#### 1. Regex là gì ?

Regex (Regular Expression) có nghĩa là “Biểu thức chính quy” dùng để mô tả một mẫu chuỗi (string pattern) được sắp xếp theo một quy luật nhất định nào đó, ví dụ số điện thoại, ngày tháng, địa chỉ email…

Khi bạn tìm 1 đoạn text, bạn có thể sử dụng một mẫu chuỗi để mô tả cái cần tìm.

Một mẫu đơn giản có thể là một ký tự đơn.

Mẫu phức tạp hơn có thể bao gồm nhiều ký tự, và có thể sử dụng để phân tích ngữ pháp, kiểm tra định dạng, thay thế và nhiều việc khác nữa.

#### 2.Tác dụng của Regex

Trong các kỹ thuật xử lý chuỗi (Ví dụ : Tìm kiếm, thay thế … ), thông thường chúng ta phải xác định các chuỗi tường minh để tìm kiếm. Các chuỗi này được gọi là từ khóa.

Tuy nhiên không phải lúc nào chúng ta cũng có thể chỉ rõ chuỗi từ khóa đó mà chỉ có thể mô tả được quy luật để sinh ra chuỗi từ khóa đó, chẳng hạn làm thế nào để lấy được toàn bộ địa chỉ email trong một văn bản?

Regex giúp ta dễ dàng giải quyết vấn đề này. Đơn giản bạn chỉ cần thiết lập ra quy luật của một địa chỉ email và yêu cầu chương trình tìm kiếm theo luật đó.

Tóm lại, Regex có các tác dụng sau :

* Regex được sử dụng trong việc xử lý chuỗi : tìm kiếm, thay thế, bóc tách dữ liệu …
* Regex được sử dụng nhiều trong việc kiểm tra tính hợp lệ đầu vào của dữ liệu, ví dụ : địa chỉ email có hợp lệ hay không? Định dạng ngày tháng có đúng không?

## 3. Cú pháp cho Regular Expression trong Java

|  |  |
| --- | --- |
| **Subexpression** | **So khớp** |
| ^ | So khớp với phần bắt đầu của dòng (line) |
| $ | So khớp với phần cuối của dòng |
| . | So khớp với bất kỳ ký tự đơn nào ngoại trừ newline (dòng mới). Sử dụng tùy chọn m cũng cho phép nó so khớp với newline (dòng mới) |
| [...] | So khớp với bất kỳ ký tự đơn nào trong dấu ngoặc vuông |
| [^...] | So khớp với bất kỳ ký tự đơn nào không trong dấu ngoặc vuông |
| \A | Phần bắt đầu của cả chuỗi |
| \z | Phần cuối của cả chuỗi |
| \Z | Phần cuối của cả chuỗi |
| re\* | So khớp với 0 hoặc nhiều sự xuất hiện của expression đặt trước |
| re+ | So khớp với 1 hoặc nhiều của cái gì đó ở trước |
| re? | So khớp với 0 hoặc 1 sự xuất hiện của expression đặt trước |
| re{ n} | So khớp một cách chính xác với n lần xuất hiện của Expression đặt trước |
| re{ n,} | So khớp với n lần xuất hiện hoặc nhiều hơn của Expression đặt trước |
| re{ n, m} | So khớp với ít nhất n và nhiều nhất m lần xuất hiện của Expression đặt trước |
| a| b | So khớp với hoặc a hoặc b |
| (re) | Nhóm các Regular Expression và ghi nhớ text đã so khớp |
| (?: re) | Nhóm các Regular Expression mà không ghi nhớ text đã so khớp |
| (?> re) | So khớp với patter độc lập mà không truy tích ngược (backtrack) |
| \w | So khớp với các ký tự từ |
| \W | So khớp với các ký tự không phải từ |
| \s | So khớp với khoảng trống trắng. Tương đương với [\t\n\r\f]. |
| \S | So khớp với các ký tự không là khoảng trống trắng |
| \d | So khớp với các chữ số. Tương đương với [0-9]. |
| \D | So khớp với ký tự không là chữ số |
| \A | So khớp với phần bắt đầu của chuỗi |
| \Z | So khớp với phần kết thúc của chuỗi. Nếu một newline (dòng mới) tồn tại, nó so khớp với ngay trước newline (dòng mới). |
| \z | So khớp với phần kết thúc của chuỗi |
| \G | So khớp với điểm, nơi mà so khớp cuối cùng kết thúc |
| \n | Tham chiếu ngược để capture group số "n" |
| \b | So khớp với các giới hạn từ bên ngoài dấu ngoặc vuông. So khớp với phím lùi (0x08) khi ở trong dấu ngoặc vuông |
| \B | So khớp các giới hạn không phải từ |
| \n, \t, etc. | So khớp với các newline (dòng mới), carriage return, tab, … |
| \Q | Thoát (trích dẫn) tất cả ký tự tới \E |
| \E | Kết thúc trích dẫn bắt đầu từ \Q |

VD: ta muốn kiểm tra 1 email có hợp lệ:

|  |  |
| --- | --- |
| Mã nguồn: | |
| 1  2  3  4  5  6  7 | string email = "vector.mic@gmail.com";  regex pattern("[a-zA-Z0-9\_\.]+@[a-zA-Z]+\.[a-zA-Z]+(\.[a-zA-Z]+)\*");  // Hàm regex\_match() dùng để kiểm tra TOÀN BỘ chuỗi  // Hàm regex\_search() dùng để kiểm tra CHUỖI CON trong chuỗi  if (regex\_match(email, pattern)    cout << "Match!" << endl;  else cout << "Not match!" << endl; |

Cùng phân tích:

**[a-zA-Z0-9\_\.]+@[a-zA-Z]+\.[a-zA-Z]+(\.[a-zA-Z]+)\***

1. Đầu tiên ta kiểm tra phần tên email, tên email bao gồm kí tự thường, in hoa, chữ số, dấu **\_** và **.**. Vì vậy nên ta phải gom tất cả vào 1 nhóm **[a-zA-Z\_\.]** (bởi vì giữa kí tự thường và in hoa có các kí tự đặc biệt nên ta không thể dùng **A-z** mà phải tách ra **a-zA-Z**). Tuy nhiên ta cần phải thêm dấu **+** vì nhóm đó chỉ đại diện cho **1 kí tự duy nhất**.
2. Kí tự @
3. Phần tên miền, ở đây mình quy định tên miền chỉ bao gồm các kí tự thường và hoa, có dấu **.** phân cách. Vì vậy nên ta ghi vào **[a-zA-Z]+\.[a-zA-Z]+**. Tuy nhiên nó chỉ đúng với vài tên miền dạng như gmail.com, yahoo.com, live.com,… nhưng sẽ không đúng với những tên miền dạng như yahoo.com.vn, dauan.com.xyz.vn,… chẳng hạn. Vì vậy ta cần phải thêm vào 1 group bao gồm dấu **.**và các kí tự chữ **(\.[a-zA-Z]+)**. Group này có thể có cũng được, không có cũng được nên ta đặt dấu **\***ngay đuôi (khớp với 0 lần trở lên).

Vậy là ta đã thiết kế xong 1 pattern kiểm tra email hợp lệ rồi! Đơn giản phải không nào?

VD: ta muốn kiểm tra 1 số điện thoại di động hợp lệ ở Việt Nam (không có khoảng trắng nhé)

|  |  |
| --- | --- |
| Mã nguồn: | |
| 1  2  3  4  5 | string input = "0987412354";  regex pattern("(\\+84|0)\\d{9,10}");  if (regex\_match(email, pattern)    cout << "Match!" << endl;  else cout << "Not match!" << endl; |

Cùng phân tích:

**(\\+84|0)\\d{9,10}**

1. Đầu tiên, do số điện thoại có phần mở đầu có thể là +84 (ở Việt Nam) hoặc là 0, nên ta cần đặt vào trong group và thêm dấu **|**
2. Kế tiếp, 1 số điện thoại bao gồm 10 hoặc 11 chữ số, nhưng ta không tính phần đầu của số điện thoại nên chỉ còn khoảng 9 – 10 chữ số