EXPRESIONES REGULARES

Información

7 DE JUNIO DE 2014

INSTITUTO TECNOLOGICO DE TEHUACAN
Heder Ithamar Romero Romero

Índice

Expresión regular

Una **expresión regular**, a menudo llamada también ***regex***, es una secuencia de caracteres que forma un patrón de búsqueda, principalmente utilizada para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones. Por ejemplo, el grupo formado por las cadenas *Handel*, *Händel* y *Haendel* se describe mediante el patrón "H(a | ä | ae)ndel". La mayoría de las formalizaciones proporcionan los siguientes constructores: una expresión regular es una forma de representar a los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje.

En informática, las expresiones regulares proveen una manera muy flexible de buscar o reconocer cadenas de texto.

Construcción de expresiones regulares

Específicamente, las expresiones regulares se construyen utilizando los operadores unión, concatenación y clausura de Kleene. Además cada expresión regular tiene un autómata finito asociado.

Alternación
: Una barra vertical separa las alternativas. Por ejemplo, "marrón | castaño" se corresponde con *marrón* o *castaño*.

Cuantificación
: Un cuantificador tras un carácter especifica la frecuencia con la que éste puede ocurrir. Los cuantificadores más comunes son +, ? y *:

  +
  : El signo más indica que el carácter que le precede debe aparecer al menos una vez. Por ejemplo, "ho+la" describe el conjunto infinito *hola*, *hoola*, *hooola*, *hoooola*, etcétera.

  ?
  : El signo de interrogación indica que el carácter que le precede puede aparecer como mucho una vez. Por ejemplo, "ob?scuro" se corresponde con *oscuro* y *obscuro*.

  *
  : El asterisco indica que el carácter que le precede puede aparecer cero, una, o más veces. Por ejemplo, "0*42" se corresponde con *42*, *042*, *0042*, *00042*, etcétera.

Agrupación
: Los paréntesis pueden usarse para definir el ámbito y precedencia de los demás operadores. Por ejemplo, "(p | m)adre" es lo mismo que "padre | madre", y "(des)?amor" se corresponde con *amor* y con *desamor*.

Los constructores pueden combinarse libremente dentro de la misma expresión, por lo que "H(ae? | ä)ndel" equivale a "H(a | ae | ä)ndel".

La sintaxis precisa de las expresiones regulares cambia según las herramientas y aplicaciones consideradas, y se describe con más detalle a continuación.

Su utilidad más obvia es la de describir un conjunto de cadenas, lo que resulta de utilidad en editores de texto y aplicaciones para buscar y manipular textos. Muchos lenguajes de programación admiten el uso de expresiones regulares con este fin. Por ejemplo, Perl tiene un potente motor de expresiones regulares directamente incluido en su sintaxis. Las herramientas proporcionadas por las distribuciones de Unix (incluyendo el

editor sed y el filtro grep) fueron las primeras en popularizar el concepto de expresión regular.

Aplicaciones

Numerosos editores de texto y otras utilidades utilizan expresiones regulares para buscar y reemplazar patrones en un texto.

Las expresiones regulares en programación

> *Nota: Para el entendimiento completo de esta sección es necesario poseer conocimientos generales acerca de lenguajes de programación o programación en general.*

En el área de la programación las expresiones regulares son un método por medio del cual se pueden realizar búsquedas dentro de cadenas de caracteres. Sin importar si la búsqueda requerida es de dos caracteres en una cadena de 10 o si es necesario encontrar todas las apariciones de un patrón definido de caracteres en un archivo de millones de caracteres, las expresiones regulares proporcionan una solución para el problema. Adicionalmente, un uso derivado de la búsqueda de patrones es la validación de un formato específico en una cadena de caracteres dada, como por ejemplo fechas o identificadores.

Para poder utilizar las expresiones regulares al programar es necesario tener acceso a un motor de búsqueda con la capacidad de utilizarlas. Es posible clasificar los motores disponibles en dos tipos: Motores para el programador y Motores para el usuario final.

Motores para el usuario final: son programas que permiten realizar búsquedas sobre el contenido de un archivo o sobre un texto extraído y colocado en el programa. Están diseñados para permitir al usuario realizar búsquedas avanzadas usando este mecanismo, sin embargo es necesario aprender a redactar expresiones regulares adecuadas para poder utilizarlos eficientemente. Estos son algunos de los programas disponibles:

- grep: programa de los sistemas operativos Unix/Linux.
- sed: programa de los sistemas operativos Unix/Linux que permite la modificación de la salida.
- PowerGrep: versión de grep para los sistemas operativos Windows.
- RegexBuddy: ayuda a crear las expresiones regulares en forma interactiva y luego le permite al usuario usarlas y guardarlas.

- EditPad Pro: permite realizar búsquedas con expresiones regulares sobre archivos y las muestra por medio de código de colores para facilitar su lectura y comprensión.

Motores para el programador: permiten automatizar el proceso de búsqueda de modo que sea posible utilizarlo muchas veces para un propósito específico. Estas son algunas de las herramientas de programación disponibles que ofrecen motores de búsqueda con soporte a expresiones regulares:

- *AWK*: Forma una parte esencial del lenguaje y por extensión de la herramienta awk de Unix/Linux
- *Java*: existen varias bibliotecas hechas para java que permiten el uso de RegEx, y Sun planea dar soporte a estas desde el SDK
- *JavaScript*: a partir de la versión 1.2 (ie4+, ns4+) JavaScript tiene soporte integrado para expresiones regulares.
- *Perl*: es el lenguaje que hizo crecer a las expresiones regulares en el ámbito de la programación hasta llegar a lo que son hoy en día.
- *PCRE*: biblioteca de ExReg para C, C++ y otros lenguajes que puedan utilizar bibliotecas dll (Visual Basic 6 por ejemplo).
- *PHP*: tiene dos tipos diferentes de expresiones regulares disponibles para el programador, aunque la variante POSIX (ereg) va a ser desechada en PHP 6.
- *Python*: lenguaje de **scripting** con soporte de expresiones regulares mediante su librería <regex>.
- *.Net Framework*: provee un conjunto de clases mediante las cuales es posible utilizar expresiones regulares para hacer búsquedas, reemplazar cadenas y validar patrones.

Nota: de las herramientas mencionadas con anterioridad se utilizan el EditPad Pro y el .Net Framework para dar ejemplos, aunque es posible utilizar las expresiones regulares con cualquier combinación de las herramientas mencionadas. Aunque en general las Expresiones Regulares utilizan un lenguaje común en todas las herramientas, las explicaciones prácticas acerca de la utilización de las herramientas y los ejemplos de código deben ser interpretados de forma diferente. También es necesario hacer notar que existen algunos detalles de sintaxis de las expresiones regulares que son propios del .Net Framework que se utilizan en forma diferente en las demás herramientas de programación. Cuando estos casos se den se hará notar en forma explícita para que el lector pueda buscar información respecto a estos detalles en fuentes adicionales. En el futuro se incluirán adicionalmente ejemplos de otras herramientas y lenguajes de programación.

Expresiones regulares como motor de búsqueda

Las expresiones regulares permiten encontrar porciones específicas de texto dentro de una cadena más grande de caracteres. Así, si es necesario encontrar el texto "lote" en la expresión "el ocelote saltó al lote contiguo" cualquier motor de búsqueda sería capaz de efectuar esta labor. Sin embargo, la mayoría de los motores de búsqueda encontrarían también el fragmento "lote" de la palabra "ocelote", lo cual podría no ser el resultado esperado. Algunos motores de búsqueda permiten adicionalmente especificar que se desea encontrar solamente palabras completas, solucionando este problema. Las expresiones regulares permiten especificar todas estas opciones adicionales y muchas otras sin necesidad de configurar opciones adicionales, sino utilizando el mismo texto de búsqueda como un lenguaje que permite enviarle al motor de búsqueda exactamente lo que deseamos encontrar en todos los casos, sin necesidad de activar opciones adicionales al realizar la búsqueda.

Expresiones regulares como lenguaje

Para especificar opciones dentro del texto a buscar se utiliza un lenguaje o convención mediante el cual se le transmite al motor de búsqueda el resultado que se desea obtener. Este lenguaje le da un significado especial a una serie de caracteres. Por lo tanto cuando el motor de búsqueda de expresiones regulares encuentre estos caracteres no los buscará en el texto en forma literal, sino que buscará lo que los caracteres significan. A estos caracteres se les llama algunas veces "meta-caracteres". A continuación se listan los principales meta-caracteres y su función y cómo los interpreta el motor de expresiones regulares.

Descripción de las expresiones regulares

El punto "."

El punto se interpreta por el motor de búsqueda como "cualquier carácter", es decir, busca cualquier carácter SIN incluir los saltos de línea. Los motores de Expresiones regulares tienen una opción de configuración que permite modificar este comportamiento. En .Net Framework se utiliza la opción RegexOptions.Singleline para especificar la opción de que busque todos los caracteres incluidos el salto de línea (\n).

El punto se utiliza de la siguiente forma: Si se le dice al motor de RegEx que busque "g.t" en la cadena "el gato de piedra en la gótica puerta de getisboro goot" el motor de búsqueda encontrará "gat", "gót" y por último

"get". Nótese que el motor de búsqueda no encuentra "goot"; esto es porque el punto representa un solo carácter y únicamente uno. Si es necesario que el motor encuentre también la expresión "goot", será necesario utilizar repeticiones, las cuales se explican más adelante.

Aunque el punto es muy útil para encontrar caracteres que no conocemos, es necesario recordar que corresponde a cualquier carácter y que muchas veces esto no es lo que se requiere. Es muy diferente buscar cualquier carácter que buscar cualquier carácter alfanumérico o cualquier dígito o cualquier no-dígito o cualquier no-alfanumérico. Se debe tomar esto en cuenta antes de utilizar el punto y obtener resultados no deseados.

La barra inversa o contrabarra "\"

Se utiliza para "marcar" el siguiente carácter de la expresión de búsqueda de forma que este adquiera un significado especial o deje de tenerlo. O sea, la barra inversa no se utiliza nunca por sí sola, sino en combinación con otros caracteres. Al utilizarlo por ejemplo en combinación con el punto "\." este deja de tener su significado normal y se comporta como un carácter literal.

De la misma forma, cuando se coloca la barra inversa seguida de cualquiera de los caracteres especiales que discutiremos a continuación, estos dejan de tener su significado especial y se convierten en caracteres de búsqueda literal.

Como ya se mencionó con anterioridad, la barra inversa también puede darle significado especial a caracteres que no lo tienen. A continuación hay una lista de algunas de estas combinaciones:

- \t — Representa un tabulador.
- \r — Representa el "retorno de carro" o "regreso al inicio" o sea el lugar en que la línea vuelve a iniciar.
- \n — Representa la "nueva línea" el carácter por medio del cual una línea da inicio. Es necesario recordar que en Windows es necesaria una combinación de \r\n para comenzar una nueva línea, mientras que en Unix solamente se usa \n y en Mac_OS clásico se usa solamente \r.
- \a — Representa una "campana" o "beep" que se produce al imprimir este carácter.
- \e — Representa la tecla "Esc" o "Escape"
- \f — Representa un salto de página
- \v — Representa un tabulador vertical

- \x — Se utiliza para representar caracteres ASCII o ANSI si conoce su código. De esta forma, si se busca el símbolo de derechos de autor y la fuente en la que se busca utiliza el conjunto de caracteres Latin-1 es posible encontrarlo utilizando "\xA9".
- \u — Se utiliza para representar caracteres Unicode si se conoce su código. "\u00A2" representa el símbolo de centavos. No todos los motores de Expresiones Regulares soportan Unicode. El .Net Framework lo hace, pero el EditPad Pro no, por ejemplo.
- \d — Representa un dígito del 0 al 9.
- \w — Representa cualquier carácter alfanumérico.
- \s — Representa un espacio en blanco.
- \D — Representa cualquier carácter que no sea un dígito del 0 al 9.
- \W — Representa cualquier carácter no alfanumérico.
- \S — Representa cualquier carácter que no sea un espacio en blanco.
- \A — Representa el inicio de la cadena. No un carácter sino una posición.
- \Z — Representa el final de la cadena. No un carácter sino una posición.
- \b — Marca el inicio y el final de una palabra.
- \B — Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos.

Notas:

- Utilidades como Charmap.exe de Windows o gucharmap de GNOME permiten encontrar los códigos ASCII/ANSI/UNICODE para utilizarlos en Expresiones Regulares.
- Algunos lenguajes, como Java, asignan su propio significado a la barra invertida, por lo que deberá repetirse para que sea considerada una expresión regular (ej. String expresion="\\d.\\d" para indicar el patrón \d.\d).

Los corchetes "[ ]"

La función de los corchetes en el lenguaje de las expresiones regulares es representar "clases de caracteres", o sea, agrupar caracteres en grupos o clases. Son útiles cuando es necesario buscar uno de un grupo de caracteres. Dentro de los corchetes es posible utilizar el guion "-" para especificar rangos de caracteres. Adicionalmente, los metacaracteres pierden su significado y se convierten en literales cuando se encuentran dentro de los corchetes. Por ejemplo, como vimos en la entrega anterior "\d" nos es útil para buscar cualquier carácter que represente un dígito. Sin embargo esta denominación no incluye el punto "." que divide la parte

decimal de un número. Para buscar cualquier carácter que representa un dígito o un punto podemos utilizar la expresión regular "[\d.]". Como se hizo notar anteriormente, dentro de los corchetes, el punto representa un carácter literal y no un metacarácter, por lo que no es necesario antecederlo con la barra inversa. El único carácter que es necesario anteceder con la barra inversa dentro de los corchetes es la propia barra inversa. La expresión regular "[\dA-Fa-f]" nos permite encontrar dígitos hexadecimales. Los corchetes nos permiten también encontrar palabras aún si están escritas de forma errónea, por ejemplo, la expresión regular "expresi[oó]n" permite encontrar en un texto la palabra "expresión" aunque se haya escrito con o sin tilde. Es necesario aclarar que sin importar cuantos caracteres se introduzcan dentro del grupo por medio de los corchetes, el grupo sólo le dice al motor de búsqueda que encuentre un solo carácter a la vez, es decir, que "expresi[oó]n" no encontrará "expresioon" o "expresioón".

La barra "|"

Sirve para indicar una de varias opciones. Por ejemplo, la expresión regular "a | e" encontrará cualquier "a" o "e" dentro del texto. La expresión regular "este | oeste | norte | sur" permitirá encontrar cualquiera de los nombres de los puntos cardinales. La barra se utiliza comúnmente en conjunto con otros caracteres especiales.

El signo de dólar "$"

Representa el final de la cadena de caracteres o el final de la línea, si se utiliza el modo multi-línea. No representa un carácter en especial sino una posición. Si se utiliza la expresión regular "\.$" el motor encontrará todos los lugares donde un punto finalice la línea, lo que es útil para avanzar entre párrafos.

El acento circunflejo "^"

Este carácter tiene una doble funcionalidad, que difiere cuando se utiliza individualmente y cuando se utiliza en conjunto con otros caracteres especiales. En primer lugar su funcionalidad como carácter individual: el carácter "^" representa el inicio de la cadena (de la misma forma que el signo de dólar "$" representa el final de la cadena). Por tanto, si se utiliza la expresión regular "^[a-z]" el motor encontrará todos los párrafos que den inicio con una letra minúscula. Cuando se utiliza en conjunto con los corchetes de la siguiente forma "[^\w]" permite encontrar cualquier carácter que NO se encuentre dentro del grupo indicado. La expresión indicada permite encontrar, por ejemplo, cualquier carácter que no sea

alfanumérico o un espacio, es decir, busca todos los símbolos de puntuación y demás caracteres especiales.

La utilización en conjunto de los caracteres especiales "^" y "$" permite realizar validaciones en forma sencilla. Por ejemplo "^\d$" permite asegurar que la cadena a verificar representa un único dígito "^\d\d/\d\d/\d\d\d\d$" permite validar una fecha en formato corto, aunque no permite verificar si es una fecha válida, ya que 99/99/9999 también sería válido en este formato; la validación completa de una fecha también es posible mediante expresiones regulares, como se ejemplifica más adelante.

Los paréntesis "()"

De forma similar que los corchetes, los paréntesis sirven para agrupar caracteres, sin embargo existen varias diferencias fundamentales entre los grupos establecidos por medio de corchetes y los grupos establecidos por paréntesis:

- Los caracteres especiales conservan su significado dentro de los paréntesis.
- Los grupos establecidos con paréntesis establecen una "etiqueta" o "punto de referencia" para el motor de búsqueda que puede ser utilizada posteriormente como se denota más adelante.
- Utilizados en conjunto con la barra "|" permite hacer búsquedas opcionales. Por ejemplo la expresión regular "al (este|oeste|norte|sur) de" permite buscar textos que den indicaciones por medio de puntos cardinales, mientras que la expresión regular "este|oeste|norte|sur" encontraría "este" en la palabra "esteban", no pudiendo cumplir con este propósito.
- Utilizados en conjunto con otros caracteres especiales que se detallan posteriormente, ofrece funcionalidad adicional.

El signo de interrogación "?"

El signo de interrogación tiene varias funciones dentro del lenguaje de las expresiones regulares. La primera de ellas es especificar que una parte de la búsqueda es opcional. Por ejemplo, la expresión regular "ob?scuridad" permite encontrar tanto "oscuridad" como "obscuridad". En conjunto con los paréntesis redondos permite especificar que un conjunto mayor de caracteres es opcional; por ejemplo "Nov(\.|iembre|ember)?" permite encontrar tanto "Nov" como "Nov.", "Noviembre" y "November". Como se mencionó anteriormente, los paréntesis nos permiten establecer un "punto de referencia" para el motor de búsqueda. Sin embargo, algunas veces, no

se desea utilizarlos con este propósito, como en el ejemplo anterior "Nov(\.|iembre|ember)?". En este caso el establecimiento de este punto de referencia (que se detalla más adelante) representa una inversión inútil de recursos por parte del motor de búsqueda. Para evitarlo se puede utilizar el signo de pregunta de la siguiente forma:
"Nov(?:\.|iembre|ember)?". Aunque el resultado obtenido será el mismo, el motor de búsqueda no realizará una inversión inútil de recursos en este grupo, sino que lo ignorará. Cuando no sea necesario reutilizar el grupo, es aconsejable utilizar este formato. De forma similar, es posible utilizar el signo de pregunta con otro significado: Los paréntesis definen grupos "anónimos", sin embargo el signo de pregunta en conjunto con los paréntesis triangulares "<>" permite "nombrar" estos grupos de la siguiente forma: "^(?<Día>\d\d)/(?<Mes>\d\d)/(?<Año>\d\d\d\d)$"; Con lo cual se le especifica al motor de búsqueda que los primeros dos dígitos encontrados llevarán la etiqueta "Día", los segundos la etiqueta "Mes" y los últimos cuatro dígitos llevarán la etiqueta "Año".

NOTA: a pesar de la complejidad y flexibilidad dada por los caracteres especiales estudiados hasta ahora, en su mayoría nos permiten encontrar solamente un carácter a la vez, o un grupo de caracteres a la vez. Los metacaracteres enumerados en adelante permiten establecer repeticiones.

Las llaves "{}"

Comúnmente las llaves son caracteres literales cuando se utilizan por separado en una expresión regular. Para que adquieran su función de metacaracteres es necesario que encierren uno o varios números separados por coma y que estén colocados a la derecha de otra expresión regular de la siguiente forma: "\d{2}" Esta expresión le dice al motor de búsqueda que encuentre dos dígitos contiguos. Utilizando esta fórmula podríamos convertir el ejemplo "^\d\d/\d\d/\d\d\d\d$" que servía para validar un formato de fecha en "^\d{2}/\d{2}/\d{4}$" para una mayor claridad en la lectura de la expresión.

"\d{2,4}" Esta forma añade un segundo número separado por una coma, el cuál indica al motor de búsqueda que como máximo debe aparecer 4 veces la expresión regular \d. Los posibles valores son:

- "^\d\d$" (mínimo 2 repeticiones)
- "^\d\d\d$"(tiene 3 repeticiones, por lo tanto entra en el rango 2-4)
- "^\d\d\d\d$" (máximo 4 repeticiones)

Nota: aunque esta forma de encontrar elementos repetidos es muy útil, algunas veces no se conoce con claridad cuantas veces se repite lo que se busca o su grado de repetición es variable. En estos casos los siguientes metacaracteres son útiles.

El asterisco "*"

El asterisco sirve para encontrar algo que se encuentra repetido 0 o más veces. Por ejemplo, utilizando la expresión "[a-zA-Z]\d*" será posible encontrar tanto "H" como "H1", "H01", "H100" y "H1000", es decir, una letra seguida de un número indefinido de dígitos. Es necesario tener cuidado con el comportamiento del asterisco, ya que éste, por defecto, trata de encontrar la mayor cantidad posible de caracteres que correspondan con el patrón que se busca. De esta forma si se utiliza "\(.*\)" para encontrar cualquier cadena que se encuentre entre paréntesis y se lo aplica sobre el texto "Ver (Fig. 1) y (Fig. 2)" se esperaría que el motor de búsqueda encuentre los textos "(Fig. 1)" y "(Fig. 2)", sin embargo, debido a esta característica, en su lugar encontrará el texto "(Fig. 1) y (Fig. 2)". Esto sucede porque el asterisco le dice al motor de búsqueda que llene todos los espacios posibles entre los dos paréntesis. Para obtener el resultado deseado se debe utilizar el asterisco en conjunto con el signo de interrogación de la siguiente forma: "\(.*?\)" Esto es equivalente a decirle al motor de búsqueda que "Encuentre un paréntesis de apertura y luego encuentre cualquier secuencia de caracteres hasta que encuentre un paréntesis de cierre".

El signo de suma "+"

Se utiliza para encontrar una cadena que se encuentre repetida una o más veces. A diferencia del asterisco, la expresión "[a-zA-Z]\d+" encontrará "H1" pero no encontrará "H". También es posible utilizar este metacarácter en conjunto con el signo de interrogación para limitar hasta donde se efectúa la repetición.

Grupos anónimos

Los grupos anónimos se establecen cada vez que se encierra una expresión regular en paréntesis, por lo que la expresión "<([a-zA-Z]\w*?)>" define un grupo anónimo que tendrá como resultado que el motor de búsqueda almacenará una referencia al texto que corresponda a la expresión encerrada entre los paréntesis.

La forma más inmediata de utilizar los grupos que se definen es dentro de la misma expresión regular, lo cual se realiza utilizando la barra inversa "\"

seguida del número del grupo al que se desea hacer referencia de la siguiente forma: "<([a-zA-Z]\w*?)>.*?</\1>" Esta expresión regular encontrará tanto la cadena "Esta" como la cadena "prueba" en el texto "Esta es una prueba" a pesar de que la expresión no contiene los literales "font" y "B".

Otra forma de utilizar los grupos es en el lenguaje de programación que se esté utilizando. Cada lenguaje tiene una forma distinta de acceder a los grupos. Los ejemplos enumerados a continuación utilizan las clases del .Net Framework, usando la sintáxis de C# (la cual puede fácilmente adaptarse a VB .Net o cualquier otro lenguaje del Framework o incluso Java o JavaScript).

Para utilizar el motor de búsqueda del .Net Framework es necesario en primer lugar hacer referencia al espacio de nombres System.Text.RegularExpressions. Luego es necesario declarar una instancia de la clase Regex de la siguiente forma:

```
Regex _TagParser = new Regex("<([a-zA-Z]\w*?)>");
```

Luego asumiendo que el texto que se desea examinar con la expresión regular se encuentra en la variable "sText" podemos recorrer todas las instancias encontradas de la siguiente forma:

```
foreach(Match CurrentMatch in _TagParser.Matches(sText)){
  // ----- Código extra aquí -----
}
```

Luego se puede utilizar la propiedad Groups de la clase Match para traer el resultado de la búsqueda:

```
foreach(Match CurrentMatch in _TagParser.Matches(sText)){
  String sTagName = CurrentMatch. Groups[1].Value;
}
```

Grupos nominales

Los grupos nominales son aquellos a los que se les asigna un nombre, dentro de la expresión regular para poder utilizarlos posteriormente. Esto se hace de forma diferente en los distintos motores de búsqueda, a continuación se explica como hacerlo en el motor del .Net Framework.

Utilizando el ejemplo anterior es posible convertir "<([a-zA-Z]\w*?)>" en "<(?<TagName>[a-zA-Z]\w*?)>" Para encontrar etiquetas HTML. Nótese el signo de pregunta y el texto "TagName" encerrado entre paréntesis triangulares, seguido de éste. Para utilizar este ejemplo en el .Net Framework es posible utilizar el siguiente código:

```
Regex _TagParser = new Regex("<(?<TagName>[a-zA-Z]\w*?)>");
foreach(Match CurrentMatch in _TagParser.Matches(sText)){
  String sTagName = CurrentMatch. Groups["TagName"]. Value;
}
```

Es posible definir tantos grupos como sea necesario, de esta forma se puede definir algo como: "<(?<TagName>[a-zA-Z]\w*?) ?(?<Attributes>.*?)>" para encontrar no solo el nombre del tag HTML sino también sus atributos de la siguiente forma:

```
Regex _TagParser = new Regex("<(?<TagName>[a-zA-Z]\w*?) ?(?<Attributes>.*?)>");
foreach(Match CurrentMatch in _TagParser.Matches(sText)){
  String sTagName = CurrentMatch. Groups["TagName"]. Value;
  String sAttributes = CurrentMatch. Groups["Attributes"]. Value;
}
```

Pero es posible ir mucho más allá de la siguiente forma:

```
"<?(?<TagName>[a-zA-Z][\w\r\n]*?) ?(?:(?<Attribute>[\w-\r\n]*?)='?"?(?<Value>[\w-::,\./= \r\n]*?)'?"? ?)>"
```

Esta expresión permite encontrar el nombre de la etiqueta, el nombre del atributo y su valor.

Sin embargo, una etiqueta HTML puede tener más de un atributo. Este puede resolverse utilizando repeticiones de la siguiente forma:

```
"<?(?<TagName>[a-zA-Z][\w\r\n]*?) ?(?:(?<Attribute>[\w-\r\n]*?)='?"?(?<Value>[\w-::,\./= \r\n]*?)'?"? ?)*?>"
```

Y en el código puede utilizarse de la siguiente forma:

```
Regex _TagParser =
  new Regex("<?(?<TagName>[a-zA-Z][\w\r\n]*?)?
  (?:(?<Attribute>[\w-\r\n]*?)='?"?
  (?<Value>[\w-::,\./= \r\n]*?)'?"? ?)*?>");
foreach(Match CurrentMatch in _TagParser.Matches(sText)){
  String sTagName = CurrentMatch. Groups["TagName"]. Value;
  foreach(Capture CurrentCapture in CurrentMatch. Groups["Attribute"]. Captures){
    AttributesCollection. Add(CurrentCapture. Value)
  }
  foreach(Capture CurrentCapture in CurrentMatch. Groups["value"]. Captures){
```

```
    ValuesCollection. Add(CurrentCapture. Value)
  }
 }
```

Es posible profundizar utilizando una expresión regular como esta:

```
  "<?(?<TagName>[a-zA-Z][\w\r\n]*?) ?(?:(?<Attribute>[\w-\r\n]*?)='?"?(?<Value>[\w-
::,\./= \r\n]*?)'?"? ?)*?>(?<Content>.*?)</\1>"
```

La cual permitiría encontrar el nombre de la etiqueta, sus atributos, valores y el contenido de esta, todo con una sola expresión regular.