PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Desenvolupament web a l'entorn client

UT 5: Expressions regulars

Joan Pons Tugores

Índex de continguts

Expressions regulars

De manera bastant habitual hem de cercar informació dins cadenes de text. Podem utilitzar els mètodes de String, però no podem anar més enllà de cercar subcadenes literals dins la cadena.

De vegades ens és més útil fer una cerca o una validació per patrons. Per exemple, per validar un correu electrònic o un número de telèfon en un determinat format ens trobarem amb moltes dificultats si no ho feim utilitzant un patró.

Les expressions regulars són una forma d'expressar aquests patrons, encara que la seva sintaxi no és precisament amigable.

En Javascript les expressions regulars són objectes.

Expressions regulars literals

Les expressions regulars són caràcters, però no cadenes de caràcters. És a dir, no són cadenes de caràcters perquè no van tancades entre cometes, ni simples ni dobles.

Els delimitadors de l'expressió regular són les barres / i dins hi podem posar qualsevol combinació de caràcters normals i caràcters especials amb un significat propi.

Per exemple

```
const expReg = /^car/;
```

assigna una expressió regular a la constant *expReg*. Està delimitada per /, té un caràcter especial, ^, i caràcters normals, *car*.

Aquesta expressió en concret validarà cadenes que comencin amb les lletres *car*, per exemple caragol, caràcter, carretera, ...

Expressions regulars amb *new*.

Podem crear les expressions regulars amb *new*. És útil quan hem de formar l'expressió regular en temps d'execució, per exemple depenent d'alguna dada entrada per l'usuari. No és tan eficient com una expressió regular literal.

Es pot passar una expressió regular literal o una cadena de caràcters al constructor.

```
const expReg = new RegExp(/^car/);
```

Si és una expressió regular va amb les barres i sense cometes, en canvi si és una cadena va amb cometes i sense barres.

```
const expReg = new RegExp("^car");
```

Podem posar com a segon paràmetre una bandera, flag, per exemple per indicar que ignori la diferència entre majúscules i minúscules.

```
const expReg = new RegExp(/^car/,"i");
const expReg2 = new RegExp("^car","i");
```

Caràcters especials

De vegades volem fer cerques més complicades que una subcadena concreta, per exemple volem trobar dins la cadena una *a* seguida d'una o més *b*. En aquests casos hem d'utilitzar els caràcters especials.

De caràcters especials n'hi ha d'unes quantes categories, les veurem a continuació.

Assercions

Permeten establir límits a les expressions regulars

- ^ L'inici de la cadena ha de coincidir amb l'expressió regular.

  /^Ara/ coincideix amb "Ara començam" però no amb "Començam Ara"

- $ El final de la cadena ha de coincidir amb l'expressió regular.

  /Ara$/ coincideix amb "Començam Ara" però no amb "Ara Començam"

- \b Marca el límit de paraula, és a dir, la posició en la que un caràcter de paraula no va seguit d'un altre caràcter de paraula, per exemple una lletra i un espai.

  /\bl/ troba la l de lluna
  /un\b/ no troba *un* de *lluna* perquè *un* va seguit de *a*.
  /una\b/ coincideix amb *una* de *lluna* perquè és el final de la cadena i no va seguit de cap caràcter de paraula.

- \B És la posició en la que el caràcter anterior i posterior són del mateix tipus, paraula o no.

  /\Bra/ Coincideix amb *caracter* però no amb *rata*

Classes de caràcters

Classifiquen els diferents caràcters, per exemple lletres o dígits.

- . Un caràcter únic excepte finals de línia com \r, \n, ...

  /.i/ coincideix amb 'li' "client"

- \d Dígit, del 0 al 9

  /\d/ coincideix amb el 2 de "B2"

- \D No és un dígit.

  /\D/ coincideix amb la B de "B2"

- \w caràcter alfanumèric, A-Z, 0-9, _ No inclou els caràcters accentuats, ç, ñ, ...

  /\w/ Coincideix amb la P de "Poma" o amb el 5 de "5€"

- \W no és un caràcter alfanumèric.

  /\w/ Coincideix amb el € de "5€"

- \s un espai en blanc o similar(tabulador, salt de línia, ...)

  /\s/ Troba l'espai entre *Entorn* i *client* a "Entorn client"

- \S la negació de l'anterior.
- \t tabulador
- \r retorn de carro
- \n salt de línia

Gups i rangs

Faciliten la tasca d'especificar rangs de valors o grups

- x|y Coincideix amb x o amb y.

  /client|servidor/ coincideix amb qualsevol de les dues opcions.

- [rsxy] Un conjunt de caràcters. Coincidirà amb qualsevol d'ells.

  [rsxyz] coincidrà amb la s de "mapes".

- [x-z] Un rang de caràcters. Coincidirà amb qualsevol dels caràcters del rang.

  [b-f] coincidira amb la e de "empelt".

- [^...] La negació d'un rang, per exemple [^rsxy] o [^b-f]

- (x) Grup. Coincideix amb *x* i recorda, torna la coincidència.

Quantificadors

El nombre de caràcters o expressions que han de coincidir

- x* 0 o mes vegades l'element anterior

  la* coincidirà amb una *l* i zero o més *a*, per exemple a "Hola", "Hol" i "Holaaa"

- x+ 1 o més vegades l'element anterior

  la+ coincidirà amb una *l* i zero o més *a*, per exemple a "Hola" i "Holaaa" però no amb "Hol"

- x? 0 o 1 vegada l'element anterior.

  /ra?/ coincidirà amb cama i amb càmera, però no amb carrara

- x{n} Exactament n repeticions de l'element anterior

  /ra{2}/ coincideix amb Carrara, però no amb cara

- x{n,} Coincideix amb almanco 2 aparicions de l'element anterior

  /ra{2,}/ coincideix amb Carrara i amb Carrarara, però no amb cara

- x{n,m} on n>=0, m>0 i m>n Coincideix amb al manco n i com a màxim m aparicions de l'element anterior.

  /ra{2,3}/ coincideix amb Carrara i amb Carrarara, però no amb cara ni amb Carrararara.

Flags

Les banderes es situen al final de les expressions regulars, darrera la / que tanca l'expressió i permeten funcions, per exemple, per ignorar majúscules i minúscules.

- g Per defecte les expressions regulars s'aturen a la primera coincidència. Si posam aquesta bandera les troba totes

  /foo/g troba les dues aparicions a foobarfoo, si l'utilitzam per substituir foo per un altre valor els substituirà tots dos.

- i ignora majúscules i minúscules

  /car/i trobarà Carrara i carrara.

- m Multilínia. Tracta la cadena com a diverses línies, de manera que, per exemple ^ i $ s'apliquen a cada línia, no només a l'inici i final de la cadena.
- y Sticky. L'expressió regular manté l'índex entre test i test.

Propietats

- .lastIndex L'index al qual començarà a cercar per la següent coincidència.
- .flags Cadena amb les banderes de l'expressió regular.
- .global Si ha de cercar totes les possibles ocurrències dins la cadena o no.
- .ignoreCase Si hi ha present la bandera *i* o no.
- .multiLine Si hi ha present la bandera *m* o no.
- .source El text de l'expressió regular.
- .sticky Si hi ha present la bandera *y* o no.

Mètodes de l'expressió regular

- .**exec:** Torna una de les aparicions que ha trobat a la cadena. Si l'expressió regular té la bandera global o sticky, a cada execució torna la següent aparició, *lastIndex* manté el valor. Si no, a cada execució torna la primera de totes, cada vegada s'inicialitza *lastIndex* a 0. Es recomana utilitzar en lloc seu *match* o *matchAll* de String.

```
const regex1 = RegExp('foo*', 'g');
const str1 = 'table football, foosball';
let array1;
while ((array1 = regex1.exec(str1)) !== null) {
  console.log(`Found ${array1[0]}. Next starts at ${regex1.lastIndex}.`);
  // expected output: "Found foo. Next starts at 9."
  // expected output: "Found foo. Next starts at 19."
}
```

- .**test**: Cerca coincidències a la cadena que rep com a paràmetre i torna true o false. És el recomanat si únicament volem saber si a la cadena hi ha cap ocurrència però no ens interessa saber quina és, per exemple per validar un formulari.

```
const str = 'table football';
const regex = /foo*/;
console.log(regex.test(str));
```

Mètodes d'String per a expressions regulars

- .**match**: Torna un array amb totes les ocurrències de l'expressió regular dins la cadena que l'executa

```
var cadena =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
var expresio = /[A-E]/gi;
var resultat = cadena.match(expresio);
console.log(resultat);
// Torna un array de 10 posicions on cada posició és una de les
coincidències.
```

- .**matchAll**: Torna un iterador de totes (si la bandera *g* està present) les ocurrències de l'expressió regular a la cadena. Inclou els grups de captura, és a dir, les parts de la cadena que coincideixen amb l'expressió regular. Recomanat si volem recuperar les ocurrències de l'expressió regular a la cadena.

```
const regexp = RegExp('foo[a-z]*','g');
const cadena = 'taula football, foosball';
const ocurrencies = cadena.matchAll(regexp);

for (const ocurrencia of ocurrencies) {
        console.log(`Trobat ${ocurrencia[0]} inici=${ocurrencia.index}
            final=${ocurrencia.index + ocurrencia[0].length}.`);
}
//Mostra la coincidència, la posició on comença i on acaba.
```

- .**replace**: té dos paràmetres, l'expressió regular i la cadena substituta. Torna una cadena amb totes les ocurrències de l'expressió regular substituides per la cadena substituta.

```
const p = 'The quick brown fox jumps over the lazy dog. If the dog reacted, was it really lazy?';
const regex = /dog/gi;
console.log(p.replace(regex, 'ferret'));
// expected output: "The quick brown fox jumps over the lazy ferret. If the ferret reacted, was it really lazy?"
```

- .**search**: Comprova si hi ha cap ocurrència de l'expressió regular a la cadena. Torna l'índex on comença la primera ocurrència o -1 si no n'hi ha cap. Recomanat utilitzar test en lloc seu.

```
const p = 'Això era i no era';
const regex = /era/;
console.log(p.search(regex));
//Mostra 5, l'index de la primera coincidència.
```

- .**split**: Te dos paràmetres, el separador i el màxim d'elements a tornar. Divideix la cadena a cada aparició del separador i torna un array amb aquestes subcadenes. El separador pot ser una cadena o una expressió regular. Si posam el segon paràmetre, l'array conté com a màxim aquesta quantitat de subcadenes.

```
var noms = "Harry Trump ;Fred Barney; Helen Rigby ; Bill Abel ";
var expresioRegular = /\s*;\s*/;
var llistaNoms = noms.split(expresioRegular);
console.table(llistaNoms);
//Mostra per consola un array on a cada posició hi ha un dels noms.
```

Expressions regulars a elements dels formularis

Els elements input de tipus *text*, *tel*, *email*, *url*, *password* i *search* tenen l'attribut pattern que permet indicar l'expressió regular que ha de cumplir el valor d'aquest element per ser considerat vàlid.

```
<input type="tel" pattern="\d{3}-\d{3}-\d{3}">
//Accepta valors del tipus 123-456-789
```

Més informació

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions