Espressioni regolari: Cosa sono e come iniziare?

Tra tutti i potenti strumenti messi a disposizione di un programmatore, **le espressioni regolari** occupano sicuramente un posto di rilievo. La loro flessibilità sia nel campo della *ricerca* che in quello della *validazione* di stringhe e dati le rende un must per un programmatore web (e non solo). Purtroppo però, tale flessibilità è pagata a caro prezzo: le espressioni regolari sono da sempre avvolte in una sorta di aurea misteriosa. Un non addetto ai lavori che legga una riga di una *regex* (come spesso vengono chiamate) riuscirebbe a cogliere solo una serie di caratteri e simboli intervallati da parentesi, apparentemente senza significato. **E proprio la sintassi criptica è uno dei motivi per cui spesso si rimanda lo studio di questo potentissimo strumento**.

In questa serie di articoli vedremo come sfatare questo mito, esaminando elemento per elemento questo affascinante linguaggio. Iniziamo con le basi.

Innanzitutto, partiamo da una semplice definizione, che non vuole essere "accademica": **un'espressione regolare è un modello, scritto in un opportuno linguaggio, attraverso il quale una parola o frase può essere cercata all'interno di un testo, oppure validata per conformarsi ad un certo formato**.

Nella vita di tutti i giorni associamo, senza rendercene conto, un certo formato dei dati a informazioni ben precise: tre numeri di due cifre separati da un trattino ci fanno subito pensare ad una data (16-07-86), delle lettere separate da una chiocciola (@) seguite dal punto e da altre lettere ci ricordano un'email (prova@esempio.exl), e così via.

Le espressioni regolari non fanno altro che tradurre tali modelli da un linguaggio naturale come l'italiano (tre numeri di due cifre separati da un trattino) in un linguaggio comprensibile per una macchina: (\d{2}-\d{2}-\d{2}).

Prima di cominciare ad esaminare i vari elementi delle espressioni regolari, vediamo come poterne testare il funzionamento. A questo riguardo, **le regex sono un linguaggio del tutto particolare**, poiché possono essere utilizzate solo all'interno di un altro linguaggio (Perl, PHP, JavaScript, ecc), o di un programma (OpenOffice per la ricerca nel testo, ad esempio).

Per provare tutte le regex proposte in questo articolo, utilizzeremo un testo tratto da "*Alice nel paese delle meraviglie*" ed uno strumento online sviluppato da Steven Levithan: regexpal il cui scopo è proprio quello di poter testare le varie espressioni regolari scritte nel campo in alto sul testo inserito nella schermata principale: tutti i caratteri che corrisponderanno saranno evidenziati.

La base delle regex: il singolo carattere

L'espressione regolare più semplice è quella rappresentata da **un singolo carattere**: la lettera '*g*' ad esempio è l'espressione regolare che deve essere utilizzata per trovare tutte le occorrenze di '*g*'

all'interno di un testo.

Ovviamente questo caso base si può estendere ad un "gruppo di caratteri": utilizzando l'espressione regolare '*ebbe*' vengono evidenziate tutte le occorrenze di questa parola all'interno del testo.

Se hai provato quest'ultimo esempio nel regexpal, avrai sicuramente notato che oltre alla parola '*ebbe*' come passato remoto del verbo avere, sono state evidenziate anche le parti finali delle parole '*avrebbe*' o '*farebbe*': questo perché **le espressioni regolari cercano di trovare quante più corrispondenze possibili, dato il modello da cercare**. Vedremo più avanti come fare per trovare solo le corrispondenze per una data parola.

I metacaratteri

Come già detto, ogni carattere è una espressione regolare. Esistono delle eccezioni, rappresentate dai seguenti caratteri:

1. L'asterisco: *
2. Il segno più: +
3. Il punto interrogativo: ?
4. Le parentesi tonde: ( )
5. Le parentesi quadre: [ ]
6. Le parentesi graffe: { }
7. Il punto: .
8. La 'pipe': |
9. L'apice: ^
10. Il simbolo del dollaro: $
11. Il backslash: \

L'asterisco (*)

Nel linguaggio delle espressioni regolari l'asterisco viene posto dopo un carattere (o un gruppo di caratteri) e significa **"trova nel testo nessuna o quante più corrispondenze possibili del carattere"**. Facciamo subito un esempio. L'espressione regolare:

che, tradotta in parole sta per "trova tutte le occorrenze di una '*p*' seguita da zero o più '*a*'", troverà corrispondenze per tutte le seguenti righe:

La cosa importante da capire è che l'asterisco (come la maggior parte dei metacaratteri) ha effetto **solo sul carattere che lo precede**: per farlo agire su gruppi di caratteri è necessario utilizzare le

parentesi tonde. L'espressione regolare:

ovvero "tutte le occorrenze di una *'c'* maiuscola seguite zero o più volte dal gruppo '*asa*'" troverà corrispondenza per le prime tre stringhe seguenti, ma non l'ultima:

Il segno più (+)

Molto simile all'asterisco come funzionamento, **il simbolo '*più*' cerca "una o più occorrenze del carattere che lo precede"**. La differenza con l'asterisco, quindi, sta nel fatto che il carattere non è più opzionale, ma deve essere presente per forza. Modificando l'esempio fatto precedentemente:

tale espressione regolare viene resa come: "trova tutte le occorrenze della lettera '*p*' minuscola seguite **da almeno una** lettera '*a*' minuscola. In tal caso la stringa '*p*' non verrà più considerata come corrispondenza (come invece accadeva utilizzando l'asterisco). Per tale metacarattere valgono le stesse considerazioni sull'uso delle parentesi per far corrispondere gruppi di caratteri.

Il punto interrogativo (?)

Questo metacarattere rende opzionale il carattere che lo precede, che quindi può essere presente al più una volta, ma non di più. Spesso viene utilizzato per trovare occorrenze di parole che hanno più scritture equivalenti, ad esempio la regex:

sta per "cerca i caratteri '*ob*' seguiti eventualmente da al più una '*b*' e poi dai caratteri '*iettivo*'", ovvero cerca tutte le occorrenze delle parole 'obiettivo' e 'obbiettivo'. Come specificato precedentemente, possiamo utilizzare le parentesi per raggruppare più caratteri, facendoli seguire dal punto interrogativo:

Il punto (.)

Il metacarattere punto è l'espressione regolare che si può riassumere in: "qualsiasi carattere". Che siano spazi, numeri, simboli o lettere, il punto trova sempre un'occorrenza. Spesso il suo uso viene scoraggiato, in quanto implica un grosso sforzo computazionale da parte del motore delle espressioni regolari; nonostante ciò torna spesso utile. Ad esempio:

Questa espressione regolare cerca tutte le occorrenze della lettera '*A*' maiuscola seguita da un

qualsiasi carattere (punteggiatura, spazi, numeri, lettere), quindi possibili *match* sono:

Molto spesso però si vogliono cercare occorrenze di una lettera seguita da altre lettere, oppure occorrenze di stringhe composte da soli numeri. Ad esempio: come fare per trovare tutte le occorrenze della lettera '*A*' seguita da altre quattro (e solo) lettere? Per questo scopo si utilizzano le classi di caratteri.

Le parentesi quadre: le classi di caratteri [ ]

Le parentesi quadre permettono di specificare al loro interno un elenco di caratteri: nel testo vengono cercate occorrenze **per ognuno dei caratteri presenti tra le parentesi**. Ad esempio, nell'espressione regolare:

si cercano tutte le occorrenze della lettera *C* maiuscola oppure minuscola. Possiamo utilizzarla per eseguire delle ricerche indipendenti dal maiuscolo:

A questo punto, la soluzione del nostro problema/esempio è abbastanza banale: basta elencare tra le parentesi quadre tutte le possibili lettere dell'alfabeto, per quattro volte, facendole precedere dalla lettera '*A*':

Come puoi notare, tale scrittura risulta un po' prolissa ed è soggetta a possibili errori di battitura. Per ovviare a tale situazione è possibile effettuare delle 'abbreviazioni': si utilizza il trattino (-) separando due lettere per indicare "**considera tutti i caratteri compresi da questa lettera a questa lettera**". Ad esempio [A-Z] sta per "dalla lettera A maiuscola alla lettera Z maiuscola, considerando tutte le lettere che ci sono tra queste due". In questo modo, però, stiamo considerando le sole lettere maiuscole.

Per aggiungere anche le minuscole, si utilizza la stessa tecnica: [A-Za-z]. Quindi il nostro esempio, "trovare tutte le occorrenze della lettera '*A*' seguita da altre quattro lettere" diventa:

Questa scrittura appare però ancora troppo ripetitiva: sarebbe utile un metodo per specificare quante volte un carattere (o un gruppo di caratteri) deve essere ripetuto.

Ripetere un carattere: le parentesi graffe { }

Per **specificare quante volte un carattere deve essere presente all'interno del testo** si possono utilizzare le parentesi graffe precedute dal carattere (o dal gruppo di caratteri). Ad

esempio:

Le stringhe che verificano tale espressioni regolari saranno dunque le uniche e sole contenenti esattamente quattro A maiuscole:

A questo punto siamo in grado di risolvere il nostro problema/esempio:

che, tradotta in italiano potrebbe essere resa come: "cerca una A maiuscola seguita da una delle lettere maiuscole o minuscole, e considera questa classe di caratteri per quattro volte. Abbiamo così ottenuto una scrittura compatta e funzionale per ciò che ci eravamo proposti di cercare.

Conclusioni

In questo articolo abbiamo introdotto **le basi per la comprensione delle espressioni regolari**: è importante comprendere e assimilare bene questi concetti, in quanto l'apprendimento delle regex è di tipo incrementale. Tutti i costrutti (da quelli più semplici a quelli più complessi) hanno più o meno lo stesso funzionamento, quindi una volta capito il meccanismo di uno, potrai facilmente utilizzare anche gli altri.

Ovviemente la pratica gioca un ruolo fondamentale in questo campo, così come l'utilizzare il giusto metodo per la creazione delle regex. Trovo comodo, prima di cominciare la scrittura del codice, appuntarmi tutte le possibili variazioni che può assumere la stringa che sto cercando, magari aiutandomi con un semplice grafico. Avere una chiara visione dell'argomento che si sta trattando è quantomai necessario per utilizzare le espressioni regolari.

Nel prossimo articolo esamineremo altri costrutti fondamentali e vedremo alcune applicazioni del "mondo reale". Alla prossima!

Sommario delle lezioni

- Le basi
- I costrutti più avanzati e modelli
- Utilizzi reali delle espressioni regolari