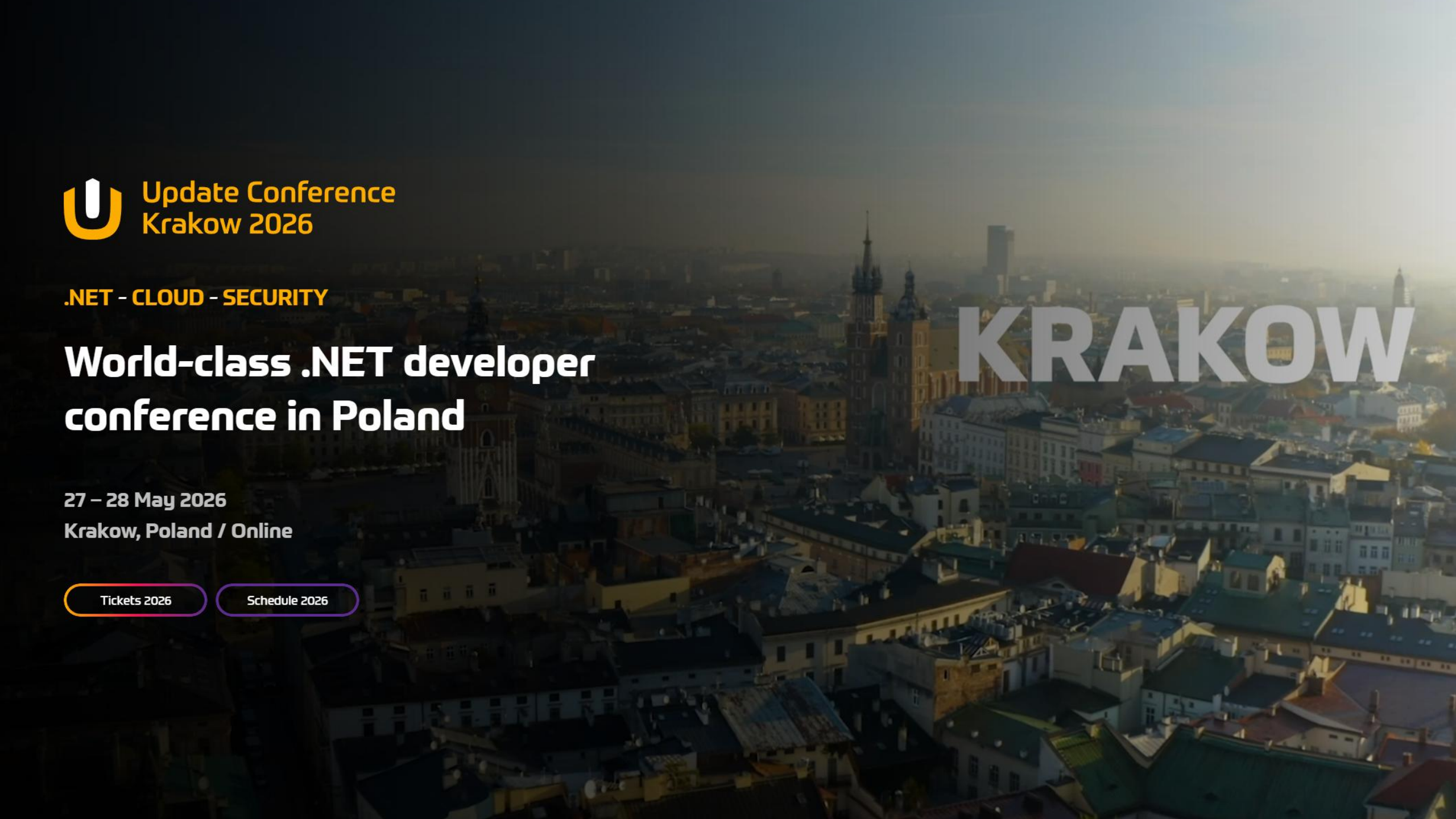
Zajímavosti z kódu

- Přechod ze string na UTF-8
  - **main**:DefaultViewModelSerializer.cs#L396
  - **release/4.3**:DefaultViewModelSerializer.cs#L301
- Serializace pomocí writeru namísto stavby JSON DOM
  - **main**:DefaultViewModelSerializer.cs#L155
  - **release/4.3**:DefaultViewModelSerializer.cs#L104
- Úpravy konverterů
  - **main**:DotvvmByteArrayConverter.cs
  - **release/4.3**:DotvvmByteArrayConverter.cs

Shrnutí

- Pokud pouze **migrujete aplikaci**, většiny rozdílů si nevšimnete
  - Pozor na změny, které nezpůsobí compile-type chyby
  - Polymorfismus

- Pokud **spravujete knihovnu**, je to složitější
  - Více scénářů, které musíte pokrýt
  - Nevíte, jaké funkce používají vaši uživatelé
  - Pomocí konverterů jde dělat hodně magie


Update Conference
Krakow 2026
.NET - CLOUD - SECURITY
KRAKOW
World-class .NET developer conference in Poland
27 – 28 May 2026
Krakow, Poland / Online
Tickets 2026
Schedule 2026

riganti

Q&A

tomasherceg

tomasherceg