# Regex em Java

# Da Validação de Dados à Extração Avançada em JSON e XML

Expressões regulares (regex) são uma ferramenta poderosa para busca, extração e manipulação de padrões em strings.

Em Java, elas são amplamente utilizadas para validar entradas, formatar dados e realizar parsing de estruturas textuais complexas.

Este artigo abordará a criação e uso de expressões regulares em diferentes versões do Java (8, 11 e 17), além de trazer exemplos práticos, desde padrões simples até regex mais complexos, voltados à extração de dados em formatos como JSON e XML.

O objetivo é proporcionar uma visão técnica aprofundada e otimizar o uso de regex em situações de alta performance.

## Estrutura Básica de Regex

Uma regex consiste em:

- **Literais**: Correspondem a caracteres exatos na string.
- **Metacaracteres**: Têm significados especiais e definem padrões complexos (., *, +, ?, ^, $, etc.).

### Exemplo simples:

A expressão `\d+` captura uma ou mais ocorrências de dígitos numéricos:

- Texto: "O código é 1234 e o valor é 5678"
- Padrão: `\d+`
- Resultado: "1234", "5678"

# Regex em Java: Uso e Implementação

Java oferece o pacote java.util.regex com as classes Pattern e Matcher para facilitar a manipulação de regex.

A estrutura geral para trabalhar com regex envolve três etapas:

1. Compilar a expressão regular usando Pattern.compile().
2. Criar um Matcher para aplicar o padrão em uma string.
3. Usar métodos como find(), matches(), e group() para interagir com os resultados.

### Exemplo Simples em Java 8:

```
import java.util.regex.*;

public class RegexExample {
    public static void main(String[] args) {
        String text = "O código é 1234 e o valor é 5678";
        String pattern = "\\d+";

        Pattern compiledPattern = Pattern.compile(pattern);
        Matcher matcher = compiledPattern.matcher(text);

        while (matcher.find()) {
            System.out.println("Número encontrado: " + matcher.group());
        }
    }
}
```

**Saída**:

yaml
Número encontrado: 1234
Número encontrado: 5678

# Regex em Java 11: Melhorias no Manipulador de Strings

Java 11 introduziu melhorias na API de strings, como o método String::lines, facilitando o processamento de texto multilinear combinado com regex.

Exemplo com Multilinhas:

```
public class RegexJava11 {
    public static void main(String[] args) {
        String multiLineText = "Linha 1\nLinha 2\nLinha 3";

        Pattern pattern = Pattern.compile("^Linha \\d$", Pattern.MULTILINE);
        Matcher matcher = pattern.matcher(multiLineText);

        while (matcher.find()) {
            System.out.println("Linha encontrada: " + matcher.group());
        }
    }
}
```

**Saída**:

yaml

```
Linha encontrada: Linha 1
Linha encontrada: Linha 2
Linha encontrada: Linha 3
```

# Regex em Java 17: Integração com Padrões Modernos

Java 17 trouxe novas funcionalidades que permitem maior flexibilidade ao combinar regex com estruturas de controle de fluxo.

Enquanto a API de regex se manteve estável, a interação com features mais modernas, como o pattern matching, traz um código mais limpo e eficiente.

## Validação de Email com Regex em Java 17:

```
public class RegexJava17 {
    public static void main(String[] args) {
        String[] emails = {"email@example.com", "user@invalid", "test@domain.co"};

        String emailPattern = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";
        Pattern pattern = Pattern.compile(emailPattern);

        for (String email : emails) {
            boolean isValid = pattern.matcher(email).matches();
            System.out.println("Email " + email + " é válido? " + isValid);
        }
    }
}
```

**Saída**:

sql
Email email@example.com é válido? true
Email user@invalid é válido? false
Email test@domain.co é válido? True

# Extração de Dados em JSON com Regex

Regex pode ser usada para tarefas simples de extração de informações em JSON. Abaixo, um exemplo que captura pares chave-valor de um objeto JSON.

## Exemplo de JSON:

json
```
{
   "nome": "João",
   "idade": 30,
   "cidade": "São Paulo"
}
```

## Extraindo Pares Chave-Valor com Regex:

```
public class JsonRegexExample {
   public static void main(String[] args) {
      String json = "{\"nome\":\"João\",\"idade\":30,\"cidade\":\"São Paulo\"}";

      String jsonPattern = "\"(\\w+)\":\\s*\"?([\\w\\s]+)\"?";
      Pattern pattern = Pattern.compile(jsonPattern);
      Matcher matcher = pattern.matcher(json);

      while (matcher.find()) {
         System.out.println("Chave: " + matcher.group(1) + ", Valor: " + matcher.group(2));
      }
   }
}
```

**Saída**:

yaml
```
Chave: nome, Valor: João
Chave: idade, Valor: 30
Chave: cidade, Valor: São Paulo
```

# Extração de Dados em XML com Regex

Para XML, regex é usada principalmente para extrair conteúdo de elementos simples.

No entanto, regex deve ser usada com cautela, uma vez que XML tem uma estrutura hierárquica que regex não pode tratar diretamente.

## Exemplo de XML:

xml
```
<cliente>
   <nome>Maria</nome>
   <idade>25</idade>
   <cidade>Rio de Janeiro</cidade>
</cliente>
```

## Extraindo Elementos XML com Regex:

```
public class XmlRegexExample {
   public static void main(String[] args) {
      String xml = "<cliente><nome>Maria</nome><idade>25</idade><cidade>Rio de
Janeiro</cidade></cliente>";

      String xmlPattern = "<(\\w+)>([^<]+)</\\1>";
      Pattern pattern = Pattern.compile(xmlPattern);
      Matcher matcher = pattern.matcher(xml);

      while (matcher.find()) {
         System.out.println("Tag: " + matcher.group(1) + ", Conteúdo: " + matcher.group(2));
      }
   }
}
```

**Saída**:

vbnet
```
Tag: nome, Conteúdo: Maria
Tag: idade, Conteúdo: 25
Tag: cidade, Conteúdo: Rio de Janeiro
```

## *Otimização* e Melhores Práticas para Regex

1. **Defina padrões claros e concisos**: Quantificadores como {2,5} são mais seguros que * ou +, que podem levar a correspondências excessivamente permissivas.
2. **Evite backtracking excessivo**: Em regex complexas, padrões como .* podem causar sobrecarga no tempo de execução devido ao backtracking. Otimize usando quantificadores mais restritos.
3. **Use flags apropriadas**: Flags como Pattern.MULTILINE ou Pattern.CASE_INSENSITIVE podem ser combinadas para melhorar a legibilidade e a flexibilidade da expressão.
4. **Testes com Ferramentas Adequadas**: Ferramentas como o Regex101 permitem testar e otimizar expressões regulares de maneira interativa.

## Conclusão

Regex é uma ferramenta essencial para desenvolvedores que lidam com manipulação de texto e validação de dados.

Este artigo demonstrou como utilizar expressões regulares desde a validação de entradas até a extração de dados de estruturas como JSON e XML.

O uso eficiente e performático de regex, combinado com as melhorias de versões mais recentes do Java, garante soluções robustas e eficientes para lidar com manipulação de padrões complexos em sistemas de alta demanda.

***EducaCiência FastCode para a comunidade***