Espressioni Regolari: i costrutti avanzati

La "cattiva" fama delle espressioni regolari è dovuta in larga parte alla sintassi oscura: esistono molte regole e abbreviazioni, alcuni simboli assumono un significato diverso a seconda della loro posizione, certi costrutti sembrano incompleti, mentre sono correttamente funzionanti.

Nella precedente lezione hai visto gli elementi base di questa sintassi. In questo articolo esamineremo alcuni usi più avanzati di vari costrutti già presentati e ne introdurremo dei nuovi. Inizieremo dalle classi di caratteri.

Le classi di caratteri: seconda parte

Nel precedente articolo abbiamo parlato delle classi di caratteri: ovvero caratteri delimitati da parentesi quadre per cui il motore delle regex cerca un'occorrenza nel testo, una per ogni carattere. Ad esempio:

Avevamo detto anche che il trattino (-) può essere utilizzato per indicare un salto da un carattere ad un altro, come in:

A causa delle differenti impostazioni presenti su alcuni computer però, queste ultime classi di caratteri non sono portabili. Questo perché, ad esempio, secondo alcune impostazioni i caratteri vengono memorizzati come "ABCD...Zabcd...z", mentre in altre i caratteri sono memorizzati come "AaBbCc...Zz". Perciò, l'espressione regolare *[A-Z]* potrebbe funzionare secondo il nostro intento su alcune macchine e fallire su altre.

Per ovviare a questo problema, sono state definite alcune classi "standard POSIX" che sono indipendenti dal sistema su cui operano, permettendo così la massima portabilità. Queste classi hanno la seguente forma: *[:nome_mnemonico:]*. Quindi parentesi quadra e due punti, seguiti da un nome, a sua volta seguito da due punti e dalla parentesi quadra chiusa. Ad esempio, una delle classi più utilizzate è *[:alnum:]* che trova le occorrenze di lettere (maiuscole e minuscole) e numeri. Dunque:

Nota bene come per utilizzare le classi POSIX **ho utilizzato due serie di parentesi quadre**: quelle più esterne indicano al motore delle regex che deve valutare una classe di caratteri, mentre quelle più interne delimitano la classe specifica. Per cui, volendo utilizzare più di una classe POSIX si avrebbe una espressione simile:

Puoi trovare l'elenco delle principali classi di caratteri POSIX nella tabella a fine articolo.

La negazione nelle classi di caratteri

Nell'articolo della scorsa settimana abbiamo visto come il carattere apice (^) serva per identificare una espressione regolare che si trovi ad inizio riga.

Quando però l'apice viene utilizzato all'interno delle classi di caratteri, assume un significato diverso: invece di cercare tutte le occorrenze di quella classe, funge da negazione, cercando quindi **"tutti i caratteri tranne quelli specificati"**. Ad esempio:

Nel primo caso, quindi, la classe ci permette di restringere il campo ai caratteri presenti all'interno delle parentesi quadre, mentre nel secondo caso decidiamo quali caratteri escludere dalla nostra ricerca. Un modo comodo per rendere in codice la frase "Tutti i caratteri tranne....".

Ovviamente questo costrutto funziona anche con le classi POSIX: per escludere tutti i numeri dalla nostra ricerca potremmo scrivere:

Questo costrutto è molto utilizzato perché permette nel 70% dei casi di sostituire il punto (che significa "ogni carattere"), che è molto dispendioso per il motore delle regex.

Le classi abbreviate

Alcune classi di caratteri sono usate così frequentemente che sono state create delle "abbreviazioni":

- **\w**: equivale alla classe [:alnum:]
- **\d**: equivale alla classe [:digit:]
- **\s**: equivale alla classe [:space:]

Per comodità sono state create delle abbreviazioni anche per le loro negazioni, utilizzando lettere maiuscole:

- **\W**: equivale alla classe [^[:alnum:]]
- **\D**: equivale alla classe [^[:digit:]]
- **\S**: equivale alla classe [^[:space:]]

A differenza però delle classi, tali abbreviazioni non necessitano delle parentesi quadre. Quindi possiamo avere la seguente espressione regolare:

Qui di seguito viene riportata una tabella riassuntiva delle principali classi di caratteri, con le relative classi POSIX e abbreviazioni:

POSIX	Abbreviazione	Classe di caratteri	Descrizione
[:alnum:]		[A-Za-z0-9]	Caratteri alfanumerici
[:word:]	\w	[A-Za-z0-9_]	Caratteri alfanumerici più underscore
	\W	[^\w]	non-word character
[:alpha:]		[A-Za-z]	Caratteri alfabetici
[:blank:]		[\t]	Spazi e tab
[:digit:]	\d	[0-9]	Numeri
	\D	[^\d]	Tutti i caratteri non numerici
[:lower:]		[a-z]	Lettere minuscole
[:punct:]		[-!"#$%&'()*+,./:;?@[\\\]^_`{|}~]	Caratteri di interpunzione
[:space:]	\s	[\t\r\n\v\f]	Tutti i caratteri di spaziatura
	\S	[^\s]	Tutti i caratteri tranne quelli di spaziatura
[:upper:]		[A-Z]	Lettere maiuscole

Delimitare le parole (\b)

Nel precedente articolo abbiamo visto come le sequenze di caratteri rappresentino una espressione regolare a tutti gli effetti. Nell'esempio abbiamo cercato la parola 'ebbe' nel testo di prova, ottenendo come risultato della ricerca effettivamente la parole 'ebbe' ma anche parole come 'avrebbe' o 'farebbe'. Molto spesso questo è un comportamento voluto, altre volte invece è necessario effetuare una ricerca sulla parola esatta, oppure su una parola che inizia per una determinata espressione regolare.

A tal scopo **si utilizza l'ancora '\b'**. Tale sequenza di caratteri può essere posta a sinistra e/o a destra di una regex e sta ad indicare rispettivamente "la parola che inizia per.." e "la parola che finisce per..". Facciamo un esempio. Per ricavare solo la parola 'ebbe' all'interno del nostro testo potremmo scrivere:

Nel nostro caso, provando tale regex sul testo di prova, sembrerebbe funzionare: vengono evidenziate tutte le parole giuste. Provando ad aggiungere la parola 'ebbene' noterai, però, che viene evidenziata. Dobbiamo quindi aggiungere anche il limite destro alla nostra espressione regolare:

Chiaramente, la sequenza di caratteri cercata è equivalente alla parola 'ebbe' e a nessun'altra, quindi il nostro compito è terminato.

Bisogna far attenzione a non confondere l'ancora '\b' con le ancore '^' e '$': la prima trova le occorrenze **all'inizio e alla fine di una parola**, le seconde lavorano **sull'inizio e sulla fine di una riga**.

Specificare il numero di occorrenze

Nella scorsa lezione abbiamo visto come le parentesi graffe poste a destra di una espressione regolare permettano di specificare quante volte essa debba ripetersi. Ad esempio:

Analizziamo l'espressione regolare appena scritta: viene utilizzata la classe abbreviata '\d' che sta per "un numero qualsiasi", dopo ogni classe viene specificato un numero tra parentesi graffe, che indica quante volte deve essere cercata la regex appena prima e le tre classi di caratteri sono separate da due trattini. Quindi potremmo renderla a parole come: *"Trova tutte le occorrenze di due numeri, seguiti da un trattino, seguiti da altri due numeri, seguiti da un trattino, seguito a sua volta da quattro numeri"*. Questo potrebbe essere dunque un buon metodo per validare una data all'interno di un form, con il formato *gg-mm-aaaa*.

Le parentesi graffe possono essere utilizzate in altri due modi:

- **{min,max}**: trova almeno 'min' occorrenze, ma non più di 'max'.
- **{min,}**: trova almeno 'min' occorrenze, senza limite superiore.

Il primo metodo serve a dare un limite inferiore e superiore alle occorrenze dell'espressione regolare:

Il secondo metodo, invece, è utilizzato per dare un limite inferiore:

Conclusioni

In questa seconda parte siamo andati più a fondo nella tratazione di alcuni costrutti base delle espressioni regolari. Tuttavia **la conoscenza del funzionamento e della sintassi di tali elementi non è una condizione sufficiente alla comprensione e all'utilizzo attivo di questo strumento**. Sono necessarie una lunga applicazione pratica e lo studio di alcuni modelli (magari realizzati da altri) per capire con profitto come si relazionano tra loro i vari elementi. Questo è proprio quello che faremo nel prossimo articolo. Alla prossima!