# Expresiones Regulares Complejas

Recientemente, mientras investigaba sobre el análisis de HTML, me encontré con una expresión regular:

```
/([\w-:\*>]*)(?:\#([\w-]+)|\.([\w-]+))?(?:\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\])?([\/, ]+)/is
```

Este es un patrón de expresión regular que se utiliza para coincidir con ciertos patrones en una cadena de texto. No se traduce, ya que es un código específico que debe mantenerse en su forma original.

Se utiliza para hacer coincidir selectores CSS, como `div > ul`.

En el pasado, he visto muchas expresiones complejas como esta, y siempre he retrocedido instintivamente. ¡Hoy voy a entenderla completamente! ¡Un hombre debe ser duro consigo mismo!

### **Coincidencia** `div > ul`

El selector `div > ul` en CSS se utiliza para seleccionar todos los elementos `<ul>` que son hijos directos de un elemento `<div>`. Esto significa que solo se seleccionarán los elementos `<ul>` que estén directamente dentro de un `<div>`, sin ningún otro elemento intermedio.

### **Ejemplo:**

```
<div>
  <ul>
    <li>Elemento 1</li>
    <li>Elemento 2</li>
  </ul>
  <div>
    <ul>
      <li>Elemento 3</li>
      <li>Elemento 4</li>
    </ul>
  </div>
</div>
```

En este ejemplo, el selector `div > ul` coincidirá con el primer `<ul>`, pero no con el segundo, ya que el segundo `<ul>` está dentro de otro `<div>` y no es un hijo directo del `<div>` principal.

**Uso en CSS:**

```
div > ul {
  background-color: yellow;
}
```

Este código aplicará un fondo amarillo solo al primer `<ul>` en el ejemplo anterior.

**Consideraciones:**

- **Especificidad**: Este selector tiene una especificidad mayor que un selector simple como `ul`, ya que incluye un elemento padre.
- **Rendimiento**: Los selectores de hijos directos (>) suelen ser más eficientes que los selectores de descendientes (espacio), ya que el navegador no necesita buscar en todos los niveles de anidamiento.

Este selector es útil cuando deseas aplicar estilos solo a elementos `<ul>` que son hijos directos de un `<div>`, excluyendo aquellos que están anidados más profundamente.

Encontré un sitio web https://regex101.com/ que permite hacer coincidencias en línea y también ofrece explicaciones.

Aunque las explicaciones de la derecha aclaran un poco, todavía no queda claro cómo se realiza la coincidencia exacta. Entonces, tomemos algunos ejemplos y analicemos cada uno.

El código específico donde aparece esta expresión regular es:

```php
$matches = [];
preg_match_all($this->pattern, trim($selector).' ', $matches, PREG_SET_ORDER);
```

`preg_match_all` significa obtener todas las cadenas que coinciden con el patrón. Si tienes:

```php
preg_match_all("abc", "abcdabc", $matches)
```

*Nota: El código PHP no se traduce, ya que es un lenguaje de programación y debe mantenerse en su forma original para que funcione correctamente.*

El primer parámetro es el patrón, el segundo parámetro es la cadena que se desea coincidir, y el tercer parámetro es la referencia al resultado. Después de ejecutar, el array `$matches` contendrá dos `abc`.

Con esta comprensión, el `div > ul` en la imagen anterior solo coincide con los primeros cuatro caracteres `div >`. ¿`regex101` no admite `preg_match_all`? Afortunadamente, basta con agregar un modificador llamado `g`:

Al agregar `g`, se buscarán todas las coincidencias, en lugar de detenerse en la primera coincidencia encontrada.

Después de agregarlo, coincidimos con `div > ul`:

A la derecha se muestra que, en la primera coincidencia, es decir, `div`, utilizamos las reglas del primer grupo para coincidir con `div`, y luego utilizamos las reglas del séptimo grupo para coincidir con el espacio .

Continuemos con la explicación del primer conjunto de reglas:

En esta larga expresión, el primer paréntesis encerrado se llama el primer grupo de reglas. Este es un grupo de captura. Los paréntesis en sí no coinciden, sino que se utilizan para agrupar. `[]` representa un conjunto de caracteres, y las reglas dentro indican cómo es ese conjunto de caracteres. Este conjunto de caracteres incluye:

- `\w` representa letras mayúsculas y minúsculas, números del 0 al 9 y el guion bajo.
- `-:` representa directamente estos dos caracteres en el conjunto.
- `\*` como `*` es un carácter reservado en expresiones regulares con un significado especial, se debe usar `\` para escaparlo, lo que indica que es un carácter `*` normal.
- `>` simplemente representa el carácter `>`.

`[\w-:\*>]*` el `*` al final indica que el carácter anterior puede aparecer 0 o más veces, pero se intentará coincidir con la mayor cantidad posible. La razón por la que coincide con `div` es porque `\w` coincide con `d`, `i`, `v`. La razón por la que no continúa coincidiendo con el espacio en blanco siguiente es porque el espacio no está presente en `[]`. Un grupo de captura significa que este grupo de coincidencias aparecerá en el array de resultados. En contraste, también existen grupos no capturadores, cuya sintaxis es `(?:)`. Si no necesitas el resultado del grupo `([\w-:\*>]*)`, puedes escribirlo como `(?:[\w-:\*>]*)`.

Entonces, si no aparece en el resultado, ¿no sería suficiente simplemente no usar paréntesis? Los paréntesis son para agrupar, y la agrupación tiene un significado importante. Puedes consultar What is a non capturing group? (?:) - StackOverflow para más información.

Después de haber explicado cómo `div` cumple con el primer conjunto de reglas, ahora vamos a hablar sobre por qué el espacio  cumple con el séptimo conjunto de reglas.

`[\/, ]` significa que coincide con cualquiera de estos cuatro caracteres, y `+` indica que la coincidencia anterior ocurre una o infinitas veces, y el número de veces debe ser lo más grande posible. Por lo tanto, como estos cuatro caracteres incluyen un espacio, coincide con nuestro espacio. Además, como el siguiente carácter después de `div` es `>`, ya no cumple con la regla del séptimo grupo, por lo que no continúa coincidiendo.

Entendí la coincidencia de `div`. Entonces, ¿por qué las reglas del segundo al sexto grupo no coincidieron con los espacios aquí, sino que los dejaron para el séptimo grupo?

Explicación de la segunda parte:

Primero, `(?:)` indica que se trata de un grupo de no captura. El `?` al final indica que la coincidencia anterior puede aparecer 0 o 1 vez. Por lo tanto, el `(?:\#([\w-]+)|\.([\w-]+))?` anterior puede estar presente o no. Al eliminar los modificadores externos, lo que queda es `\#([\w-]+)|\.([\w-]+)`, donde el `|` en el medio significa “o”, es decir, basta con que se cumpla una de las dos condiciones. En `\#([\w-]+)`, el `\#` coincide con el carácter `#`, y `[\w-]+` coincide con otros caracteres. Luego, en la segunda parte, `\.([\w-]+)`, el `\.` coincide con el carácter `.`.

Por lo tanto, los grupos del 2 al 6 pueden no cumplir con los requisitos porque el espacio no es el carácter inicial que estos grupos requieren. Además, como estos grupos tienen un modificador `?`, no es necesario que se cumplan, por lo que se salta al séptimo grupo.

El `>` que sigue a `div > ul` sigue siendo el mismo:

El primer conjunto de reglas `([\w-:\*>]*)` coincide con `>`, y el séptimo conjunto de reglas `([\/, ]+)` coincide con un espacio. Luego, `ul` funciona de manera similar a `div`.

**Coincidencia** `#answer-4185009 > table > tbody > td.answercell > div > pre`

A continuación, un selector un poco más complejo: `#answer-4185009 > table > tbody > td.answercell > div > pre` (también puedes abrir https://regex101.com/ y pegarlo allí para probarlo):

Esto es lo que se copió y pegó desde Chrome:

Primera coincidencia:

Debido a que en la primera regla del grupo `([\w-:\*>]*)`, ninguno de los caracteres dentro de `[]` puede coincidir con `#`, y debido a que el `*` al final permite coincidir 0 o más veces, en este caso es 0 veces. Luego, la descripción de la segunda regla es:

Ya se ha analizado anteriormente. Vamos directamente a `\#([\w-]+)` antes de `|`. `\#` coincide con `#`, y `[\w-]+` coincide con `answer-4185009`. En cuanto a `\.([\w-]+)` que sigue, si es `.answer-4185009`, se aplicará esta coincidencia.

A continuación, veamos la coincidencia `td.answercell`.

La primera regla del grupo `([\w-:\*>]*)` coincide con `td`, y la segunda parte de la expresión `(?:\#([\w-]+)|\.([\w-]+))?`, es decir, `\.([\w-]+)`, coincide con `.answercell`.

El análisis de este selector también llega a su fin.

### Coincidir con `a[href="http://google.com/"]`

A continuación, vamos a coincidir con el selector `a[href="http://google.com/"]`:

Echemos un vistazo al tercer bloque:

La tercera expresión es `(?:\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\])?`. Primero, el `(?:)` más externo indica que se trata de un grupo no capturador, y el `?` al final significa que todo este bloque puede coincidir 0 o 1 vez. Si lo eliminamos, queda `\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\]`. `\[` coincide con el carácter `[`. `@?` indica que el carácter `@` es opcional. Luego, el siguiente grupo `(!?[\w-:]+)`, donde `!` es opcional, y `[\w-:]+` coincide con `href`. El siguiente grupo `(?:([!*^$]?=)["']?(.*?)["']?)` es un grupo no capturador, y si eliminamos el nivel más externo, queda `([!*^$]?=)["']?(.*?)["']?`. Aquí, `([!*^$]?=)`, `[!*^$]?` indica que coincide con 0 o 1 de los caracteres dentro de `[]`. Luego, `=` coincide directamente. Después, `["']?(.*?)["']?` coincide con `"http://google.com/"`, donde `["']?` indica que coincide con `"` o `'` o ninguno de los dos. Si eliminamos este nivel más externo, queda `(.*?)` que coincide con `http://google.com/`. Aquí, `*?` indica que debe coincidir con la menor cantidad posible, es decir, si hay `"` o `'`, debe dejarse para que la expresión posterior `["']?` lo coincida. Por lo tanto, no coincidirá con `http://google.com/"`, sino solo con `http://google.com/`. Así, el selector completo `a[href="http://google.com/`

”]‘ coincide con el final de la coincidencia.

### Resumen

¡Finalmente lo entendí! Vamos a aclararlo una vez más. Primero, toda la expresión compleja `([\w-:\*>]*)(?:\#([\w-]+)|\.([\w-]+))?(?:\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\])?([\/, ]+)` se compone de cuatro partes principales:

- `([\w-:\*>]*)`

- `(?:\#([\w-]+)|\.([\w-]+))?`
- `(?:\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\])?`
- `([\/, ]+)`

Nota: Las expresiones regulares (regex) no se traducen, ya que son patrones de búsqueda específicos que no dependen del idioma. Por lo tanto, se mantienen en su forma original en inglés.

La parte más compleja, la tercera, se compone a su vez de estas secciones:

- `\[`
- `(!?[\w-:]+)`
- `(?:([!*^$]?=)["']?(.*?)["']?)?`
- `\]`

Nota: Este bloque parece ser una expresión regular escrita en formato de código. Las expresiones regulares no se traducen, ya que son patrones específicos de texto que se utilizan para buscar coincidencias en cadenas de caracteres. Por lo tanto, se mantiene el código original sin cambios.

Por lo tanto, estas partes lo suficientemente pequeñas pueden ser abordadas una por una. Luego, busca más ejemplos y observa cómo cada uno de ellos coincide, al mismo tiempo que analizas con la ayuda de https://regex101.com/. Así es como logras entender esta expresión regular que parecía tan compleja, ¡resulta que era un tigre de papel!