Java → Regular expressions → Match results

Theory: Match results

25 minutes 0 / 5 problems solved

Skip this topic Start practicing

1328 users solved this topic. Latest completion was about 9 hours ago.

A simple check whether a string contains a substring matching our regular expression is not the only thing we can do with a `Matcher` object. It also provides us with additional information about matches, which is essential in some tasks.

Current topic:

Match results

Topic depends on:

Patterns and Matcher

Stage 7 What is an exception

§1. Getting match results

As you know, the `find` method of `Matcher` can check whether a substring of a string matches the pattern. Here is an example.

```
String javaText = "Java supports regular expressions. LET'S USE JAVA!!!";

Pattern javaPattern = Pattern.compile("java", Pattern.CASE_INSENSITIVE);
Matcher matcher = javaPattern.matcher(javaText);

System.out.println(matcher.find()); // prints "true"
```

When `find()` method returns `true` it is possible to get some info about the substring matching the pattern. `start()` and `end()` return the starting and the last indices of the match respectively, while `group()` returns the matching substring itself.

```
System.out.println(matcher.start()); // 0, the starting index of match
System.out.println(matcher.end());   // 4, the index followed the last index of match
System.out.println(matcher.group()); // "Java", a substring that matches the pattern
```

There is a special class `MatchResult` that comprises all this information about the match:

```
MatchResult result = matcher.toMatchResult(); // a special object containing match results

System.out.println(result.start()); // 0
System.out.println(result.end());   // 4
System.out.println(result.group()); // "Java"
```

Be careful, if you invoke the methods `start`, `end`, `group` before invoking `find()` method or in case it was invoked and returned `false`, they will throw `IllegalStateException`. To avoid the exception, you should always check the boolean result of `find()` before invoking these methods.

```
if (matcher.find()) {
    System.out.println(matcher.start());
    System.out.println(matcher.end());
    System.out.println(matcher.group());
} else {
    System.out.println("No matches found");
}
```

This code prints "No matches found" if the `find` method returns `false`. It also makes sure that `start()`, `end()`, `group()` are invoked only after the `find()` method.

§2. Iterating over multiple matches

Sometimes more than one substring matches the same pattern. In the previous example, there are two suitable strings "Java" and "JAVA", because the pattern is case insensitive. The `find()` method allows us to iterate in a loop over all substrings that match the pattern.

```
String javaText = "Java supports regular expressions. LET'S USE JAVA!!!";

Pattern javaPattern = Pattern.compile("java", Pattern.CASE_INSENSITIVE);
Matcher matcher = javaPattern.matcher(javaText);

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

This code outputs the following lines:

```
group: Java, start: 0
group: JAVA, start: 45
```

The condition of the loop allows invoking `start()` and `group()` only when the `find()` method returns `true`.

§3. Conclusions

As you can see, `Matcher` saves all the necessary info about matches. We can learn where every matching substring starts and where it ends and how it looks like. Later this data can be saved in a `MatchResult` object.

Report a typo

147 users liked this theory. 3 didn't like it. **What about you?**

Start practicing

Comments (18)   Hints (0)   Useful links (0)   Show discussion

Table of contents: