Лекція 7.
Регулярні вирази в JavaScript

Загальний опис

Регулярні вислови є зразки для пошуку заданих комбінацій символів у текстових рядках (такий пошук називається зіставленням із зразком). Існує два способи привласнення змінним регулярних виразів, а саме:
Використання ініціалізатора об'єкта: var re = / pattern / switch?.

Використання конструктора RegExp: var re = new RegExp ("pattern" [, "switch"]?).

Тут pattern - регулярний вираз, а switch - необов'язкові опції пошуку.

При створенні регулярного виразу слід враховувати, що укладення його в лапки тягне за собою необхідність використовувати escape-послідовності, як і в будь-який інший строковой константі. Наприклад, наступні два вирази еквівалентні:

```
var re = /\w+/g;
var re = new RegExp("\\w+", "g"); // В рядку "\" повинно змінитися на "\\
```

Регулярні вирази використовуються методами exec і test об'єкта RegExp і методами match, replace, search і split об'єкта String. Якщо нам потрібно просто перевірити, чи містить даний рядок підрядок, відповідну зразком, то використовуються методи test або search. Якщо ж нам необхідно витягти підрядок (або підрядка), відповідні зразком, то нам доведеться скористатися методами exec або match. Метод replace забезпечує пошук заданого підрядка і заміни її на інший рядок, а метод split дозволяє розбити рядок на кілька підрядків, грунтуючись на регулярному виразі або звичайною текстовому рядку. Більш докладні відомості про застосування регулярних виразів наведено в описі відповідних методів.

Синтаксис регулярних виразів

Регулярний вираз може складатися із звичайних символів; в цьому випадку воно буде відповідати заданій комбінації символів в рядку. Наприклад, вираз / ком / відповідає виділеним підстроками в наступних рядках: "грудка", "ласунка", "головком флоту". Однак, гнучкість і міць регулярними виразами надає можливість використання в них спеціальних символів, які перераховані в наступній таблиці.

Спеціальні символи в регулярних виразах:

\ - Для символів, які зазвичай трактуються буквально, означає, що наступний символ є спеціальним. Наприклад, / n / відповідає букві n, а / \ n / відповідає символу перекладу

рядка. Для символів, які зазвичай трактуються як спеціальні, означає, що символ повинен розумітися буквально. Наприклад, / ^ / означає початок рядка, а / \ ^ / відповідає просто символу ^. / \ \ / Відповідає зворотного косою межах \.

Символ	Значення
****	Для звичайних символів - робить їх спеціальними. Наприклад, вираз / s / шукає просто символ 's'. А якщо поставити \ перед s, то / \ s / вже позначає пробільний символ. І навпаки, якщо символ спеціальний, наприклад *, то \ зробить його просто звичайним символом "зірочка". Наприклад, / a * / шукає 0 або більше поспіль йдуть символів 'a'. Щоб знайти a із зірочкою 'a *' - поставимо \ перед спец. символом: / a \ * /.
^	Позначає початок вхідних даних. Якщо встановлений прапор багаторядкового пошуку ("m"), то також спрацьовує при початку нової рядка. Наприклад, / ^ A / не знайде 'A' в "an A", але знайде перші 'A' в "An A."
$	<Позначає кінець вхідних даних. Якщо встановлений прапор багаторядкового пошуку, то також спрацює в кінці рядка. Наприклад, / t $ / не знайде 't' в "eater", але знайде - в "eat".
*****	Позначає повторення 0 або більше разів. Наприклад, / bo * / знайде 'boooo' в "A ghost booooed" і 'b' в "A bird warbled", але нічого не знайде в "A goat grunted".
+	Позначає повторення 1 або більше разів. Еквівалентно (1,). Наприклад, / a + / знайде 'a' в "candy" і все 'a' в "caaaaaaandy"./td>
?	Означає, що елемент може як бути присутнім, так і бути відсутнім. Наприклад, / e? Le? / Знайде 'el' в "angel" і 'le' в "angle." Якщо використовується відразу після одного з квантіфікаторов *, +,?, Або (), то задає "нежадібних" пошук (повторення мінімально можлива кількість разів, до найближчого наступного елемента патерну), на противагу "жадібному" режиму за замовчуванням, при якому кількість повторень максимально, навіть якщо наступний елемент патерну теж підходить. Крім того,? використовується в попередньому перегляді, який описаний в таблиці під (?=), (?!), і (?:).
.	(Десяткова точка) позначає будь-який символ, крім перекладу рядки: \ n \ r \ u2028 or \ u2029. (Можна використовувати [\ s \ S] для пошуку будь-який символ, включаючи переклади рядків). Наприклад, /. N / знайде 'an' і 'on' в "nay, an apple is on the tree", але не 'nay'.
(x)	Знаходить x і запам'ятовує. Це називається "запам'ятовуючі дужки". Наприклад, / (foo) / знайде і запам'ятає 'foo' в "foo bar." Знайдена підрядка зберігається в масиві-результаті пошуку або в наперед визначених властивості об'єкта RegExp: $ 1, ..., $ 9. Крім того, дужки об'єднують те, що в них знаходиться, в єдиний елемент патерну. Наприклад, (abc) * - повторення abc 0

	і більше разів.
(?:x)	Знаходить x, але не запам'ятовує знайдене. Це називається "незапям'ятовуючи дужки". Знайдена підрядка не зберігається в масиві результатів та властивості RegExp. Як і всі дужки, об'єднують що знаходиться в них в єдиний підпатерн.
x(?=y)	Знаходить x, тільки якщо за x слід y. Наприклад, / Jack (? = Sprat) / знайде 'Jack', тільки якщо за ним слідує 'Sprat'. / Jack (? = Sprat | Frost) / знайде 'Jack', тільки якщо за ним слідує 'Sprat' або 'Frost'. Однак, ні 'Sprat' nor 'Frost' не увійдуть в результат пошуку.
x(?!y)	Знаходить x, тільки якщо за x не слід y. Наприклад, / \ d + (?! \.) / Знайде число, тільки якщо за ним не слід десяткова крапка. / \ D + (?! \.) /. Exec ("3.141") знайде 141, але не 3.141.
x|y	Знаходить x або y. Наприклад, / green | red / знайде 'green' в "green apple" і 'red' в "red apple."
{n}	Де n - позитивне ціле число. Знаходить рівно n повторення попереднього елемента. Наприклад, / a (2) / не знайде 'a' в "candy," але знайде обидва a в "caandy," і перші два a в "caaandy."
{n,}	Де n - позитивне ціле число. Знаходить n і більш за повторення елементу. Наприклад, / a (2,) не знайде 'a' в "candy", але знайде всі 'a' в "caandy" і в "caaaaaaandy."
{n,m}	Де n і m - позитивні цілі числа. Знаходять від n до m повторень елемента.
[xyz]	Набір символів. Знаходить будь-який з перелічених символів. Ви можете вказати проміжок, використовуючи тире. Наприклад, [abcd] - те ж саме, що [ad]. Знайде 'b' в "brisket" і 'c' в "ache".
[^xyz]	Будь-який символ, крім зазначених у наборі. Ви також можете вказати проміжок. Наприклад, [^ abc] - те ж саме, що [^ ac]. Знайде 'r' в "brisket" і 'h' в "chop."
[\b]	Знаходить символ backspace. (Не плутати з \ b.)
\B	Позначає не кордон слів. Наприклад, / \ w \ Bn / знайде 'on' в "noonday", а / y \ B \ w / знайде 'ye' в "possibly yesterday."
\cX	Де X - буква від A до Z. Позначає контрольний символ в рядку. Наприклад, / \ cM / позначає символ Ctrl-M.
\d	знаходить цифру з будь-якого алфавіту (у нас же юнікод). Використовуйте [0-9], щоб знайти тільки звичайні цифри. Наприклад, / \ d / або / [0-9] / знайде '2 'в "B2 is the suite number."
\D	Знайде нецифрових символ (усі алфавіти). [^ 0-9] - еквівалент для

	звичайних цифр. Наприклад, / \ D / або / [^ 0-9] / знайде 'B' в "B2 is the suite number."
\f,\r,\n	Відповідні спецсимволи form-feed, line-feed, переклад рядка.
\s	Знайде будь-який пробільний символ, включаючи пробіл, табуляцію, переклади рядки і інші юнікодні пробільні символи. Наприклад, / \ s \ w * / знайде 'bar' в "foo bar."
\S	Знайде будь-який символ, крім пробільних. Наприклад, / \ S \ w * / знайде 'foo' в "foo bar."
\t	Символ табуляції.
\v	Символ вертикальної табуляції.
\w	Знайде будь-який словесний (латинський алфавіт) символ, включаючи літери, цифри і знак підкреслення. Еквівалентно [A-Za-z0-9_]. Наприклад, / \ w / знайде 'a' в "apple," '5 'в "$ 5.28," і '3' в "3D."
\W	Знайде будь-який не-(лат.) словесний символ. Еквівалентно [^ A-Za-z0-9_]. Наприклад, / \ W / i / [^ $ A-Za-z0-9_] / однаково знайдуть '%' в "50%."
\n	де n - ціле число. Зворотній посилання на n-у запам'ятовані дужками підрядок. Наприклад, / apple (,) \ sorange \ 1 / знайде 'apple, orange,' в "apple, orange, cherry, peach.". За таблицею є більш повний приклад.
\0	Знайде символ NUL. Не додавайте в кінець інші цифри.
\xhh	Знайде символ з кодом hh (2 шестнадцатірічні цифри)
\uhhhh	Знайде символ з кодом hhhh (4 шестнадцатірічні цифри).

Регулярні вирази обчислюються аналогічно іншим виразами JavaScript, тобто з урахуванням пріоритету операцій: операції, що мають більший пріоритет, виконуються першими. Якщо операції мають рівний пріоритет, то вони виконуються зліва направо. У наступній таблиці наведено список операцій регулярних виразів в порядку убування їх пріоритетів; операції, розташовані в одному рядку таблиці, мають рівний пріоритет.

Операції:

\ () (?:) (?=) (?!) [] * + ? . {n} {n,} {n,m} ^ $ \метасимвол |

Опції пошуку

При створенні регулярного виразу ми можемо вказати додаткових опції пошуку:
i (ignore case). Чи не розрізняти рядкові і прописні літери.

g (global search). Глобальний пошук всіх входжень зразка.

m (multiline). Багатостроковий пошук.

Будь-які комбінації цих трьох опцій, наприклад ig або gim.

Приклад.

```
var s = "Вивчаємо мову JavaScript";
var re = /JAVA/;
var result = re.test(s) ? ""  : " не";
alert("Рядок " + s + result + " відповідає зразку " + re);
```

Оскільки регулярні вирази розрізняють малі та великі літери, цей приклад виведе у вікно оглядача текст:
Рядок "Вивчаємо мову JavaScript" не відповідає зразку / JAVA /

Якщо ми тепер замінимо другий рядок прикладу на var re = / JAVA / i;, то на екран буде виведений текст:
Рядок "Вивчаємо мову JavaScript" відповідає зразку / JAVA / i

Тепер розглянемо опцію глобального пошуку. Вона зазвичай застосовується методом replace при пошуку зразка і заміни знайденої підрядка на нову. Справа в тому, що за умовчанням цей метод робить заміну тільки першої знайденої підрядка і повертає отриманий результат. Розглянемо таку ситуацію:

```
var s = "Ми пишемо сценарії на JavaScript, " + "але JavaScript - не єдина сценарна мова.";
var re = /JavaScript/;
alert(s.replace(re, "VBScript"));
```

Він виводить у вікно оглядача текст, який явно не відповідає бажаному результату: Ми пишемо сценарії на VBScript, але JavaScript - не єдиний сценарний мову. Для того, щоб всі входження рядка "JavaScript" були замінені на "VBScript", ми повинні змінити значення регулярного виразу на **var re = / JavaScript / g;**. Тоді результуючий рядок буде мати вигляд:
Ми пишемо сценарії на VBScript, але VBScript - не єдина сценарна мова.

Нарешті, опція багаторядкового пошуку дозволяє проводити зіставлення зі зразком строкового вираження, що складається з декількох рядків тексту, з'єднаних символами розриву рядка. Типово, зіставлення зі зразком припиняється, якщо знайдений символ розриву рядка. Дана опція долає вказане обмеження і забезпечує пошук зразка по всій заданій стрічці. Вона також впливає на інтерпретацію деяких спеціальних символів у регулярних виразах, а саме: Зазвичай символ **^** зіставляється тільки з першим елементом рядка. Якщо ж опція багаторядкового пошуку включена, то він також зіставляється з будь-

яким елементом рядка, якому передує символ розриву рядка. Зазвичай символ **$** зіставляється тільки з останнім елементом рядка. Якщо ж опція багаторядкового пошуку включена, то він також зіставляється з будь-яким елементом рядка, який є символом розриву рядка.

Запам'ятовування знайдених підрядків

Якщо частина регулярного виразу укладена в круглі дужки, то відповідна їй підрядок буде запам'ятати для подальшого використання. Для доступу до запомненним підстроками використовуються властивості $ 1,:, $ 9 об'єкта RegExp або елементи масиву, що повертається методами exec і match. В останньому випадку кількість знайдених і запомненних подстрок не обмежена.

Наступний сценарій використовує метод replace для перестановки слів у рядку. Для заміни знайденого тексту використовуються властивості $ 1 і $ 2.

```
var re = /(\w+)\s(\w+)/i; var str = "Mikhail Bulgakov"; document.write(str.replace(re, "$2, $1"))
```

Цей сценарій виведе у вікно оглядача текст:
Bulgakov, Mikhail

т.я. \ W = [A-Za-z0-9_], то російські літери працювати не будуть. Якщо ми хочемо використовувати російські літери, то нам доведеться трішки модифікувати код:

```
var re = /([а-я]+)\s([а-я]+)/i; var str = "Михаил Булгаков"; document.write(str.replace(re, "$2, $1")); // Булгаков, Михаил
```

Цей сценарій виведе у вікно оглядача текст:
Булгаков, Михайло

Введення

Регулярні вирази - це потужний засіб для обробки вхідних даних. Завдання, що вимагає заміни або пошуку тексту, може бути красиво вирішена за допомогою цього "мови всередині мови". І хоча максимальний ефект від регулярних виразів можна домогтися при використання серверних мов, все ж не варто недооцінювати можливості цього додатка і на стороні клієнта.

Основні поняття

Регулярний вираз (regular expression) - засіб для обробки рядків або послідовність символів, що визначає шаблон тексту.

Модифікатор - призначений для "інструктування" регулярного виразу.

Метасимволи - спеціальні символи, які служать командами мови регулярних виразів.

Регулярний вираз задається як звичайна змінна, тільки замість лапок використовується слеш, наприклад:

```
var reg=/рег_вираз/
```

Під найпростішими шаблонами будемо розуміти такі шаблони, які не потребують будь-яких спеціальних символах.

Припустимо, нашим завданням є заміна всіх букв "р" (малих і великих) на латинську велику букву "R" в словосполученні Регулярні вирази.

Створюємо шаблон var reg = / p / і скористайтесь методом replace, здійснюємо задумане

```
<script language="JavaScript"> var str="Регулярні вирази" var reg=/p/ var result=str.replace(reg, "R") document.write(result) </script>
```

В результаті отримаємо рядок 'Регулярні вирази', заміна відбулася лише на першому входженні букви "р" з урахуванням регістру. Але під умови нашого завдання цей результат не підходить ... Тут нам знадобляться модифікатори "g" і "i", які можуть використовуватися як окремо, так і спільно.
Ці модифікатори ставляться в кінці шаблону регулярного виразу, після слеша, і мають такі значення: модифікатор "g" - задає пошук в рядку як "глобальний", тобто в нашому випадку заміна відбудеться для всіх входжень літери "р". Тепер шаблон виглядає так: **var reg = / p / g**, підставивши його в наш код

```
<script language="JavaScript"> var str="Регулярні вирази" var reg=/p/g var result=str.replace(reg, "R") document.write(result) </script>
```

отримаємо рядок 'Регулярні вирази'.
Модифікатор "i" - задає пошук в рядку без урахування регістра, додавши цей модифікатор в наш шаблон var reg = / p / gi, після виконання скрипта одержимо шуканий результат нашого завдання - 'Регулярні вирази'.

Спеціальні символи (метасимволи)

Метасимволи задають тип символів шуканого рядка, спосіб оточення шуканого рядка в тексті, а так само кількість символів окремого типу в переглядаючому тексті. Тому

метасимволи можна розділити на чотири групи:

Метасимволи пошуку збігів.

Кількісні метасимволи.

Метасимволи позиціонування.

Метасимволи пошуку збігів

\ B межа слова, задає умову, при якому шаблон повинен виконуватися на початку або кінці слів.

\ B НЕ межа слова, задає умову, при якому шаблон не виконується на початку або кінці слова.

\ D цифра від 0 до 9. \ D трохи цифра.

\ S одиночний порожній символ, відповідає символу пробілу.

\ S одиночний непорожній символ, будь-який один символ за винятком пробілу.

**** W літера, цифра або символ підкреслення.

**** W не буква, цифра або символ підкреслення.

. Будь-який символ, будь-які знаки, букви, цифри і т.д.

[] Набір символів, задає умову, при якому шаблон повинен виконуватися при будь-якому збігу символів ув'язнених в квадратні дужки.

[^] Набір не входять символів, задає умову, при якому шаблон не повинен виконуватися при будь-якому збігу символів ув'язнених в квадратні дужки.

Кількісні метасимволи

***** Нуль і більшу кількість разів.

? Нуль або один раз + Один і більшу кількість разів.

{N} точно n разів.

{N,} n або більшу кількість разів.

{N, m} принаймні, n разів, але не більше ніж m разів.

Метасимволи позиціонування

^ На початку рядка.

$ В кінці рядка.

Деякі методи для роботи з шаблонами

replace - даний метод ми вже використали на самому початку статті, він призначений для пошуку зразка і заміни знайденої підрядка на нову підрядок.
test - даний метод перевіряє, чи є збіги в рядку щодо шаблону і повертає false, якщо зіставлення зі зразком закінчилося невдачею, в іншому випадку true.

Приклад:

```
<script language="JavaScript"> var str="JavaScript" var reg=/PHP/ var result=reg.test(str) document.write(result) </script>
```

виведе в якості результату false, тому що рядок "JavaScript" не дорівнює рядку "PHP".

Також метод test може повертати замість true або false будь-яку іншу рядок задану програмістом. наприклад:

```
<script language="JavaScript"> var str="JavaScript" var reg=/PHP/ var result=reg.test(str) ? "Строка совпала" : "Строка не совпала" document.write(result) </script>
```

в цьому випадку в якості результату буде рядок 'Рядок не збігся'.
exec - даний метод виконує зіставлення рядка із зразком, заданим шаблоном. Якщо зіставлення зі зразком закінчилося невдачею, то повертається значення null. В іншому випадку результатом є масив підрядків, відповідних заданому зразку. / * Перший елемент масиву дорівнюватиме вихідної рядку задовольняє заданому шаблону * / наприклад:

```
<script language="JavaScript"> var reg=/(\d+).(\d+).(\d+)/ var arr=reg.exec("Я родился 15.09.1980") document.write("Дата рождения: ", arr[0], "< br>")

document.write("День рождения: ", arr[1], "< br>") document.write("Месяц рождения: ", arr[2], "< br>") document.write("Год рождения: ", arr[3], "< br>") </script>
```

в результаті отримаємо чотири рядки:
Дата народження: 15.09.1980

День народження: 15

Місяць народження: 09

Рік народження: 1980