深入理解正则表达式 - 向北方 - 博客园(1)

一 前言

对于正则表达式，相信很多人都知道，但是很多人的第一感觉就是难学，因为看第一眼时，觉得完全没有规律可寻，而且全是一堆各种各样的特殊符号，完全不知所云。

其实只是对正则不了解而以，了解了你就会发现，原来就这样啊正则所用的相关字符其实不多，也不难记，更不难懂，唯一难的就是组合起来之后，可读性比较差，而且不容易理解，本文旨在让大家对正则有一个基本的了解，能看得懂简单的正则表达式，写得出简单的正则表达式，用以满足日常开发中的需求即可。

0\d{2}-\d{8}|0\d{3}-\d{7} 先来一段正则，如果你对正则不了解，是不是完全不知道这一串字符是什么意思？这不要紧文章会详细解释每个字符的含义的。

1.1 什么是正则表达式

正则表达式是一种特殊的字符串模式，用于匹配一组字符串，就好比用模具做产品，而正则就是这个模具，定义一种规则去匹配符合规则的字符。

1.2 常用的正则匹配工具

在线匹配工具：

1 http://www.regexpal.com/

2 http://rubular.com/

正则匹配软件

McTracer

用过几个之后还是觉得这个是最好用的，支持将正则导成对应的语言如java C# js等还帮你转义了，Copy直接用就行了很方便，另外支持把正则表达式用法解释，如哪一段是捕获分组，哪段是贪婪匹配等等，总之用起来 So Happy .

二 正则字符简单介绍

2.1 元字符介绍

"^" ：^会匹配行或者字符串的起始位置，有时还会匹配整个文档的起始位置。

**"$"** ：$会匹配行或字符串的结尾

如图

而且被匹配的字符必须是以This开头有空格也不行，必须以Regex结尾，也不能有空格与其它字符

"\b" :不会消耗任何字符只匹配一个位置，常用于匹配单词边界 如 我想从字符串中"This is Regex"匹配单独的单词 "is" 正则就要写成 "\bis\b"

\b 不会匹配is 两边的字符，但它会识别is 两边是否为单词的边界

"\d": 匹配数字，

例如要匹配一个固定格式的电话号码以0开头前4位后7位，如0737-5686123 正则:^0\d\d\d-\d\d\d\d\d\d\d$ 这里只是为了介绍"\d"字符，实际上有更好的写法会在 下面介绍。

"\w"：匹配字母，数字，下划线.

例如我要匹配"a2345BCD__TTz" 正则："\w+" 这里的"+"字符为一个量词指重复的次数，稍后会详细介绍。

"\s"：匹配空格

例如字符 "a b c" 正则："\w\s\w\s\w" 一个字符后跟一个空格，如有字符间有多个空格直接把"\s" 写成 "\s+" 让空格重复

"."：匹配除了换行符以外的任何字符

这个算是"\w"的加强版了"\w"不能匹配 空格 如果把字符串加上空格用"\w"就受限了，看下用 "."是如何匹配字符"a23 4 5 B C D__TTz" 正则：".+"

"[abc]": 字符组 匹配包含括号内元素的字符

这个比较简单了只匹配括号内存在的字符，还可以写成[a-z]匹配a至z的所以字母就等于可以用来控制只能输入英文了，

2.2 几种反义

写法很简单改成大写就行了，意思与原来的相反，这里就不举例子了

"\W"匹配任意不是字母，数字，下划线 的字符

"\S"匹配任意不是空白符的字符

"\D"匹配任意非数字的字符

"\B"匹配不是单词开头或结束的位置

"[^abc]"匹配除了abc以外的任意字符

2.3 量词

先解释关于量词所涉及到的重要的三个概念

贪婪(贪心) 如"*"字符 贪婪量词会首先匹配整个字符串，尝试匹配时，它会选定尽可能多的内容，如果 失败则回退一个字符，然后再次尝试回退的过程就叫做回溯，它会每次回退一个字符，直到找到匹配的内容或者没有字符可以回退。相比下面两种贪婪量词对资源的消耗是最大的，

懒惰(勉强) 如 "?" 懒惰量词使用另一种方式匹配，它从目标的起始位置开始尝试匹配，每次检查一个字符，并寻找它要匹配的内容，如此循环直到字符结尾处。

占有 如"+" 占有量词会覆盖事个目标字符串，然后尝试寻找匹配内容 ，但它只尝试一次，不会回溯，就好比先抓一把石头，然后从石头中挑出黄金

"*"(贪婪) 重复零次或更多

例如"aaaaaaaa" 匹配字符串中所有的a 正则： "a*" 会出到所有的字符"a"

"+"(懒惰) 重复一次或更多次

例如"aaaaaaaa" 匹配字符串中所有的a 正则： "a+" 会取到字符中所有的a字符， "a+"与"a*"不同在于"+"至少是一次而"*" 可以是0次，

稍后会与"?"字符结合来体现这种区别

"?"(占有) 重复零次或一次

例如"aaaaaaaa" 匹配字符串中的a 正则 ： "a?" 只会匹配一次，也就是结果只是单个字符a

"{n}" 重复n次

例如从"aaaaaaaa" 匹配字符串的a 并重复3次 正则： "a{3}" 结果就是取到3个a字符 "aaa";

"{n,m}" 重复n到m次

例如正则 "a{3,4}" 将a重复匹配3次或者4次 所以供匹配的字符可以是三个"aaa"也可以是四个"aaaa" 正则都可以匹配到

"{n,}" 重复n次或更多次

与{n,m}不同之处就在于匹配的次数将没有上限，但至少要重复n次 如 正则"a{3,}" a至少要重复3次

把量词了解了之后之前匹配电话号码的正则现在就可以改得简单点了^0\d\d\d-\d\d\d\d\d\d\d" role="presentation" style='font-size:13px;font-style:normal;font-weight:normal;font-family:verdana, sans-serif;color:rgb(64, 64, 64);'>可以改为"0\d+−\d7"。

这样写还不够完美如果因为前面的区号没有做限定，以至于可以输入很多们，而通常只能是3位或者4位，

现在再改一下 "^0\d{2,3}-\d{7}"如此一来区号部分就可以匹配3位或者4位的了

2.4 懒惰限定符

"*?" 重复任意次，但尽可能少重复

如 "acbacb" 正则 "a.*?b" 只会取到第一个"acb" 原本可以全部取到但加了限定符后，只会匹配尽可能少的字符 ，而"acbacb"最少字符的结果就是"acb"

"+?" 重复1次或更多次，但尽可能少重复

与上面一样，只是至少要重复1次

"??" 重复0次或1次，但尽可能少重复

如 "aaacb" 正则 "a.??b" 只会取到最后的三个字符"acb"

"{n,m}?" 重复n到m次，但尽可能少重复

如 "aaaaaaaa" 正则 "a{0,m}" 因为最少是0次所以取到结果为空

"{n,}?" 重复n次以上，但尽可能少重复

如 "aaaaaaa" 正则 "a{1,}" 最少是1次所以取到结果为 "a"

三 正则进阶

3.1 捕获分组

先了解在正则中捕获分组的概念，其实就是一个括号内的内容 如 "(\d)\d" 而"(\d)" 这就是一个捕获分组，可以对捕获分组进行 后向引用 (如果后而有相同的内容则可以直接引用前面定义的捕获组，以简化表达式) 如(\d)\d\1 这里的"\1"就是对"(\d)"的后向引用

那捕获分组有什么用呢看个例子就知道了

如 "zery zery" 正则 \b(\w+)\b\s\1\b 所以这里的"\1"所捕获到的字符也是 与(\w+)一样的"zery"，为了让组名更有意义，组名是可以自定义名字的

"\b(?\w+)\b\s\k\b" 用"?"就可以自定义组名了而要后向引用组时要记得写成 "\k";自定义组名后,捕获组中匹配到的值就会保存在定义的组名里

下面列出捕获分组常有的用法

"(exp)"匹配exp,并捕获文本到自动命名的组里

"(?exp)"匹配exp,并捕获文本到名称为name的组里

"(?:exp)"匹配exp,不捕获匹配的文本，也不给此分组分配组号

以下为零宽断言

"(?=exp)"匹配exp前面的位置

如 "How are you doing" 正则"(?.+(?=ing))" 这里取ing前所有的字符，并定义了一个捕获分组名字为 "txt" 而"txt"这个组里的值为"How are you do";

"(?<=exp)"匹配exp后面的位置

如 "How are you doing" 正则"(?(?<=How).+)" 这里取"How"之后所有的字符，并定义了一个捕获分组名字为 "txt" 而"txt"这个组里的值为" are you doing";

"(?!exp)"匹配后面跟的不是exp的位置

如 "123abc" 正则 "\d{3}(?!\d)"匹配3位数字后非数字的结果

"(?匹配前面不是exp的位置

如 "abc123 " 正则 "(?<\d)123"

四 正则实战

正则在做验证，与数据过滤时体现的威力是巨大的，我想用过的朋友都知道，下面我们把刚刚了解的全部结合起来做一次实战 做数据采集用正则过滤Html标签并取相应的数据

我们的战场就选在博客园吧，假设现在要采集博客园首页的所有文章信息 包括文章标题，链接接 作者博客地址，文章简介，文章发布时间，阅读数据，评论数 ，推荐数。

先看博客园文章的Html格式


```
<div class="post_item"><div class="digg"><div class="diggit" onclick="DiggIt(3439076,120879,1)"><span class="diggnum" id="digg_count_3439076">4span>div><div class="clear">div><div id="digg_tip_3439076" class="digg_tip">div>div><div class="post_item_body"><h3><a class="titlelnk" href="http://www.cnblogs.com/swq6413/p/3439076.html" target="_blank">分享完整的项目工程目录结构a>h3><p class="post_item_summary"><a href="http://www.cnblogs.com/swq6413/" target="_blank"><img width="48" height="48" class="pfs" src="http://pic.cnitblog.com/face/142964/20131116170946.png" alt=""/>a>  在项目开发过程中，如何有序的保存项目中的各类数据文件，建立一个分类清晰、方便管理的目录结构是非常重要的。 综合以前的项目和一些朋友的项目结构，我整理了一份我觉得还不错的项目目录结构。 在这里分享给大家，欢迎各位提出你宝贵的意见和建议。如果喜欢请"推荐"则个，感激万分！！ 整个目录设置到4级子目录，实...
p><div class="post_item_foot"><a href="http://www.cnblogs.com/swq6413/" class="lightblue">七少爷a>  发布于 2013-11-23 15:48 <span class="article_comment"><a href="http://www.cnblogs.com/swq6413/p/3439076.html#commentform" title="2013-11-23 16:40" class="gray">  评论(4)a>span><span class="article_view"><a href="http://www.cnblogs.com/swq6413/p/3439076.html" class="gray">阅读(206)a>span>div>div><div class="clear">div>div>
```

通过构造一个Http请求来取到数据并对数据进行相应处理得到关键信息，在过滤Html标签取文章时正则的强大的威力就体现出来了，正则的知识点也都基本用上了比如 "\s \w+ . * ? "还有捕获分组，零宽断言等等。喜欢的朋友可以试一试，然后自己看如何通过正则取相应数据的，代码中的正则都是很基本简单的，其意思与用法都在上文中详细写了。

```
class Program  {  staticvoid Main(string[] args)  {  string content = HttpUtility.HttpGetHtml(); HttpUtility.GetArticles(content);  }  }  internalclass HttpUtility  {  //默认获取第一页数据publicstaticstring HttpGetHtml()  {  HttpWebRequest request = (HttpWebRequest)WebRequest.Create("http://www.cnblogs.com/");  request.Accept = "text/plain, */*; q=0.01";  request.Method = "GET";  request.Headers.Add("Accept-Language", "zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3");  request.ContentLength = 0;  request.Host = "www.cnblogs.com"; request.UserAgent = "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.1 (KHTML, like Gecko) Maxthon/4.1.3.5000 Chrome/26.0.1410.43 Safari/537.1";  HttpWebResponse response = (HttpWebResponse)request.GetResponse(); Stream responStream = response.GetResponseStream();  StreamReader reader = new StreamReader(responStream, Encoding.UTF8);  string content = reader.ReadToEnd();  return content;  }  publicstatic List
 GetArticles(string htmlString)  {  List
 articleList = new List
(); Regex regex = null;  Article article = null;  regex = new Regex("
(?
" + @"
\s*
)", RegexOptions.Singleline); if (regex.IsMatch(htmlString)) { MatchCollection aritcles = regex.Matches(htmlString); foreach (Match item in aritcles) { article = new Article(); //取推荐 regex = new Regex( "
.*(?.*)
" + @"" + ".*
", RegexOptions.Singleline); article.DiggNum = regex.Match(item.Value).Groups["digNum"].Value; //取文章标题 需要去除转义字符 regex = new Regex("
(?.*)
", RegexOptions.Singleline); string a = regex.Match(item.Value).Groups["a"].Value; regex = new Regex(".*?)\".*>(?
.*)", RegexOptions.Singleline); article.AritcleUrl = regex.Match(a).Groups["href"].Value; article.AritcleTitle = regex.Match(a).Groups["summary"].Value; //取作者图片 regex = new Regex("(?].*>)", RegexOptions.Singleline); article.AuthorImg = regex.Match(item.Value).Groups["img"].Value; //取作者博客URL及链接的target属性 regex = new Regex(".*)\"\\s*?target=\"(?.*?)\">.*", RegexOptions.Singleline); article.AuthorUrl = regex.Match(item.Value).Groups["href"].Value; string urlTarget = regex.Match(item.Value).Groups["target"].Value; //取文章简介 //1 先取summary Div中所有内容 regex = new Regex("
(?
.*)
", RegexOptions.Singleline); string summary = regex.Match(item.Value).Groups["summary"].Value; //2 取简介 regex = new Regex("(?(?<=).*)", RegexOptions.Singleline); article.AritcleInto = regex.Match(summary).Groups["indroduct"].Value; //取发布人与发布时间 regex = new Regex( "
```

```
\\s*(?.*)(?.*)", RegexOptions.Singleline); article.Author = regex.Match(item.Value).Groups["publishName"].Value; article.PublishTime = regex.Match(item.Value).Groups["publishTime"].Value.Trim(); //取评论数 regex = new Regex( "(?.*)", RegexOptions.Singleline); article.CommentNum = regex.Match(item.Value).Groups["comment"].Value; //取阅读数 regex = new Regex("(?.*)", RegexOptions.Singleline); article.ReadNum = regex.Match(item.Value).Groups["readNum"].Value; articleList.Add(article); } } return articleList; } publicstaticstring ClearSpecialTag(string htmlString) { string htmlStr = Regex.Replace(htmlString, "\n", "", RegexOptions.IgnoreCase); htmlStr = Regex.Replace(htmlStr, "\t", "", RegexOptions.IgnoreCase); htmlStr = Regex.Replace(htmlStr, "\r", "", RegexOptions.IgnoreCase); htmlStr = Regex.Replace(htmlStr, "\"", "'", RegexOptions.IgnoreCase); return htmlStr; } } publicclass Article { ///
/// 文章标题 ///
publicstring AritcleTitle { get; set; } ///
/// 文章链接 ///
publicstring AritcleUrl { get; set; } ///
/// 文章简介 ///
publicstring AritcleInto { get; set; } ///
/// 作者名 ///
publicstring Author { get; set; } ///
/// 作者地址 ///
publicstring AuthorUrl { get; set; } ///
/// 作者图片 ///
publicstring AuthorImg { get; set; } ///
/// 发布时间 ///
publicstring PublishTime { get; set; } ///
/// 推荐数 ///
publicstring DiggNum { get; set; } ///
/// 评论数 ///
publicstring CommentNum { get; set; } ///
/// 阅读数 ///
publicstring ReadNum { get; set; } }
```

正则部分可能写得不很完美，但至少也匹配出来了，另外因为自己也是刚接触正则，也只能写出这种比较简单的正则。还望大家海涵~~

五　总结

正则其实并不难，了解每个符号的意思后，自己马上动手试一试多写几次自然就明白了，正则是出了名的坑多，随便少写了个点就匹配不到数据了，我也踩了很多坑，踩着踩着就踩出经验了。

本文也只是对正则做了很基本的介绍，还有很多正则的字符没有介绍，只是写了比较常用的一些。如有错误之处，还望在评论中指出，我会马上修改。