正则表达式

虚幻引擎交互开发工程师

火星时代教育

第一部分

概念

火星时代游戏设计学院

检查字符串是否是邮箱地址

概念

软件设计中，我们经常会遇到这样的问题：检查给定的字符串是否匹配格式需求，例如，检查给定的字符串，是否匹配邮箱地址格式，给定的字符串，是否匹配电话号码格式。还有就是检查给定的字符串中是否包含给定内容。

从描述中我们得知，这是典型的格式校验匹配需求，**正则表达式的设计目的就是为了检索给定的串是否匹配某种规则子串，替换或是获取符合规则的子串**。

正则表达式，又称规则表达式。（英语：Regular Expression，在代码中常简写为regex、regexp或RE），计算机科学的一个概念。正则表达式通常被用来检索、替换那些符合某个模式(规则)的文本。

正则表达式是对**字符串操作的一种逻辑公式**，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

正则表达式是一种字符串校验检查表达式，与语言无关，大部分编程语言中都有正则表达式库

既定检查

一般我们检查字符串的方法，都是通过恒等的方式完成，例如，检查字符串“abc”是不是“abc”，可以通过“abc”==“abc”完成。

在正则表达式中，我们也可以通过这种方式进行检查。但是实际是没有意义的，你可以理解检查字符串使用的串逻辑完全相同。

如果我们换一种需求，检查给定的字符串中是否包含“ab”。对于“abc”“abe”“1ab”，“asdabe”“kcab@”都是符合要求的。如果增加条件，例如：ab必须处于开头，则只有1，2字符串符合要求。其实这就是**字符串规则**。

正则表达式

例如检查电话号码的规则

^1\d{10}$ 这是一个不精准的电话号码检查规则（pattern），符号^代表开头，\d代表数字，{10}代表前面的元素出现的次数，$代表结尾

如果我们希望检查的是精准的，我们需要考虑国内电话运营商的前三位组成格式，例如131，132，133，134，135，136，137，138，139，150，151，152，153...很多，都需要进行匹配。

Pattern

关键字pattern较常出现在正则检查中，用于记录字符串规则逻辑。

规则本身也是串，是对需要查询或是检索的字符串的一种**规则描述**。它可也是具体的，例如查询abc，规则也设定为abc。当然也可以是模糊的，例如只要带有ab的串就被认为是对的。那么我们如何去描述第二种串规则呢？这就是正则表达式要做的事情了。

第二部分

语法

火星时代游戏设计学院

语法

正则表达式中设置了很多**基础表达式**，通过讲**基础表达式与元字符和运算符结合**，可以构建更大的表达式。这种构建表达式的方法就是正则表达式的语法。与编程语言不同的是，正则表达式并没有严谨个格式要求，只要符合表达式的构建组成规则即可。

2-1 普通字符

单一字符表达式

假定有字符串 abcdABCD

- 单一字符表达式：a 结果是 abcdABCD 单一表达式即表达匹配对应的字符（区分大小写），注意特殊字符需用转义符号
- 单一字符和单一字符：ab->abcdABCD，其中单一字符a和b构建了新的表达式，即ab，匹配时需要按照ab顺序规则进行匹配。例如字符串abdksab，ab-> abdksab，均被匹配

2-2 基础符号

[]集符号

假定有字符串 aekdkABks32

- []：字符集，表示圈定内的字符逐个按照单一字符方式匹配。
  - [a]：aekdkABks32
  - [abB]： aekdkABks32
  - [aedA]： aekdkABks32

-符号

假定有字符串 aekdkABks32

- -：配合[]符号，一起使用，表示区间拾取
  - [a-z]：aekdkABks32
  - [0-9]： aekdkABks32
  - [A-Z]： aekdkABks32
  - [a-zA-Z]：匹配所有的小写字母与大写字母，也可以记作[A-Za-z]
  - [a-z0-9A-Z]：匹配所有的字母与数组，字母不区分大小写

^符号

假定有字符串 aekd32

- ^：有两重意义，配合[]符号，表示取反，即除去字符规则外的其他字符。第二种强制检索字符串首，后面有介绍。
  - [^a]： aekd32
  - [^0-9]： aekd32
  - [^A-Z]： aekd32
  - [^a-z0-9]：匹配除去小写字母与数字之外的所有字符
  - [^ae]：匹配除去a和e以外的字符，结果是kd32，**作用于整个集中。**

.符号

假定有字符串 aekd32.

- 点：用于匹配任意字符，除去换行符外，如果需要匹配字符点时，需要通过使用转义符\，点是最简单的元字符
  - .： aekd32.
  - [.]：aekd32.
  - [0-9].：aekd32. 注意因为检索是从前往后匹配两个字符，32被第一次匹配后不参与后面的匹配。
  - [a-z].：aekd32. ae被当作两个字符匹配到，kd被当作两个字符匹配到
  - \.： aekd32. 等价[.]

正则表达式中的转义符

转义符，主要是表示在字符串中给定的一个标记符号的其他意义，使用比较多的是在换行符，回车符。例如字符n在字符串中代表符号n，但是如果在字符串中，想要表示换行，就需要借助转义符构建\n。这就是转义的目的。赋予另外一种意义。

一般计算机语言只认识常规的转义符，\r、\n、\t等。所以正则中的转义需要注意，输入方式，即输入符号\，做为正则表达式的转义符。那么\符号在字符串中的表示方式为\\双斜杠。所以这点需要着重注意。如下图。

```
FRegexPattern Pattern1(SourceString:TEXT("\w"));//语法错误，因为转义符中不存在\w
FRegexPattern Pattern2(SourceString:TEXT("\\w"));//语法正确，因为通过双斜杠完成了斜杠的转义变成了字符\，
```

2-3 非打印符

非打印字符

字符	描述
\cx	匹配由x指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。
\f	匹配一个换页符。等价于 \x0c 和 \cL。
\n	匹配一个换行符。等价于 \x0a 和 \cJ。
\r	匹配一个回车符。等价于 \x0d 和 \cM。
\t	匹配一个制表符。等价于 \x09 和 \cI。
\v	匹配一个垂直制表符。等价于 \x0b 和 \cK。

2-3 元字符

\w和\W

假定有字符串 aekd32._

- \w：匹配字母，数字和下划线 与[a-zA-Z0-9_]等价
  - \w： aekd32._
- \W：匹配非字符，数字和下划线，等价于[^a-zA-Z0-9_]
  - \W： aekd32._

\s\S

假定有字符串 aek d32._

- \s：匹配空白符，换行符
  - \s： aek d32._
- \S：匹配非空白符，但保留换行符
  - \S： aek d32._

元字符

元字符是正则表达式的重要组成部分，他以模糊描述的方式或是特定规则方式，加入文本串匹配中。元字符提供了更多的匹配规则，有些规则是增加限定，有些是为了组合新的规则，如果希望学好正则表达式，必须需要学好元字符。元字符和普通字符是构成正则表达式的基础。

转义符\

在正则表达式的元字符中，第一个要学习的就是转义符。所谓转义即将原有意义进行替换。例如在正则表达式中，一些字符被赋予了特殊的函数，**例如点符号。那如果我们希望匹配字符点**，而非正则表达式的元字符点，那怎么办呢？这时候就需要使用转义符了。**通过转义符可以将正则中的操作符号，转换为文本串符号。**

语法：\. 表意匹配点字符

限定符号^和$

已知字符串 abcd

- ^：匹配开始位置，即此符号为强制匹配符号后的第一个表达式内容为文本串头。
  - ^a：结果为 abcd，如果匹配规则为^b，则字符串不符合匹配。
- $：匹配结尾位置。即此符号为强制匹配符号前一个表达式为字符串尾巴。
  - d$：结果为abcd，如果匹配规则为c$，则字符串不符合匹配

表达式个数匹配符号{}

已知字符串 aaabbbbbccccc

{}符号，用于匹配符号前表达式个数，具体格式如下

{n}：n代表非负整数，匹配前面的规则n次。a{2}结果aaabbbbbccccc

{n,}：代表至少匹配前面的规则n次，也就是匹配n次到无限次。c{1,}结果是aaabbbbbccccc

{n,m}：m应大于等于n，表示至少匹配前面的规则n次，最多匹配m次。a{1,2}结果是aaabbbbbccccc

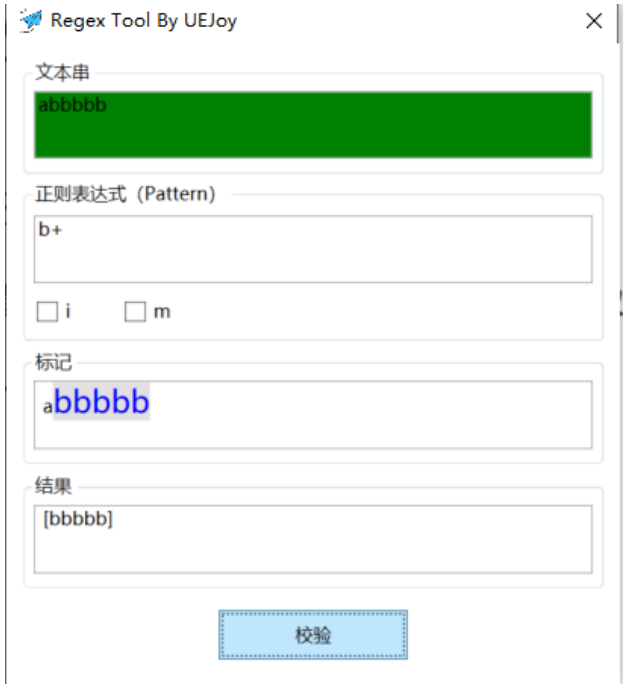
次数匹配符号（*、+、？）

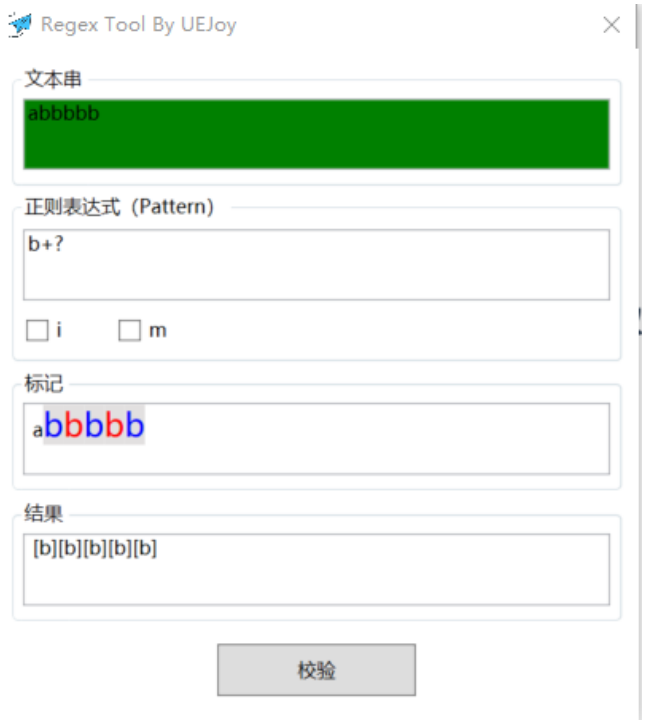
已知字符串 abcc

- *：匹配符号前的表达式，出现零次或多次。表明符号有两种匹配规则。*符号等价于“{0,}”。注意有个逗号。
  - c*：结果为 空，空，cc，空。注意：由于a符合出现零次，所以被匹配，但是a不是c所以匹配结果为空。
- +：匹配符号前的表达式，出现一次或多次（至少出现一次）。等价于{1,}。
  - c+：结果为cc
- ？：匹配符号前的表达式，零次或一次。等价于{0,1}。
  - c?：结果为，空，空，c，c，空

？另一意义

问号符号还有另外一层意义，即停止贪婪匹配。但必须将问号放置在其他规则符号后（？、+、*，{n}、{n,}、{n,m}），表示匹配最少结果。例如：字符串abbbbbb，如果我们用规则b+匹配，会匹配所有b。但是用b+?，只会匹配单个b，但会有多个结果。





“或”规则匹配符号（|）

符号“|”用于将多个表达式合并为一个表达式，匹配时只要其中一个表达式成立则成立。例如(a|b)cd，对于字符串acd和bcd均可以匹配

\b和\B元字符

\b和\B用来做单词边界（边界的定义为空格或是文本串尾）匹配。所谓单词，即出现在符号前的所有字符被设定为一个单词。

\b：匹配单词必须处于边界，例如表达式“ok\b”如用用来匹配串“abok ok”结果为abok ok

\B：匹配单词非处于边界。例如表达式“ok\B”,如果用来匹配串“adkoks ok”结果为adkoks ok

\d和\D元字符

\d用来匹配数字字符，等价于[0-9]。\D用于匹配非数字字符，等价于[^0-9]

2-4 符号组合优先级

组合优先级

当多个符号出现在同一个表达式中，我们需要考虑符号应用的优先级。例如 \.*，是优先组合规则.*还是\.呢?

运算符	描述
\	转义符
(), (?:), (?=), []	圆括号和方括号
*, +, ?, {n}, {n,}, {n,m}	限定符
^, $, \任何元字符、任何字符	定位点和序列（即：位置和顺序）
\| | 或 |

2-4 修饰符

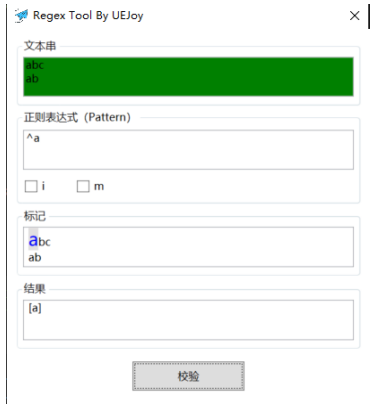
修饰符

修饰符主要用来在正则表达式之外额外增加全局匹配规则。

语法格式如下：/pattern/符号

修饰符	含义	描述
i	ignore - 不区分大小写	将匹配设置为不区分大小写，搜索时不区分大小写: A 和 a 没有区别。
m	multi line - 多行匹配	使边界字符 ^ 和 $ 匹配每一行的开头和结尾，记住是多行，而不是整个字符串的开头和结尾。

关于m：由于正则在匹配时，只考虑单行匹配，遇到换行符停止匹配。如果标记m则跨行完成匹配。



修饰符m

关于m：由于正则在匹配时，如果遇到表达式中添加了首尾限定，只考虑单行匹配，遇到换行符停止匹配。如果标记m则跨行完成匹配。

第三部分

虚幻中的正则表达式

火星时代游戏设计学院

示例代码

虚幻引擎中的正则表达式属于mini版本，大部分表达式功能不可用。例如修饰符无法被使用。

```
FRegexPattern Pattern(SourceString:TEXT("[a]"));
FRegexMatcher Matcher(Pattern, Input:TEXT("bdkacadkda"));

//寻找下一个匹配项，如果有匹配成功，则返回真。
//匹配完成后第一次调用检查匹配是否成立
Matcher.FindNext();
//获取匹配成功的文本串的字符首位置
Matcher.GetCaptureGroup(Index:0);
//获取
Matcher.GetCaptureGroupBeginning(Index:0);
//第二次调用时，则向下移动匹配结果，例如匹配结果有多个时，
//第一次调用匹配第一个，第二次调用匹配第二个，以此类推
Matcher.FindNext();
```

谢谢观看