[**Spring之AntPathMatcher**](https://www.cnblogs.com/zhangxiaoguang/p/5855113.html)

**前言**

AntPathMatcher是什么？主要用来解决什么问题？

背景：在做uri匹配规则发现这个类，根据源码对该类进行分析，它主要用来做类URLs字符串匹配；

**效果**

可以做URLs匹配，规则如下

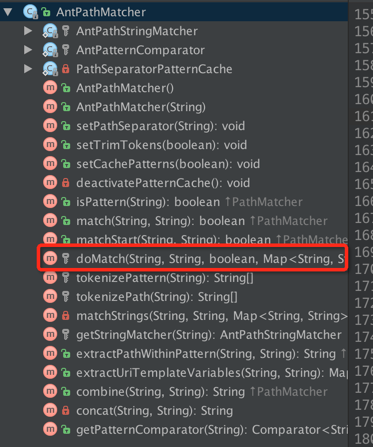
1. ？匹配一个字符
2. \*匹配0个或多个字符
3. \*\*匹配0个或多个目录

用例如下

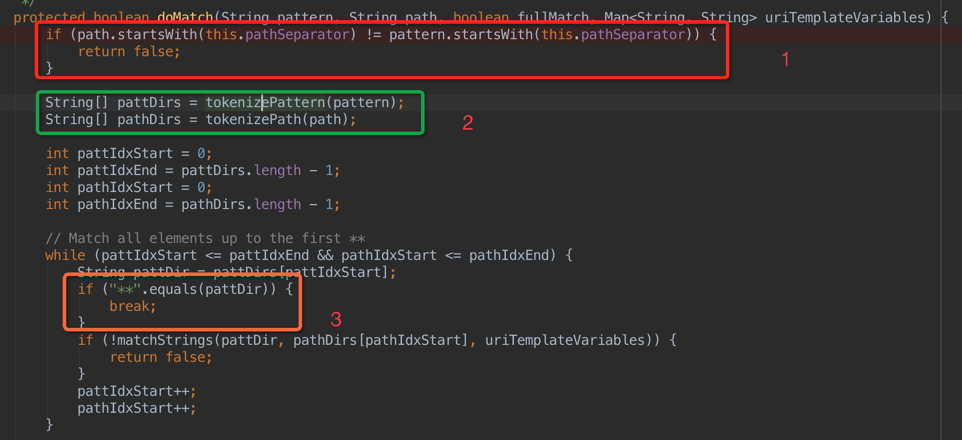
* /trip/api/\*x    匹配 /trip/api/x，/trip/api/ax，/trip/api/abx ；但不匹配 /trip/abc/x；
* /trip/a/a?x    匹配 /trip/a/abx；但不匹配 /trip/a/ax，/trip/a/abcx
* /\*\*/api/alie    匹配 /trip/api/alie，/trip/dax/api/alie；但不匹配 /trip/a/api
* /\*\*/\*.htmlm   匹配所有以.htmlm结尾的路径

**核心**

AntPathMatcher API接口



由上图可知，AntPathMatcher提供了丰富的API，主要以doMatch为主，下边来讲doMatch的实现上（其中pattern为制定的url模式，path为具体的url，下边以英文为主讲解）：



1    首先判断pattern和path的首字符是否同时为设置的分隔符，结果不一致则直接返回false，不进行下边的操作；

2    分别对pattern和path进行分词，形成各自的字符串数组，其中分词的主要代码如下（这段代码很清晰）：

[复制代码](javascript:void(0);)

public static String[] tokenizeToStringArray(String str, String delimiters, boolean trimTokens, boolean ignoreEmptyTokens) {

if (str == null) {

return null;

}

StringTokenizer st = new StringTokenizer(str, delimiters);

List<String> tokens = new ArrayList<String>();

while (st.hasMoreTokens()) {

String token = st.nextToken();

if (trimTokens) {

token = token.trim();

}

if (!ignoreEmptyTokens || token.length() > 0) {

tokens.add(token);

}

}

return toStringArray(tokens);

}

[复制代码](javascript:void(0);)

注：str代表要进行分词的字符串，delimiters是进行分词的分隔符，trimTokens表示是否对每一个分词进行首尾去空字符串，ignoreEmptyTokens代表分割之后是否保留空字符串；

我们发现，每次计算这个也是要花费一定的时间消耗，那每次真的是要重新计算么 ？看下边的代码来找答案（下边的代码是在上个方法tokenizeToStringArray调用之前进行）：

[复制代码](javascript:void(0);)

private final Map<String, String[]> tokenizedPatternCache = new ConcurrentHashMap<String, String[]>(256);

......

protected String[] tokenizePattern(String pattern) {

String[] tokenized = null;

Boolean cachePatterns = this.cachePatterns;

if (cachePatterns == null || cachePatterns.booleanValue()) {

tokenized = this.tokenizedPatternCache.get(pattern);

}

if (tokenized == null) {

tokenized = tokenizePath(pattern);

if (cachePatterns == null && this.tokenizedPatternCache.size() >= CACHE\_TURNOFF\_THRESHOLD) {

// Try to adapt to the runtime situation that we're encountering:

// There are obviously too many different patterns coming in here...

// So let's turn off the cache since the patterns are unlikely to be reoccurring.

deactivatePatternCache();

return tokenized;

}

if (cachePatterns == null || cachePatterns.booleanValue()) {

this.tokenizedPatternCache.put(pattern, tokenized);

}

}

return tokenized;

}

[复制代码](javascript:void(0);)

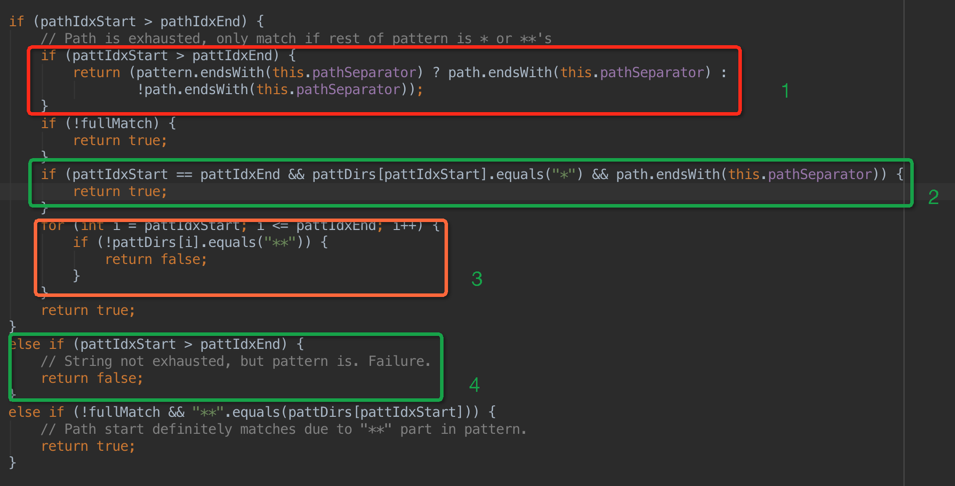
我们看到，这里存了一个pattern的cache **tokenizedPatternCache**，key为pattern，value为分次之后的字符串数组，每次先到cache获取，没有的话则计算，然后放入到cache里边，这样在做频繁的url mapping的时候，由于规则是有限的，可以很大程度减少计算；

同理，path也是通过同样的计算，不过，path则不会缓存，每次都需要调用tokenizeToStringArray进行分词（为什么呢？[**1**]）

接着来说3：

3    对分词之后的pattern数组和path数组从begin进行遍历，一旦pattern的第一个字符串是\*\*的话，则跳出来，此时没有直接返回true，为什么呢[**2**]？

下边接着看doMatch的中间部分代码（也就是说当break或者运行完毕while循环的时候，在退出之前会接着执行下边的代码）



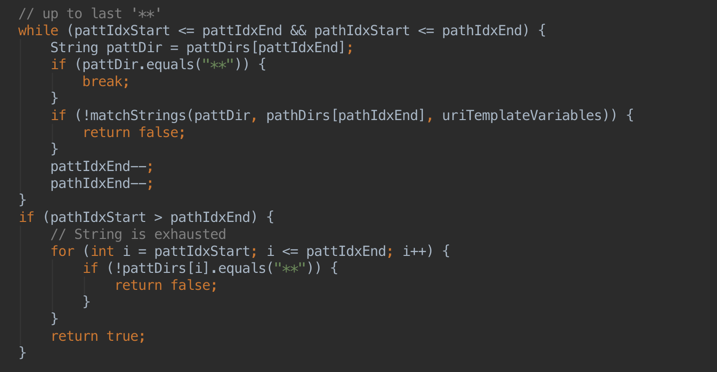
1    如果path分词数组正常执行完毕，则pathIdxStart是会比pathIdxEnd大1的，这个时候，如果pattern的字符串数组也正常耗尽，则来判断pattern和path的最后一个字符是否同步，按结果返回；

2    如果上边的循环只执行了一次，则这时候pattIdxStart则和pattIdxEnd相等，同时pattern的最后一个字符是\*且path最后是一个分隔符，则直接返回true；

3    如果pattern的最后一个字符串是\*\*则path不需要判断直接返回true；

4    这一步代表，pattern已经耗尽但是path还没耗尽，这时候肯定不匹配，直接返回false

接下来接着看，紧接着上边第二幅黑色背景图，如果第一次因为\*\*弹出来，看下边如何处理：



这个时候，开始从后往前遍历，如果再次弹出来不是因为遇到了\*\*，是正常遍历完成，这个时候，pathIdxStart是大于pathIdxEnd，这个时候字符串已经耗尽，如果pattern还没有耗尽，并且最后并不是\*\*，则直接返回false；

如果中间再次出现\*\*，并且path并没有耗尽，则进行下边的步骤：



这一部分代码主要用来循环处理中间再次\*\*的情况，直到完全处理完成，这里边用到了Java的标签语法：strLoop，符合条件则跳转到strLoop（类似goto）;

**总结**

这一部分的处理理解起来不是非常难懂，但是这个关于字符串匹配的过程是及其细致的，每一个边界问题都想得比较完美，这一点是相当值得学习的。

**后记**

其中，每一个path的分词是如何匹配到pattern的分词是怎么做的呢？答案就在  **matchStrings**这个方法里边了**：**

首先用path来匹配pattern的时候，要获取一个matcher，代码如下：

[复制代码](javascript:void(0);)

final Map<String, AntPathStringMatcher> stringMatcherCache = new ConcurrentHashMap<String, AntPathStringMatcher>(256);

......

protected AntPathStringMatcher getStringMatcher(String pattern) {

AntPathStringMatcher matcher = null;

Boolean cachePatterns = this.cachePatterns;

if (cachePatterns == null || cachePatterns.booleanValue()) {

matcher = this.stringMatcherCache.get(pattern);

}

if (matcher == null) {

matcher = new AntPathStringMatcher(pattern);

if (cachePatterns == null && this.stringMatcherCache.size() >= CACHE\_TURNOFF\_THRESHOLD) {

// Try to adapt to the runtime situation that we're encountering:

// There are obviously too many different patterns coming in here...

// So let's turn off the cache since the patterns are unlikely to be reoccurring.

deactivatePatternCache();

return matcher;

}

if (cachePatterns == null || cachePatterns.booleanValue()) {

this.stringMatcherCache.put(pattern, matcher);

}

}

return matcher;

}

[复制代码](javascript:void(0);)

这里new AntPathStringMatcher（AntPathMatcher的一个内部类）的时候也是需要一些计算，matcher构建的精华全部在这里了：

[复制代码](javascript:void(0);)

public AntPathStringMatcher(String pattern) {

StringBuilder patternBuilder = new StringBuilder();

Matcher m = GLOB\_PATTERN.matcher(pattern);

int end = 0;

while (m.find()) {

patternBuilder.append(quote(pattern, end, m.start()));

String match = m.group();

if ("?".equals(match)) {

patternBuilder.append('.');

}

else if ("\*".equals(match)) {

patternBuilder.append(".\*");

}

else if (match.startsWith("{") && match.endsWith("}")) {

int colonIdx = match.indexOf(':');

if (colonIdx == -1) {

patternBuilder.append(DEFAULT\_VARIABLE\_PATTERN);

this.variableNames.add(m.group(1));

}

else {

String variablePattern = match.substring(colonIdx + 1, match.length() - 1);

patternBuilder.append('(');

patternBuilder.append(variablePattern);

patternBuilder.append(')');

String variableName = match.substring(1, colonIdx);

this.variableNames.add(variableName);

}

}

end = m.end();

}

patternBuilder.append(quote(pattern, end, pattern.length()));

this.pattern = Pattern.compile(patternBuilder.toString());

}

[复制代码](javascript:void(0);)

这部分计算比较频繁，也会耗费一定量的时间，所以这里用到了一个叫做 **stringMatcherCache**的cache，上文中提到的两个cache的数量都不能超过65536，有其中任意一个cache超过这个限制，则会清空整个cache。

凌渡冰 目前就职于美团 Do what you like to impact the world.

分类: [java](https://www.cnblogs.com/zhangxiaoguang/category/787787.html), [spring](https://www.cnblogs.com/zhangxiaoguang/category/787786.html)