Focused forward

Netcracker

Лекция 5.

Регулярные выражения в JS (RegExp)

Web System Design and Development

План лекции

- Что такое регулярные выражения?
- Синтаксис регулярных выражений JavaScript.
- Спецсимволы.
- Примеры.
- Методы JavaScript для работы с RegExp.
- Рекомендации.

Что такое регулярные выражения?

Регулярные выражения (regular expressions)
— формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов.

Регулярное выражение - это **строка-образец (pattern)**, состоящая из символов и метасимволов и задающая правило поиска.

Большинство современных языков программирования имеет встроенную поддержку регулярных выражений.

Perl **Python** **Ruby** **JavaScript** **Java** **PHP** **HTML5** **Delphi** **и другие….**

Что такое регулярные выражения?

Общая задача регулярных выражений
- находить или не находить совпадения строки или ее части с шаблоном.

Синтаксис регулярных выражений JavaScript

Формы записи

1

```
var expr = new RegExp(pattern[,flags]);
```

2

```
var expr = /pattern/flags;
```

pattern	Шаблон поиска (текст регулярного выражения).	**flags**	Способы поиска по шаблону.

Синтаксис регулярных выражений JavaScript

1-я форма записи:

Использование конструктора влечет за собой **компиляцию** регулярного выражения **во время исполнения скрипта**. Используйте данный способ, если знаете, что выражение будет изменяться или не знаете шаблон заранее. Например вы получаете его из стороннего источника, при пользовательском вводе.

2-я форма записи:

Литералы регулярных выражений вызывают **предварительную компиляцию** регулярного выражения **при анализе скрипта**. Если ваше регулярное выражение постоянно, то пользуйтесь ним, чтобы увеличить производительность.

Синтаксис регулярных выражений JavaScript

flags

`g`	глобальный поиск (обрабатываются все совпадения с шаблоном поиска);
`i`	не различать строчные и заглавные буквы;
`m`	многострочный поиск.

Порядок указания флагов не имеет значения.

Синтаксис регулярных выражений JavaScript

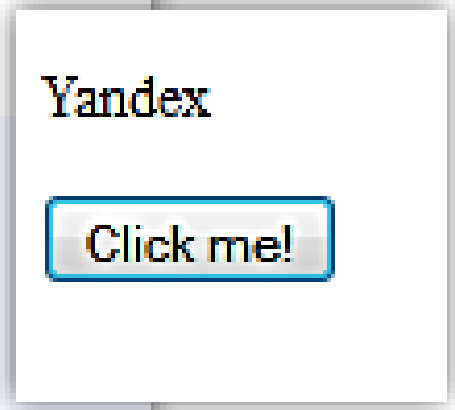
Пример поиска без учета регистра

```
<body>
<p id="demo">Нажмите на кнопку, чтобы осуществить
не чувствительный к регистру поиск</p>

<button onclick="myFunction()">Click me!</button>

<script>
function myFunction()
{
var str = "Visit Yandex";
var patt1 = /yandex/i;
var result = str.match(patt1);
document.getElementById("demo").innerHTML=result;
}
</script>

</body>
```



Синтаксис регулярных выражений JavaScript

Пример глобального поиска

```
var str="Is this all there is?";
        var patt1=/is/g;
```

Is this all there is?

Пример глобального поиска без учета регистра

```
var str="Is this all there is?";
        var patt1=/is/gi;
```

Is this all there is?

Спецсимволы

Ёжик вышел из уб**еж**ища под **ёл**кой к **дж**ипу "Бер**ёз**ка".

Та**ёж**ник записал: "01.20.2000 года был **ёж**. Назвал "Сер**ёж**кой".

```
<div id="replace">
  <p><b>Ёж</b>ик вышел из уб<b>еж</b>ища под <b>ёл</b>кой к <b>дж</b>ипу
  "Бер<b>ёз</b>ка".
  <p>Та<b>ёж</b>ник записал: "01.20.2000 года был <b>ёж</b>.
  Назвал "Сер<b>ёж</b>кой".
</div>
```

\ - экранирует символы

```
( document.getElementById('replace').innerHTML.replace(/\./gi,
'&#10084document.write;') );
```

Ёжик вышел из уб**еж**ища под **ёл**кой к **дж**ипу "Бер**ёз**ка"❤
Та**ёж**ник записал: "01❤20❤2000 года был **ёж**❤ Назвал "Сер**ёж**кой"❤

Спецсимволы

Без указания спецсимвола

- первый элемент в указанном регистре

```
document.write(
document.getElementById('replace').innerHTML.replace(/ёж/,
'&#10084;') );
```

Ёжик вышел из уб**еж**ища под **ёл**кой к **дж**ипу "Бер**ёз**ка".
Та❤ник записал: "01.20.2000 года был **ёж**. Назвал "Сер**ёж**кой".

Спецсимволы. Позиционирование

$

Последний элемент родителя - обозначает конец входных данных. Если установлен флаг многострочного поиска, то также сработает в конце строки.

`/t$/` не найдет **'t'** в **"eater"**,
но найдет - в **"eat"**.

^

Первый элемент родителя - обозначает начало входных данных. Если установлен флаг многострочного поиска ("m"), то также сработает в начале новой строки.

`/^A/` не найдет **'A'** в **"an A"**,
но найдет первое **'A'** в **"An A"**.

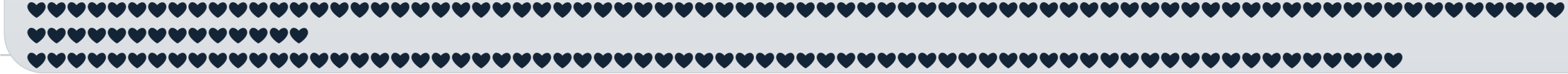
Спецсимволы. Символьные классы

`[\s\S]` — любой символ

```
document.write(
document.getElementById('replace').innerHTML.replace(/[\s\S]/gi,
'&#10084;') );
```

`.` — любой символ, кроме перевода строки

```
document.write(
document.getElementById('replace').innerHTML.replace(/./gi,
'&#10084;') );
```



Спецсимволы. Символьные классы.

\n — перевод строки

```
document.write(
document.getElementById('replace').innerHTML.replace(/\n/gi,'&#10084;'));
```

♥
Ёжик вышел из уб**еж**ища под **ёл**кой к **дж**ипу "Бер**ёз**ка".♥
Та**ёж**ник записал: "01.20.2000 года был **ёж**. Назвал "Сер**ёж**кой".♥

\s — любой пробельный символ, в том числе перевод строк

```
document.write(
document.getElementById('replace').innerHTML.replace(/\s/gi,'&#10084;'));
```

♥♥♥
Ёжик♥вышел♥из♥уб**еж**ища♥под♥**ёл**кой♥к♥**дж**ипу♥"Бер**ёз**ка".♥♥♥
Та**ёж**ник♥записал:♥"01.20.2000♥года♥был♥**ёж**.♥Назвал♥"Сер**ёж**кой".♥

Спецсимволы. Символьные классы.

\s — Любой символ, кроме пробела и перевода строки

```
document.write(
document.getElementById('replace').innerHTML.replace(/\S/gi,'&#10084;'));
♥♥♥♥♥♥♥♥♥♥♥♥♥♥ ♥♥♥♥♥ ♥♥ ♥♥♥♥♥♥♥♥♥♥♥♥♥♥ ♥♥♥ ♥♥♥♥♥♥♥♥♥♥♥♥♥ ♥
♥♥♥♥♥♥♥♥♥♥♥♥ ♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥ ♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥ ♥♥♥♥♥♥♥
♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥ ♥♥♥♥ ♥♥♥ ♥♥♥♥♥♥♥♥♥♥ ♥♥♥♥♥♥
♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥ ♥♥♥♥
```

\t — Символ табуляции

\b — Соответствует несловообразующей границе. Начало и конец строки считаются несловообразующими символами.

/\bn\w/ найдет 'no' в "noonday";
/\wy\b/ найдет 'ly' в "possibly yesterday."

Спецсимволы. Символьные классы.

\d — только цифры, эквивалент [0-9]

```
document.write(document.getElementById('replace').innerHTML.replace(
/\d/gi, '&#10084;') );
```

Ёж**ик вышел из убе**ж**ища под **ёл**кой к **дж**ипу "Берё**з**ка".
Таё**ж**ник записал: "❤❤.❤❤.❤❤❤❤ года был **ёж**. Назвал "Серё**ж**кой".

\D — не цифра, эквивалент [^0-9]

```
document.write(document.getElementById('replace').innerHTML.replace(
/\D/gi, '&#10084;') );
```

❤❤❤
❤❤❤
❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤01❤20❤2000❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤
❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤

Спецсимволы. Символьные классы.

\w — любой цифробуквенный символ, включая нижнее подчеркивание. Эквивалентен [A-Za-z0-9_].

/\w/ совпадает с 'a' в "apple,"
'5' в "$5.28," и '3' в "3D."

\W — соответствует любому не цифробуквенному символу. Эквивалентен [^A-Za-z0-9_].

/\W/ or **/[^A-Za-z0-9_]/** совпадает с '%' в "50%."

Спецсимволы. Символьные классы.

[xyz]

Только перечисленные в квадратных скобках символы.

- Точку в квадратных скобках экранировать не нужно, то есть не **[\.]**, а **[.]**. Зато нужно экранировать тире, то есть не **[-]**, а **[\-]**.
- Через дефис указывается диапазон значений, например,

[0-9] — это тоже самое, что и **[0123456789]**,
[а-я] — все кириллические строчные буквы,
[А-Я] — все кириллические заглавные буквы

```
document.write(document.getElementById('replace').innerHTML.replace(
/[eё]/gi, '&#10084;') );
```

❤**ж**ик выш❤л из уб❤**ж**ища под ❤**л**кой к **дж**ипу "Б❤р❤**з**ка".
Та❤**ж**ник записал: "01.20.2000 года был ❤**ж**. Назвал "С❤р❤**ж**кой".

Спецсимволы. Символьные классы.

[^xyz]

Любой символ, кроме перечисленных в квадратных скобках. Не путать с ^

```
document.write(
document.getElementById('replace').innerHTML.replace(/[^eё]/gi,
'&#10084;') );
```

❤❤❤❤❤❤❤❤❤Ё❤❤❤❤❤❤❤❤❤❤❤e❤❤❤❤❤❤❤❤❤❤e❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤ё❤❤
❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤e❤❤❤❤ё❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤
❤❤❤ё❤❤❤
❤ё❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤e❤❤❤❤ё❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤

Спецсимволы. Повторение.

Повторение

x{n} — n повторений x

```
document.write( 'ëëëëëëëë ëëëëëë ëëëëë ëëëë ëëë ëë ë
'.replace(/ë{3}/g, '&#10084;') );
```

♥♥ë ♥♥ ♥ëë ♥ë ♥ ëë ë

x{n,} — n и более повторений x

```
document.write( 'ëëëëëëëë ëëëëëë ëëëëë ëëëë ëëë ëë ë
'.replace(/ë{3,}/g, '&#10084;') );
```

♥ ♥ ♥ ♥ ♥ ëë ë

Спецсимволы. Повторение.

`x{n,k}` — с n до k повторений x

```
document.write( 'ëëëëëëëë ëëëëëëë ëëëëëë ëëëëë ëëëë ëëë ë
'.replace(/ë{3,5}/g, '&#10084;') );
```

♥ëë ♥ë ♥ ♥ ♥ ëë ë

`x?` — повторение x ноль или один раз. Эквивалент x{0,1}.

```
document.write( 'ëëëëëëëë ëëëëëëë ëëëëëë ëëëëë ëëëë ëëë ë
'.replace(/ë?/g, '&#10084;') );
```

♥♥♥♥♥♥♥♥ ♥♥♥♥♥♥♥ ♥♥♥♥♥♥ ♥♥♥♥♥ ♥♥♥♥ ♥♥♥ ♥♥ ♥ ♥

Спецсимволы. Повторение.

x* — повторение x ноль или более раз. Эквивалент x{0,}.

```
document.write( 'ëëëëëëëë ëëëëëëë ëëëëë ëëëë ëëë ëë ë
'.replace(/ë*/g, '&#10084;') );
```

♥♥ ♥♥ ♥♥ ♥♥ ♥♥ ♥♥ ♥♥ ♥ ♥

x+ — повторение x один и более раз. Эквивалент x{1,}.

```
document.write( 'ëëëëëëëë ëëëëëëë ëëëëë ëëëë ëëë ëë ë
'.replace(/ë+/g, '&#10084;') );
```

♥ ♥ ♥ ♥ ♥ ♥ ♥

Спецсимволы. Чередование и группирование.

Чередование и группирование

()

группировка и создание шаблона. Круглые скобки могут быть вложенными.
(x) - находит x и запоминает. Это называется "запоминающие скобки". Например, /(foo)/ найдет и запомнит 'foo' в "foo bar."
Найденная подстрока хранится в массиве-результате поиска или в предопределенных свойствах объекта RegExp: $1, ..., $9.
Кроме того, скобки объединяют то, что в них находится, в единый элемент паттерна.

(abc)* - повторение abc 0 и более раз.

Спецсимволы. Чередование и группирование.

`(?:x)`	Находит, но не запоминает x
`x(?=y)`	Запоминает x, если после него идёт y
`x(?!y)`	запоминает все x, кроме того, после которого идёт y.

заменить ё на ❤, но исключить то, за которым стоит ж

```
document.write(
document.getElementById('replace').innerHTML.replace(/ё(?!ж)/gi,
'<\/b>&#10084;') );
```

Ёжик вышел из уб**еж**ища под ❤**л**кой к **дж**ипу "Бер❤зка".
Та**ёж**ник записал: "01.20.2000 года был **ёж**. Назвал "Се**рёж**кой".

Спецсимволы. Чередование и группирование.

x|y — запоминает или x, или y

```
document.write(document.getElementById('replace').
innerHTML.replace(/ë|е/gi, '&#10084;') );
```

❤**ж**ик выш❤л из уб❤**ж**ища под ❤**л**кой к **дж**ипу "Б❤р❤**з**ка".
Та❤**ж**ник записал: "01.20.2000 года был ❤**ж**. Назвал "С❤р❤**ж**кой".

Примеры

```
document.write(document.getElementById('replace').
innerHTML.replace(/\s/gi,"") );
```

удалить пробелы из строки

```
document.write(document.getElementById('replace').
innerHTML.replace(/<[^>]*>/g, '') );
```

удалить все теги

Примеры

```
^(\+380\(542\)[0-9]{6})$
```

телефонный номер в формате **+380(542)xxxxxx**

Методы JavaScript для работы с RegExp

Чтобы проверить, подходит ли строка под регулярное выражение, используется метод test:

```
if ( /\s/.test("строка") ) {
...В строке есть
пробелы!...
}
```

```
Returned value: true
Returned value: false
```

```
var str="Hello world!";

//поиск "Hello"
var patt=/Hello/g;
var result=patt.test(str);
document.write("Returned value: " + result);

//поиск “hi"
patt=/hi/g;
result=patt.test(str);
document.write("<br>Returned value: " +
result);
```

Методы JavaScript для работы с RegExp

exec()

Используется для сравнения строк и возвращает совпадения, если их нет – возвращает NULL

```
var str = "The best things in life are free";
var patt = new RegExp("e");
var res = patt.exec(str);
```

Результат:

e

Методы JavaScript для работы с RegExp

match()

выполняет поиск совпадения в строке. Возвращет массив данных либо null, если совпадения отсутствуют.

```
var str =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
var regexp = /[A-E]/gi;
var matches_array = str.match(regexp);
```

Результат: `['A', 'B', 'C', 'D', 'E', 'a', 'b', 'c', 'd', 'e']`

Методы JavaScript для работы с RegExp

search()

тестирует на совпадение в строке. Возвращет индекс совпадения, или -1 если совпадений не будет найдено.

```
var str = "Visit Sumy";
var n = str.search(/Sumy/i);
```

Результат: **6**

Методы JavaScript для работы с RegExp

replace()
ищет строку заданного значения или заданную регулярным выражением и заменяет её на указанную строку

```
var str = "Visit Sumy State University!";
    var res = str.replace("Sumy State
          University", "SSU");
```

Результат:
Visit SSU!

Методы JavaScript для работы с RegExp

split()

Использует регулярное выражение или фиксированую строку чтобы разбить строку на массив подстрок.

```
var myString = 'Hello World. How are you
doing?';
var splits = myString.split(' ', 3);
```

Результат: ["Hello", "World.", "How"]

Методы JavaScript для работы с RegExp

toString

возвращает строковое значение регулярного выражения

```
var patt = new RegExp("Hello World", "g");
var res = patt.toString();
```

Результат:

/Hello World/g

Рекомендации

- **Используйте только те регулярные выражение, которые вы написали сами.**
- **Запомните значения мета-символов в контексте.**
- **Думайте как работает парсер. Старайтесь ему помочь.**
- **Регулярные выражения — это разбор строки. Все прочее лучше делается по-другому**

Материалы для самостоятельного изучения

- https://www.w3schools.com/js/js_regexp.asp
- https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Regular_Expressions
- Google.com

Q&A

Thank You

Focused forward

Netcracker