Regular Expressions

Umsetzung in Programmiersprachen

C#

C# – Klassenhierarchie

Namespace **System.Text.RegularExpressions**

Capture	Represents the results from a single successful subexpression capture.
CaptureCollection	Represents the set of captures made by a single capturing group.
Group	Represents the results from a single capturing group.
GroupCollection	Returns the set of captured groups in a single match.
Match	Represents the results from a single regular expression match.
MatchCollection	Represents the set of successful matches found by iteratively applying a regular expression pattern to the input string.
Regex	Represents an immutable regular expression.To browse the .NET Framework source code for this type, see the Reference Source.

System.Object
  System.Text.RegularExpressions.Capture
    System.Text.RegularExpressions.Group
      System.Text.RegularExpressions.Match

C# – Klasse Regex

- Namespace **System.Text.RegularExpressions**
- Statische- oder Instanz-Methoden
- Optionen wie Singleline werden im Konstruktor mitgegeben
- Beispiel:
  - `bool ok = Regex.IsMatch(input, pattern);`
  - `Regex r = new Regex(pattern, RegexOptions.Singleline);`
    `bool ok = r.IsMatch(input);`
- Weitere Methoden
  - `Match Match(string input)`
  - `MatchCollection Matches(string input)`
  - `String Replace(string input, string pattern, string replace)`

```
var regex = new Regex(@"\d\d-\d\d-\d{4}");
bool isOk = regex.IsMatch("03-12-2020");
var match = regex.Match("03-12-2020");
```

```
Match match = Regex.Match(input, pattern);
```

C# – Klasse Match

- **`bool Success`**: war Match erfolgreich?
- **`string Value`**: Text des Treffers
- **`int Length`**: Anzahl der Zeichen im erkannten Text
- **`GroupCollection Groups`**: Liste der Gruppen im Match

- Achtung: **`Groups[0]`** liefert den Text des gesamten Treffers

```
Match m = Regex.Match("Hans Huber", @"^(?<first>\w*)\s+(?<last>\w*)$");
if (m.Success)
{
  int len = m.Length;
  string name = m.Value;
  Group last = m.Groups["last"];
  Group first = m.Groups["first"];
}
```

len	10
name	"Hans Huber"
last	{Huber}
first	{Hans}

C# – Klasse Group

```
Match match = Regex.Match(input, pattern);
Group group = match.Groups[0];
```

- Erhält man bei Iteration über **`GroupCollection`**.
- Wenn in einem Match mehrere Gruppen vorkommen
- Properties (analog Klasse Match, da Basisklasse):
  - **`bool Success`**: wurde die Gruppe gefunden?
  - **`int Index`**: Position im String
  - **`int Length`**: Länge des Substrings
  - **`string Value`**: erkannter Text
  - **`CaptureCollection Captures`**: Alle Treffer der Gruppe
    - wenn eine Gruppe mehrmals vorkommt: …(…(\d)*…)…

```
Match m = Regex.Match("Hans Huber", @"^(?<first>\w*)\s+(?<last>\w*)$");
if (m.Success)
{
  Group last = m.Groups["last"];
  int index = last.Index;
  int length = last.Length;
  string lastName = last.Value;
  var captures = last.Captures;
  var capture = captures[0];
}
```

last	{Huber}
index	5
length	5
lastName	"Huber"
captures	{System.Text.RegularExpressions.CaptureCollection}
capture	{Huber}

C# – Beispiele

- Treffer finden:

```
Regex regex = new Regex(@"M[ae][yi]e?r");
string input = "Auer Berger Maier Müller Mayr Humer Meier Lehner";
MatchCollection matches = regex.Matches(input);
foreach (Match match in matches)
{
  Console.WriteLine(match.Value);
}
```

- Gruppen finden:

```
Match m = Regex.Match("Hans Max Huber", @"^(?<first>\w*)\s+(?<middle>\w*)\s+(?<last>\w*)$");
if (m.Success)
{
  string sFirst = m.Groups["first"].Value;
  string sMiddle = m.Groups["middle"].Value;
  string sLast = m.Groups["last"].Value;
  Console.WriteLine($"{sLast}, {sFirst} {sMiddle.ToUpper().First()}.");
}
```

- GroupCollection:

```
Match m = Regex.Match("Hans Huber", @"^(?<first>\w*)\s+(?<last>\w*)$");
if (m.Success)
{
  GroupCollection groups = m.Groups;
  foreach (Group g in groups)
  {
    Console.WriteLine($"Pos/Len {g.Index}/{g.Length} --> {g.Value}");
  }
}
```

```
Pos/Len 0/10 --> Hans Huber
Pos/Len 0/4 --> Hans
Pos/Len 5/5 --> Huber
```

C# – MatchEvaluator

- Text ersetzen
- Ersetzungstext ergibt sich aus Match
- public delegate string MatchEvaluator(Match match)
- Beispiel:

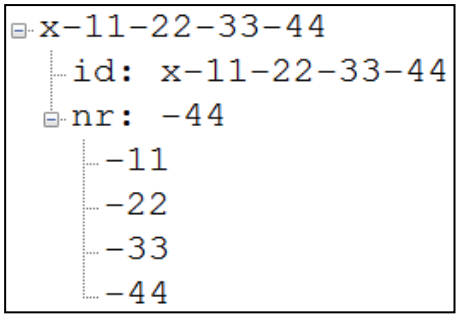
```
string s = Regex.Replace("abbbcddddddeefgggggggh", @"(\w)\1{2,}", match =>
    {
      string sHit = match.Groups[0].Value;
      return $"{sHit[0]}{sHit.Length}";
    });
```

- Ergibt: ab3cd6eefg7h
- Für LINQ-Freunde

```
string s = Regex.Replace(input, @"(\w)\1{2,}",
  x =>x.Groups.Cast<Group>()
  .Select(x => x.Value)
  .Select(x => $"{x[0]}{x.Length}")
  .First()
);
```

C# – Beispiel Captures

- Wenn **Gruppe** in Suchtext **mehrmals** vorkommt
- Suchtext: **`x-11-22-33-44`**
- Regex: **`[a-z](?<nr>-\d\d)*`**
- Expresso:



- C#:

```
var regex = new Regex(@"[a-z](?:-(?<nr>\d\d))*");
Match m = regex.Match("x-11-22-33-44");
if (!m.Success) return;
Group nrs = m.Groups["nr"];
foreach (Capture capture in nrs.Captures)
{
  Console.WriteLine($"  nr {capture:00} at {capture.Index:00}");
}
```

```
nr 11 at 2
nr 22 at 5
nr 33 at 8
nr 44 at 11
```

C#

Entities

C# – Entities

- Columns/Properties können mit RegEx validiert werden
- Annotation **`RegularExpressions`**
- Namespace **`System.ComponentModel.DataAnnotations`**
- Syntax:

```
[RegularExpression("pattern", ErrorMessage = "…")]
```

- Beispiel:

```
public class RegexData
{
  public int Id { get; set; }

  [Required]
  [RegularExpression(@"^[A-Z]{2}$",
    ErrorMessage = "Country code can only be two alphabetic characters in CAPITALS")]
  public string CountryCode { get; set; }

  [RegularExpression(@"^\w*(\.\w+)*@\w*(\.\w+)+$",
    ErrorMessage = "Email hast to look like x@x.x")]
  public string Email { get; set; }

  public override string ToString() => $"CountryCode: {CountryCode}, Email: {Email}";
}
```

C# – Ajax

- Der Modelstate wird automatisch überprüft (ab Net Core 2.1)

```
Send Request
POST {{regex}} HTTP/1.1
Content-Type: application/json

{
 "countryCode" : "ABc",
 "email": "hansi.huber@com"
}
```

```
"type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
"title": "One or more validation errors occurred.",
"status": 400,
"traceId": "|1657a9f1-4e5979ec6f17775a.",
"errors": {
  "Email": [
    "Email hast to look like x@x.x"
  ],
  "CountryCode": [
    "Country code can only be two alphabetic characters in CAPITALS"
  ]
}
```

C# – EF Core

- Man kann bei Speichern in Datenbank auch prüfen lassen
- ➔ Methode SaveChanges überschreiben
- Exception **ValidationException**

```
public override int SaveChanges()
{
  var entities = ChangeTracker.Entries()
                 .Where(x => x.State == EntityState.Added || x.State == EntityState.Modified)
                 .Select(x => x.Entity);
  foreach (var entity in entities)
  {
    System.Console.WriteLine($"  SaveChanges: Validating {entity}");
    var validationContext = new ValidationContext(entity);
    Validator.ValidateObject(entity, validationContext, true);
  }
  return base.SaveChanges();
}
```

Javascript

Javascript – /…/ bzw. RegExp

- Zwei Möglichkeiten, um Regex zu erzeugen
  - `var regex = /…/;`
  - `var regex = new RegExp('…');`
- Achtung: bei /…/ keine Anführungszeichen!
- Optionen wie Multiline werden so angegeben:
  - `var regex = /…/mi;`
  - **m** für Multiline
  - **i** für Ignore Case
  - **g** für Global, d.h. es sollen alle Treffer gefunden werden
- Entsprechen folgenden Properties:
  - regex.**multiline**
  - regex.**ignoreCase**
  - regex.**global**

Javascript – mit String 1/3

- **search**(/…/)
  - Position des ersten Treffers
  - -1 bei keinem Treffer
  - Global Flag „g“ wird nicht unterstützt
  - Bsp: `'Barbapapa'.search(/pa/); //returns 5`

- **split**(/…/)
  - Trennt String an den Treffern
  - Retourniert Array mit Teilstrings
  - Bsp: `'1 , 2,3'.split(/\s*,\s*/);`
  - Ergibt: `[ '1', '2', '3' ]`
  - Bzw.: `'1 , 2,3'.split(/\s*,\s*/).map(x=>parseInt(x));`

Javascript – mit String 2/3

- **replace**(/…/,'…')
  - Ersetzt gefundene Strings durch den String im 2. Parameter
  - Global Flag „g“ ➔ ersetzt alle Treffer
  - Bsp: `'Barbapapa'.replace(/ba/ig,'Ki');` //→KirKipapa

- **replace**(/…/, **function**(){…})
  - Für jeden Treffer wird die Funktion aufgerufen
  - Bsp: `'12 65 78 23'.replace(/\d\d/g, s => parseInt(s) < 30 ? '--' : s);`
  - ergibt: `'-- 65 78 --'`

Javascript – mit String 3/3

- **match**(/…/)
  - Häufigste Variante
  - Unterschiedliche Rückgabewerte mit oder ohne Flag „g“
  - Bei keinem Treffer: retourniert **null** (in beiden Fällen)

- **match**(/…/) ohne **g**
  - Retourniert Treffer als Array
  - Zusätzlich Properties index u. input

```
> var res = 'x1x2x3'.match(/\d/);
  undefined
> res
  [ "1" ]
> res[0]
  "1"
> res.index
  1
> res.input
  "x1x2x3"
```

- **match**(/…/g) mit **g**
  - Retourniert Array mit allen Treffern

```
> var res = 'x1x2x3'.match(/\d/g);
  undefined
> res
  [ "1", "2", "3" ]
> res[1]
  "2"
> res.input
  undefined
```

Javascript – mit RegExp

- **exec('…')**:
  - Funktioniert praktisch wie match von String

```
>> /\d/.exec('x1x2x3')
<- Array [ "1" ]
```

```
>> /\d/.exec('x1x2x3')
<- (1) […]
    0: "1"
    index: 1
    input: "x1x2x3"
    length: 1
```

  - Unterschied bei Global-Flag
  - ➔ man braucht **Schleife**

```
const pattern = /\d/g;
while (res = pattern.exec('x1x2x3')) {
  console.log(`Match ${res[0]} at pos ${res.index}`);
}
```

```
Match 1 at pos 1
Match 2 at pos 3
Match 3 at pos 5
```

- **test('…')**:
  - Retourniert **true**, falls mindestens ein match
  - Entspricht also: **exec('…')!=null**

Javascript- Beispiele

- Treffer finden:

```
const pattern = /M[ae][yi]e?r/g;
const input = 'Auer Berger Maier Müller Mayr Humer Meier Lehner';
const matches = input.match(pattern);
for (const m of matches) console.log(m);
```

```
D:\Node\Regex>node regexdemo.js
Maier
Mayr
Meier
```

- Gruppen finden:

```
const matches = 'Hans Max Huber'.match(/^(\w*)\s+(\w*)\s+(\w*)$/);
console.log(matches);
const sFirst = matches[1];
const sMiddle = matches[2];
const sLast = matches[3];
console.log(`${sLast}, ${sFirst} ${sMiddle.toUpperCase()[0]}.`);
```

```
D:\Node\Regex>node regexdemo.js
Huber, Hans M.
```

- Mit Destructuring:

```
const [, sFirst, sMiddle, sLast] = 'Hans Max Huber'.match(/^(\w*)\s+(\w*)\s+(\w*)$/);
console.log(`${sLast}, ${sFirst} ${sMiddle.toUpperCase()[0]}.`);
```