Práce s textovými daty

Ing. Michal Radecký, Ph.D.

Regulární výrazy

- Regulární výraz je textový řetězec, který popisuje určitý vzor na základě syntaxe
- Použití regulárních výrazů
  - Ověření, že vstupní text obsahuje nějaký podřetězec (validace)
  - Ověření, že vstupní text odpovídá vzoru (validace)
  - Vyhledání podřetězce (dle vzoru) ve vstupním textu
  - Nahrazení podřetězce (dle vzoru) ve vstupním textu

  - Různá míra přesnosti – odpovídá, neodpovídá, téměř odpovídá (`RegexOptions`)

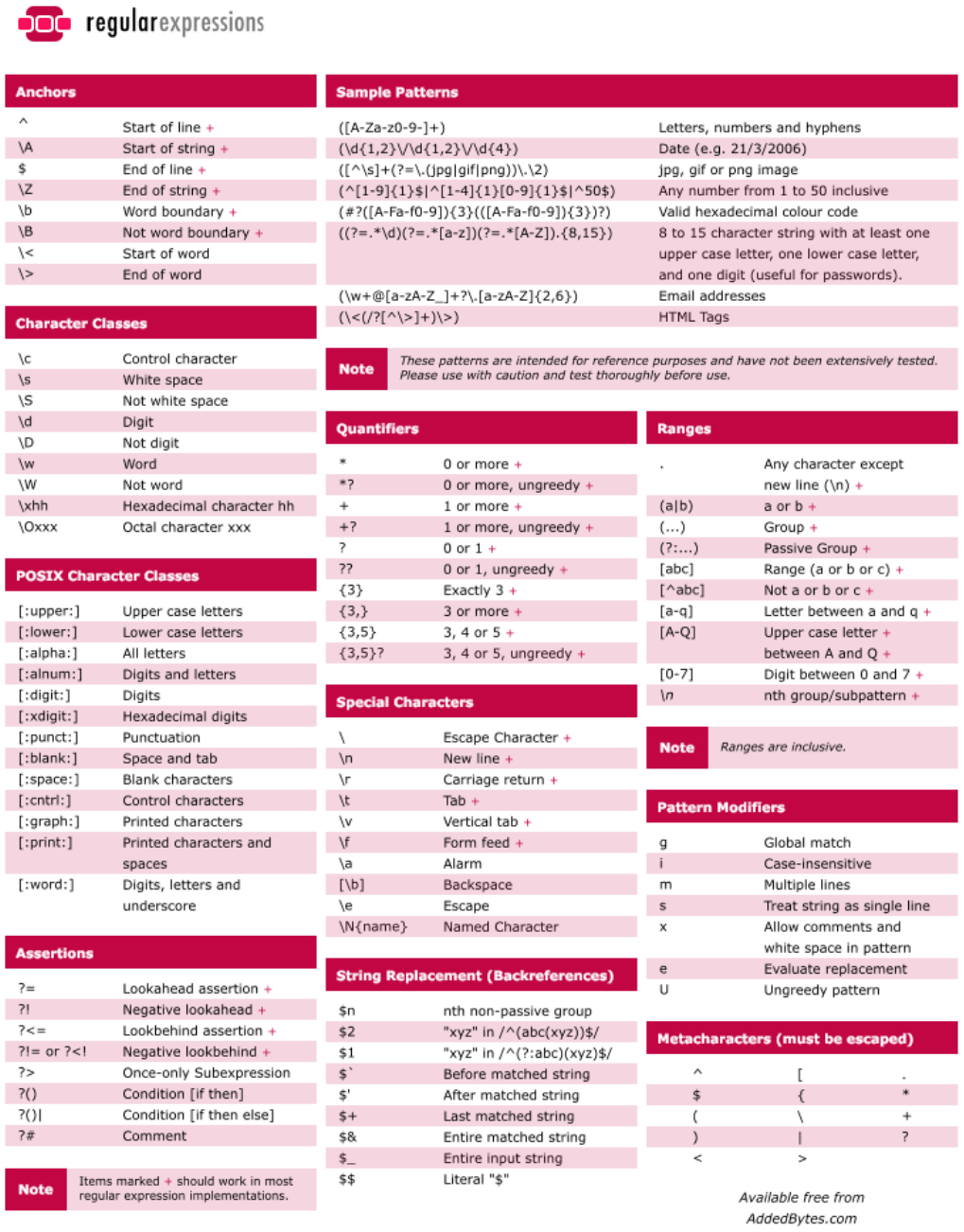
- Implementace pomocí třídy **`Regex`**
  - Některé metody je možné volat jako statické (`IsMatch`, `Match`)

- **`System.Text.RegularExpressions`**
  https://docs.microsoft.com/cs-cz/dotnet/api/system.text.regularexpressions?view=net-5.0

Syntaxe regulárních výrazů

- Syntaxe tvorby regulárních výrazů vychází z obecných pravidel
- Různá implementační prostředí mohou mít určitá specifika, např. třídy znaků, zápis speciálních znaků
- Základem zpracování je konečný (ideálně deterministický) automat
- Standardně se chová „hladově“, tj. snaží se výsledkem pokrýt co nejvíce vstupu, lze ovlivnit kvantifikátory, apod.

- https://docs.microsoft.com/cs-cz/dotnet/standard/base-types/regular-expressions
- https://www.itnetwork.cz/csharp/oop/tutorial-csharp-dot-net-regularni-vyrazy
- https://www.zive.cz/clanky/poznavame-c-a-microsoft-net-31-dil--regularni-vyrazy/sc-3-a-125647/default.aspx



regularexpressions

Anchors

^	Start of line +
\A	Start of string +
$	End of line +
\Z	End of string +
\b	Word boundary +
\B	Not word boundary +
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\xhh	Hexadecimal character hh
\Oxxx	Octal character xxx

POSIX Character Classes

[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions

?=	Lookahead assertion +
?!	Negative lookahead +
?<=	Lookbehind assertion +
?!= or ?<!	Negative lookbehind +
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Note Items marked + should work in most regular expression implementations.

Sample Patterns

([A-Za-z0-9-]+)	Letters, numbers and hyphens
(\d{1,2}\/\d{1,2}\/\d{4})	Date (e.g. 21/3/2006)
([^\s]+(?=\.(jpg|gif|png))\.\2)	jpg, gif or png image
(^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$|^50$)	Any number from 1 to 50 inclusive
(#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?)	Valid hexadecimal colour code
((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15})	8 to 15 character string with at least one upper case letter, one lower case letter, and one digit (useful for passwords).
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Email addresses
(\<(/?[^\>]+)\>)	HTML Tags

Note *These patterns are intended for reference purposes and have not been extensively tested. Please use with caution and test thoroughly before use.*

Quantifiers

*	0 or more +
*?	0 or more, ungreedy +
+	1 or more +
+?	1 or more, ungreedy +
?	0 or 1 +
??	0 or 1, ungreedy +
{3}	Exactly 3 +
{3,}	3 or more +
{3,5}	3, 4 or 5 +
{3,5}?	3, 4 or 5, ungreedy +

Special Characters

\	Escape Character +
\n	New line +
\r	Carriage return +
\t	Tab +
\v	Vertical tab +
\f	Form feed +
\a	Alarm
[\b]	Backspace
\e	Escape
\N{name}	Named Character

String Replacement (Backreferences)

$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string
$_	Entire input string
$$	Literal "$"

Ranges

.	Any character except new line (\n) +
(a|b)	a or b +
(...)	Group +
(?:...)	Passive Group +
[abc]	Range (a or b or c) +
[^abc]	Not a or b or c +
[a-q]	Letter between a and q +
[A-Q]	Upper case letter + between A and Q +
[0-7]	Digit between 0 and 7 +
\n	nth group/subpattern +

Note *Ranges are inclusive.*

Pattern Modifiers

g	Global match
i	Case-insensitive
m	Multiple lines
s	Treat string as single line
x	Allow comments and white space in pattern
e	Evaluate replacement
U	Ungreedy pattern

Metacharacters (must be escaped)

^	[	.
$	{	*
(	\	+
)	|	?
<	>	

Available free from AddedBytes.com

Regex

- **`IsMatch`**
  metoda, která vrací TRUE,
  pokud je vzor nalezen ve vstupu

```
string data = "Hello World!";
if (Regex.IsMatch(data, "Hello"))
{
    System.Console.WriteLine("1: Hello Found, joy!");
}
data = "Goodbye World!";
if (Regex.IsMatch(data, "Hello"))
{
    System.Console.WriteLine("2: Hello Found, joy!");
}
```

- **`Match`**
  metoda a třída umožňující další práci
  s výsledkem vyhledání
- **`Matches`**
  vrací kolekci všech shod hledání

```
string[] id = { "123-45-6789",
     "1234-5-6789",
     "547-12-6346",
     "54-12-5623",
     "3513-15134",
     "608-12-61347",
     "8608-12-6134",
     "608-12-6134" };

 for (int i = 0; i < id.Length; i++)
 {
     if (Regex.IsMatch(id[i], @"\d{3}-\d{2}-\d{4}"))
     {
         System.Console.WriteLine(Regex.Match(id[i], @"\d{3}-\d{2}-\d{4}"));
     }
 }
```

Regex

- **`GroupCollection`**
  V rámci regulárního výrazu je možné určit tzv. skupiny (kulaté závorky) a s těmi následně pracovat

```
string str = "$1.57 $316.15 $19.30 $0.30 $0.00 $41.10 $5.1 $.5";
string strMatch = @"\$(\d+)\.(\d\d)";

for (Match m = Regex.Match(str, strMatch); m.Success; m = m.NextMatch())
{
    GroupCollection gc = m.Groups;

    System.Console.WriteLine("${0}.{1}", int.Parse(gc[1].Value) + 5, gc[2].Value);
}
```

Regex

- **`Replace`**
  metoda umožňující nahradit řetězec odpovídající regulárnímu výrazu jiným řetězcem
  podporuje také `Groups` – je možné je využít jako substituce ve výstupu

```
string str = Regex.Replace("Hello World", "Hello", "Goodbye");
System.Console.WriteLine(str);

string a = "abcd hello ello yellow";
str = Regex.Replace(a, @"\b\w{4}\b", "****");
System.Console.WriteLine(str);

str = "30-50 100-200 123-647 952-142 5-1231";
string search = @"(\d+)-(\d+)";
string replace = "$2-$1";
str = Regex.Replace(str, search, replace);
System.Console.WriteLine(str);
```

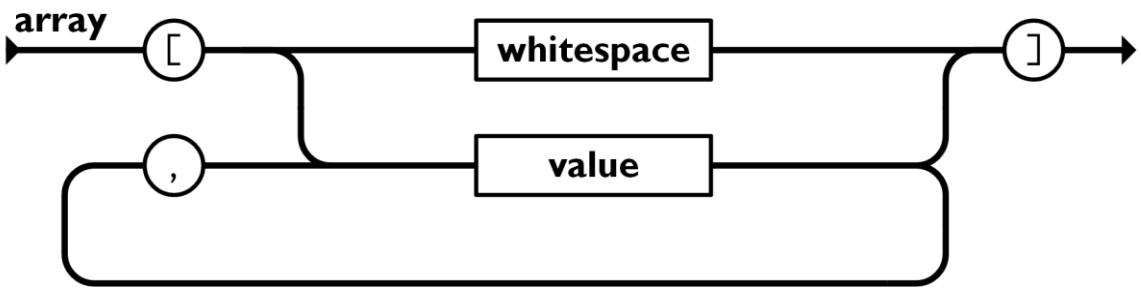
Poznámky

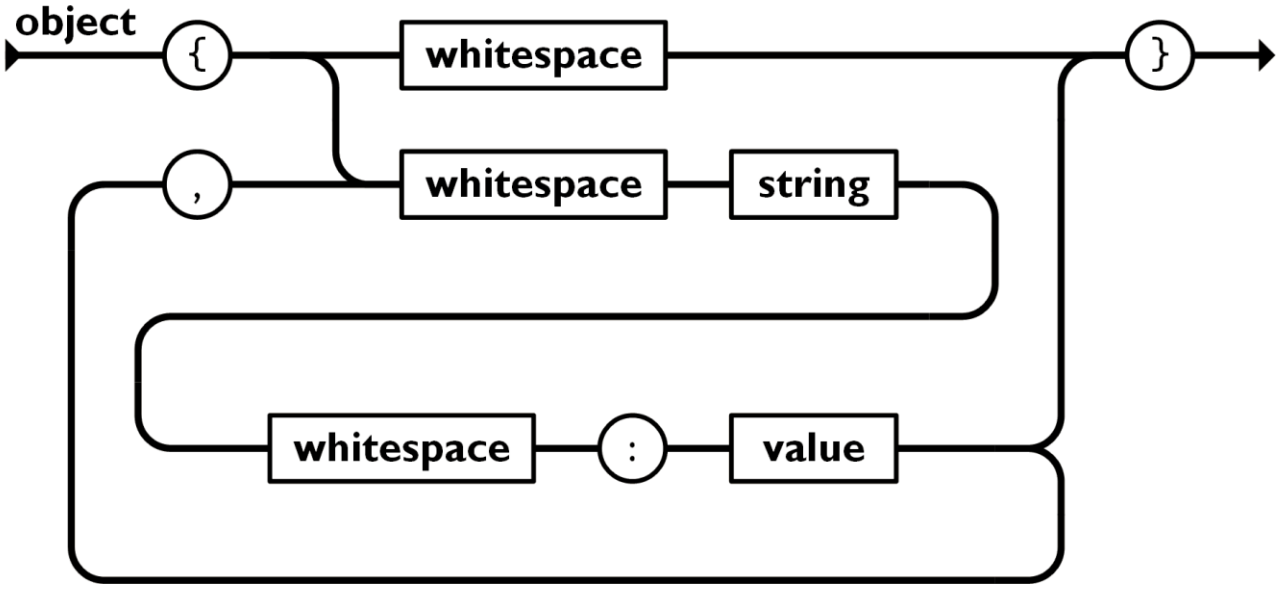
- Zpracování regulárních výrazů může být poměrně náročné, především z pohledu algoritmizace výrazu. Zpracování/kompilace regulárního jazyka je možné třemi způsoby – sekvence vnitřních instrukcí, jazyk CIL, předkompilace. Využívá se mezipaměť zkompilovaných regulárních výrazů.
- Zavináč před stringem znamená, že se nebere ohled na znaky, které by musely být „escapované“

```
string cesta = "C:\\Adresar\\Soubor.txt";
string cesta = @"C:\Adresar\Soubor.txt";
```

JSON

- JavaScript Object Notation
- Způsob textového zápisu objektových dat pro jejich výměnu (specifikace ECMA-404)
- Dva hlavní konstrukční prvky (klíč a hodnota; pole)





- https://www.json.org/json-en.html

Zpracování JSON

- Dříve hojně používaný přístup `Newtonsoft.Json`, nahrazen **`System.Text.Json`** (NuGet package, od .NET 5 nativně)
- Není zpětná kompatibilita, ne vše je podporováno (postupně se vyvíjí)

- Poskytuje výkonné a integrované zpracování JSON dat s důrazem na standardy a bezpečnost.
- Integrovaná podpora v rámci ASP.NET Core (např. JSON response)

Scenario	Speed	Memory
Deserialization	2x faster	Parity or lower
Serialization	1.5x faster	Parity or lower
Document (read-only)	3-5x faster	~Allocation free for sizes < 1 MB
Reader	2-3x faster	~Allocation free (until you materialize values)
Writer	1.3-1.6x faster	~Allocation free

- **`System.Text.Json`**
  https://docs.microsoft.com/en-us/dotnet/api/system.text.json?view=net-5.0

JsonSerializer

- Poskytuje funkcionalitu přímé serializace a deserializace na úrovni tzv. POCO (Plain Old CLR Object)
- Proces je možné ovlivnit pomocí JsonSerializerOptions
- Možné serializovat/deserializovat se znakovou sadou UTF8 – pracuje s byty namísto řetězci, cca 5-10% rychlejší
- Podporuje také async metody (`SerializeAsync`, `DeserializeAsync`)
- Custom atributy u properties pro ovlivnění serializace/deserializace

JsonSerializer

```
string SerializePrettyPrint(WeatherForecast value)
    {
        var options = new JsonSerializerOptions
        {
            WriteIndented = true
        };

        return JsonSerializer.Serialize<WeatherForecast>(value, options);
    }
```

```
 WeatherForecast Deserialize(string json)
     {
         var options = new JsonSerializerOptions
         {
             AllowTrailingCommas = true
         };

         return JsonSerializer.Deserialize<WeatherForecast>(json, options);
     }
```

```
class WeatherForecast
{
    public DateTimeOffset Date { get; set; }

    // Always in Celsius.
    [JsonPropertyName("temp")]
    public int TemperatureC { get; set; }

    public string Summary { get; set; }

    // Don't serialize this property.
    [JsonIgnore]
    public bool IsHot => TemperatureC >= 30;
}
```

```
[
    {
        "date": "2013-01-07T00:00:00Z",
        "temp": 23,
    },
    {
        "date": "2013-01-08T00:00:00Z",
        "temp": 28,
    },
    {
        "date": "2013-01-14T00:00:00Z",
        "temp": 8,
    },
]
```

JsonDocument

- Umožňuje přístup ke strukturovaným datům bez potřeby plné serializace/deserizace
- Založeno na principu DOMu (Document Object Model)

- **`Parse`**
  metoda, která s textového formátu JSONu vytvoří DOM strukturu
- **`RootElement`**
  property dokumentu, která tvoří vstupní bod do struktury DOM
- **`WriteTo`**
  metoda, která z DOM struktury uloží JSON do `Utf8JsonWriter` (Stream)

JsonDocument

```
double ComputeAverageTemperatures(string json)
    {
        var options = new JsonDocumentOptions
        {
            AllowTrailingCommas = true
        };

        using (JsonDocument document = JsonDocument.Parse(json, options))
        {
            int sumOfAllTemperatures = 0;
            int count = 0;

            foreach (JsonElement element in document.RootElement.EnumerateArray())
            {
                DateTimeOffset date = element.GetProperty("date").GetDateTimeOffset();

                if (date.DayOfWeek == DayOfWeek.Monday)
                {
                    int temp = element.GetProperty("temp").GetInt32();
                    sumOfAllTemperatures += temp;
                    count++;
                }
            }

            var averageTemp = (double)sumOfAllTemperatures / count;
            return averageTemp;
        }
    }
```

Utf8JsonReader, Utf8JsonWriter

- „Vnitřní“ metody používané dříve zmíněnými přístupy
- Umožňují přímé čtení/tvorbu obsah JSON textu na nižší úrovni abstrakce
- Postaveno na principu sekvenčního čtení/zapisování jednotlivých elementů JSON struktury
  - Čtení – postupně se čtou prvky a je potřeba je zpracovávat dle typu (tokeny)
  - Zápis – postupně se vytváří každý prvek struktury
- Vhodné ve chvíli, kdy chceme mít plnou kontrolu nad procesem čtení/zápisu JSON struktury

Utf8JsonReader, Utf8JsonWriter

```
byte[] data = Encoding.UTF8.GetBytes(json);
        Utf8JsonReader reader = new Utf8JsonReader(data,
           isFinalBlock: true, state: default);
        while (reader.Read())
        {
            Console.Write(reader.TokenType);
            switch (reader.TokenType)
            {
                case JsonTokenType.PropertyName:
                case JsonTokenType.String:
                    {
                        string text = reader.GetString();
                        Console.Write(" ");
                        Console.Write(text);
                        break;
                    }
                case JsonTokenType.Number:
                    {
                        int value = reader.GetInt32();
                        Console.Write(" ");
                        Console.Write(value);
                        break;
                    }
            }
            Console.WriteLine();
        }
```

```
var options = new JsonWriterOptions
        {
            Indented = true
        };
        using (var stream = new MemoryStream())
        {
            using (var writer = new Utf8JsonWriter(stream, options))
            {
                writer.WriteStartObject();
                writer.WriteNumber("StudentID", 123);
                writer.WriteString("StudentNames", „Pepa");
                writer.WriteString("StudentCourseCode", "P101");
                writer.WriteEndObject();
            }
            string json = Encoding.UTF8.GetString(stream.ToArray());
            Console.WriteLine(json);
        }
```

XML

- Extensible Markup Language
- Značkovací jazyk určený pro zápis strukturovaných dat
- Založený na principech SGML
- Struktura dat založená na **zanořování**, **elementech** a jejich **obsahu** resp. **atributech**
- Starší přístup než JSON, složitější struktura a specifikace
- Možnosti sofistikovaných nástrojů pro specifikaci a validaci (DTD, XML Schéma), podpora namespace
- Nástroj pro dotazování Xpath, pro transformaci XSLT
- **`XDocument` alternativa, obdobné použití - součást LINQ**

- **`System.Xml`**
  https://docs.microsoft.com/en-us/dotnet/api/system.xml?view=net-5.0

Práce s XML

- Principiálně zpracování podobné jako zpracování JSON
- **`XMLSerializer`**
  - Třída poskytující přímou serializaci/deserializaci mezi XML a objektem
  - Hojně se využívají custom atributy pro mapování
- **`XMLDocument`**
  - Třída poskytující DOM přístup k obsahu XML dokumentu
  - Klíčové metody: `Load`, `LoadXml`, `WriteTo`, `Validate`
  - Třídy pokrývající strukturu: `XmlNode`, `XmlElement`, `XmlAttribute`

XmlDocument

```
XmlDocument doc = new XmlDocument();
doc.Load("library.xml");
XmlNode rootNode = doc.DocumentElement;
Console.WriteLine("Root node: {0}", rootNode.Name);
foreach (XmlAttribute atr in rootNode.Attributes)
{
    Console.WriteLine("Attribute: {0}={1}", atr.Name, atr.Value);
}
foreach (XmlNode node in rootNode.ChildNodes)
{
    Console.WriteLine("\nBook title = {0}", node["title"].InnerText);
    Console.WriteLine("Book author = {0}", node["author"].InnerText);
    Console.WriteLine("Book isbn = {0}", node["isbn"].InnerText);
}
```

XmlReader, XmlWriter

- Přístup založený na principu SAX (Simple API for XML) – event-driven proces
- Postupné načítání/zapisování prvků XML na nízké implementační úrovni

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.IgnoreWhitespace = true;
using (XmlReader reader = XmlReader.Create("customer.xml", settings))
while (reader.Read())
{
    Console.Write(new string(' ', reader.Depth * 2)); // Write indentation
    Console.WriteLine(reader.NodeType);
}
```

```
XmlWriterSettings settings = new XmlWriterSettings();
settings.Indent = true;
using (XmlWriter writer = XmlWriter.Create("..\\..\\foo.xml", settings))
{
    writer.WriteStartElement("customer");
    writer.WriteElementString("firstname", "Jim");
    writer.WriteElementString("lastname", " Bo");
    writer.WriteEndElement();
}
```

XPath

- Dotazovací jazyk pro XML dokumenty
- Mimo dotazování umí i realizovat základní operace a výpočty s daty
- Multiplatformní využití nezávislé na implementaci (až na případné drobnosti)
- https://www.w3schools.com/xml/xpath_intro.asp
- https://www.youtube.com/watch?v=EfPvYZhLdKc

- **`SelectSingleNode`**, **`SelectNodes`**
  metody nad `XmlDocument` a `XmlNode`, které vracejí `XmlNode` resp. `XmlNodeList`

XPath

```
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load("../../items.xml");
string xPathQuery = "/items/item[@type='beer']";
XmlNodeList beersList = xmlDoc.SelectNodes(xPathQuery);
foreach (XmlNode beerNode in beersList)
{
    XmlNode beerName = beerNode.SelectSingleNode("name");
    Console.WriteLine(beerName.InnerText);
}
```

Další zdroje

- https://www.c-sharpcorner.com/article/c-sharp-regex-examples/
- https://devblogs.microsoft.com/dotnet/whats-next-for-system-text-json/
- https://devblogs.microsoft.com/dotnet/try-the-new-system-text-json-apis/
- https://jonathancrozier.com/blog/xml-serialization-with-c-sharp