### Calendar类

Calendar 类是 Java 中用于操作日期和时间的类，它提供了丰富的方法来处理日期和时间的计算、比较和格式化等操作。下面是关于 Calendar 类的基础使用方法：

创建 Calendar 对象：Calendar 类是一个抽象类，不能直接实例化，需要通过其子类来创建对象。常用的方法是通过 getInstance() 静态方法获取 Calendar 对象：

Calendar calendar = Calendar.getInstance();

Calendar calendar=new Calendar(); //错误

获取日期和时间信息：Calendar 类提供了一系列方法用于获取日期和时间的各个部分，例如年、月、日、时、分、秒等：

int year = calendar.get(Calendar.YEAR);

int month = calendar.get(Calendar.MONTH) + 1; // 月份从0开始，需要加1

int dayOfMonth = calendar.get(Calendar.DAY\_OF\_MONTH);

int hour = calendar.get(Calendar.HOUR\_OF\_DAY);

int minute = calendar.get(Calendar.MINUTE);

int second = calendar.get(Calendar.SECOND);

设置日期和时间信息：Calendar 类也提供了一系列方法用于设置日期和时间的各个部分：

calendar.set(Calendar.YEAR, 2024);

calendar.set(Calendar.MONTH, Calendar.APRIL); // 月份从0开始，可以使用常量来表示

calendar.set(Calendar.DAY\_OF\_MONTH, 22);

calendar.set(Calendar.HOUR\_OF\_DAY, 10);

calendar.set(Calendar.MINUTE, 30);

calendar.set(Calendar.SECOND, 0);

日期和时间的计算：Calendar 类支持对日期和时间进行加减运算，例如：

calendar.add(Calendar.DAY\_OF\_MONTH, 1); // 加一天

calendar.add(Calendar.MONTH, -1); // 减一个月

日期和时间的格式化：Calendar 类本身并不负责日期和时间的格式化，通常需要配合 SimpleDateFormat 类来进行格式化：

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String formattedDate = sdf.format(calendar.getTime());

System.out.println("Formatted Date: " + formattedDate);



### 正则表达式

正则表达式主要用于文本处理，对字符串进行查找、解析和验证，比如单词的查找替换，电子邮件、身份证号格式的校验等。这些操作都涉及字符串的模式匹配问题，模式匹配是一种复杂的字符串运算算法。正则表达式赋予一些字符特殊的含义，用它们描述字符串模式，功能化了模式匹配的过程。

#### 1、预定义字符类

|  |  |  |  |
| --- | --- | --- | --- |
| 预定义字符 | 说明 | 预定义字符 | 说明 |
| \d | 匹配0~9任意一个数字 | \D | 匹配非0~9数字 |
| \s | 匹配\t、\n、\r等空白符 | \S | 匹配非空白符 |
| \w | 匹配数0~9，字母a~z，A~Z，下划线 | \W | 匹配非数字和字母 |
| **.** | 匹配任意一个字符 |  |  |

#### 2、方括号

|  |  |
| --- | --- |
| 示例 | 说明 |
| [aeiou] | 表示枚举。匹配a,e,i,o,u中的任意一个字符 |
| [a-fx-z] | -表示范围。匹配a~f和x~z范围内的任意一个字符 |
| [^abcd] | ^表示求反。匹配a,b,c,d之外的任意一个字符 |
| [a-e&&[dfg]] | &&表示交集运算。匹配a~e与d，f，g的交集，即d |

#### 3、量词

|  |  |
| --- | --- |
| 示例 | 说明 |
| X? | 1≥X表达式出现的次数≥0，即0次或1次 |
| X\* | X表达式出现的次数≥0，即0次或多次 |
| X+ | X表达式出现的次数≥1，即1次或多次 |
| X{n} | X表达式出现的次数=n次，即n次 |
| X{n,} | X表达式出现的次数≥n次，即最少出现n次 |
| X{n,m} | m≥X表达式出现的次数≥n，即最少出现n次，最多出现m次 |

#### Pattern 类：

java.util.regex.Pattern 类用于表示编译后的正则表达式模式。通过 Pattern.compile() 方法可以将正则表达式字符串编译成一个 Pattern 对象。

import java.util.regex.Pattern;

public class Main {

public static void main(String[] args) {

String regex = "\\d+"; // 匹配一个或多个数字

Pattern pattern = Pattern.compile(regex);

}

}

#### Matcher 类：

java.util.regex.Matcher 类用于在输入字符串中执行匹配操作。通过调用 Pattern.matcher() 方法可以创建一个 Matcher 对象，然后可以使用 find()、matches()、group() 等方法进行匹配操作。

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Main {

public static void main(String[] args) {

String input = "12345";

String regex = "\\d+"; // 匹配一个或多个数字

Pattern pattern = Pattern.compile(regex);

Matcher matcher = pattern.matcher(input);

if (matcher.matches()) {

System.out.println("输入只包含数字");

}

}

}

#### 替换操作【replaceAll方法】：

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Main {

public static void main(String[] args) {

String input = "Hello, world!";

String regex = "\\w+";

Pattern pattern = Pattern.compile(regex);

Matcher matcher = pattern.matcher(input);

String replaced = matcher.replaceAll("Java");

System.out.println("Replaced string: " + replaced);

}

}

#### find方法

find() 方法是 Matcher 类中的一个方法，用于在输入字符串中查找与正则表达式匹配的子序列。它尝试在输入字符串中寻找与正则表达式匹配的下一个子序列，如果找到则返回 true，否则返回 false。

find() 方法通常与 while 循环结合使用，以便多次查找匹配的子序列。

下面是一个简单的示例，演示了如何使用 find() 方法查找输入字符串中的所有数字子序列：

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Main {

public static void main(String[] args) {

String input = "Java 123 is fun and 456 exciting!";

String regex = "\\d+"; // 匹配一个或多个数字

Pattern pattern = Pattern.compile(regex);

Matcher matcher = pattern.matcher(input);

while (matcher.find()) {

String matched = matcher.group(); // 获取匹配的子序列

System.out.println("Matched substring: " + matched);

}

}

}

#### group方法

group() 方法是 Matcher 类中的一个方法，用于获取与上一次匹配操作匹配的子序列。

当调用 Matcher 对象的 find() 方法进行匹配操作后，如果匹配成功，可以使用 group() 方法获取匹配的子序列。如果没有匹配成功，调用 group() 方法会抛出 IllegalStateException 异常。

group() 方法有两种形式（有参、无参）：

无参：返回与整个表达式匹配的子字符串。

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Main {

public static void main(String[] args) {

String input = "Hello, world!";

String regex = "\\w+";

Pattern pattern = Pattern.compile(regex);

Matcher matcher = pattern.matcher(input);

if (matcher.find()) {

String matched = matcher.group();

System.out.println("Matched substring: " + matched);

}

}

}

带参数形式：返回在前一次匹配操作期间由给定组捕获的输入子序列。

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Main {

public static void main(String[] args) {

String input = "Hello, world!";

String regex = "(\\w+), (\\w+)!";

Pattern pattern = Pattern.compile(regex);

Matcher matcher = pattern.matcher(input);

if (matcher.find()) {

String first = matcher.group(1); // 获取第一个组捕获的子序列

String second = matcher.group(2); // 获取第二个组捕获的子序列

System.out.println("First group: " + first);

System.out.println("Second group: " + second);

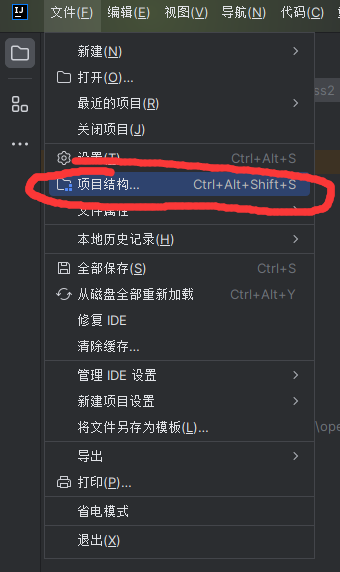
}

}

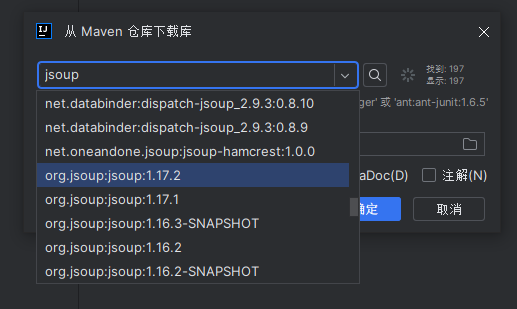
}

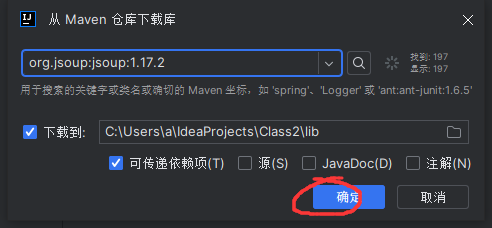
## Java jsoup爬虫

1. 安装依赖项



1. 
2. org.jsoup:jsoup:1.17.2 找到这个





使用

class spider{

String url;

String userAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36"; // 设置 User-Agent

public spider(String url) {

this.url = url;

}

public Document down\_html(){

// 使用 Jsoup 发送 HTTP 请求并获取网页内容

try {

Document doc = Jsoup.connect(url)

.userAgent(userAgent) // 设置 User-Agent

.get();

return doc;

} catch (IOException e) {

throw new RuntimeException(e);

}

}

}

public class TEST {

public static void main(String[] args) {

spider s = new spider("https://xl.16888.com/origin-1.html");

System.out.println( s.down\_html());

String html\_content = s.down\_html().toString();

String regex = "href=\"(.\*?)\"";

Pattern pattern = Pattern.compile(regex);

Matcher matcher1 = pattern.matcher(html\_content);

while(matcher1.find()){

String first = matcher1.group(1); // 获取第一个组捕获的子序列

System.out.println("First group: " + first);

}

}

}