最近在看Spring 源码,源码中多次出现AntPatternMatcher 类, 此类作用用于实现对URL的匹配

关于此类的介绍,其中有些代码是从Apache Ant 那 "借"过来的

PathMatcher implementation for Ant-style path patterns. Examples are provided below.

Part of this mapping code has been kindly borrowed from [Apache Ant](http://ant.apache.org/).

The mapping matches URLs using the following rules:

* ? matches one character
* \* matches zero or more characters
* \*\* matches zero or more 'directories' in a path

Some examples:

* com/t?st.jsp - matches com/test.jsp but also com/tast.jsp or com/txst.jsp
* com/\*.jsp - matches all .jsp files in the com directory
* com/\*\*/test.jsp - matches all test.jsp files underneath the com path
* org/springframework/\*\*/\*.jsp - matches all .jsp files underneath the org/springframework path
* org/\*\*/servlet/bla.jsp - matches org/springframework/servlet/bla.jsp but also org/springframework/testing/servlet/bla.jsp and org/servlet/bla.jsp

看一下其核心的方法:

1. */\*\**
2. *\* Actually match the given <code>path</code> against the given <code>pattern</code>.*
3. *\* @param pattern the pattern to match against*
4. *\* @param path the path String to test*
5. *\* @param fullMatch whether a full pattern match is required (else a pattern match*
6. *\* as far as the given base path goes is sufficient)*
7. *\* @return <code>true</code> if the supplied <code>path</code> matched, <code>false</code> if it didn't*
8. *\*/*
9. protected boolean doMatch(String pattern, String path, boolean fullMatch,
10. Map<String, String> uriTemplateVariables) {
12. if (path.startsWith(this.pathSeparator) != pattern.startsWith(this.pathSeparator)) {
13. return false;
14. }
16. String[] pattDirs = StringUtils.tokenizeToStringArray(pattern, this.pathSeparator);
17. String[] pathDirs = StringUtils.tokenizeToStringArray(path, this.pathSeparator);
19. int pattIdxStart = 0;
20. int pattIdxEnd = pattDirs.length - 1;
21. int pathIdxStart = 0;
22. int pathIdxEnd = pathDirs.length - 1;
24. *// Match all elements up to the first \*\**
25. while (pattIdxStart <= pattIdxEnd && pathIdxStart <= pathIdxEnd) {
26. String patDir = pattDirs[pattIdxStart];
27. if ("\*\*".equals(patDir)) {
28. break;
29. }
30. if (!matchStrings(patDir, pathDirs[pathIdxStart], uriTemplateVariables)) {
31. return false;
32. }
33. pattIdxStart++;
34. pathIdxStart++;
35. }
37. if (pathIdxStart > pathIdxEnd) {
38. *// Path is exhausted, only match if rest of pattern is \* or \*\*'s*
39. if (pattIdxStart > pattIdxEnd) {
40. */\*\* /a/b/c and /a/b/c/ ==>false \*\*/*
41. return (pattern.endsWith(this.pathSeparator) ? path.endsWith(this.pathSeparator) :
42. !path.endsWith(this.pathSeparator));
43. }
44. if (!fullMatch) {
45. return true;
46. }

*/\*\* /a/b/c/\* and /a/b/c/ ==>true /a/b/c/\* and /a/b/c ==> false \*\*/*

1. if (pattIdxStart == pattIdxEnd && pattDirs[pattIdxStart].equals("\*") && path.endsWith(this.pathSeparator)) {
2. return true;
3. } */\*\* /a/b/c/\*\*/*\*\*/ and /a/b/c ==> true /a/b/c*/\*\*/a/ and /a/b/c ==>false \*\*/*
4. for (int i = pattIdxStart; i <= pattIdxEnd; i++) {
5. if (!pattDirs[i].equals("\*\*")) {
6. return false;
7. }
8. }
9. return true;
10. }
11. else if (pattIdxStart > pattIdxEnd) {
12. *// String not exhausted, but pattern is. Failure.*
13. return false;
14. }
15. else if (!fullMatch && "\*\*".equals(pattDirs[pattIdxStart])) {
16. *// Path start definitely matches due to "\*\*" part in pattern.*
17. return true;
18. }
20. *// up to last '\*\*'*
21. while (pattIdxStart <= pattIdxEnd && pathIdxStart <= pathIdxEnd) {
22. String patDir = pattDirs[pattIdxEnd];
23. if (patDir.equals("\*\*")) {
24. break;
25. }
26. if (!matchStrings(patDir, pathDirs[pathIdxEnd], uriTemplateVariables)) {
27. return false;
28. }
29. pattIdxEnd--;
30. pathIdxEnd--;
31. }
32. if (pathIdxStart > pathIdxEnd) { */\*\* /\*\*/*a/b/c and /a/b/c ==> true \*\*/
33. *// String is exhausted*
34. for (int i = pattIdxStart; i <= pattIdxEnd; i++) {
35. if (!pattDirs[i].equals("\*\*")) {
36. return false;
37. }
38. }
39. return true;
40. } */\*\* /a/b/c/d/\*\*/*\*\*/e/f/g/h*/\*\*/i/j/k/\*\*/*l/m/n*/\*\*/o/p/q and /a/b/c/d/a/b/e/f/g/h/a/b/i/j/k/a/b/l/m/n/a/b/o/p/q ==> true \*\*/*
41. while (pattIdxStart != pattIdxEnd && pathIdxStart <= pathIdxEnd) {
42. int patIdxTmp = -1;
43. for (int i = pattIdxStart + 1; i <= pattIdxEnd; i++) {
44. if (pattDirs[i].equals("\*\*")) {
45. patIdxTmp = i;
46. break;
47. }
48. }
49. if (patIdxTmp == pattIdxStart + 1) {
50. *// '\*\*/\*\*' situation, so skip one*
51. pattIdxStart++;
52. continue;
53. }
54. *// Find the pattern between padIdxStart & padIdxTmp in str between*
55. *// strIdxStart & strIdxEnd*
56. int patLength = (patIdxTmp - pattIdxStart - 1);
57. int strLength = (pathIdxEnd - pathIdxStart + 1);
58. int foundIdx = -1;
60. strLoop:
61. for (int i = 0; i <= strLength - patLength; i++) {
62. for (int j = 0; j < patLength; j++) {
63. String subPat = pattDirs[pattIdxStart + j + 1];
64. String subStr = pathDirs[pathIdxStart + i + j];
65. if (!matchStrings(subPat, subStr, uriTemplateVariables)) {
66. continue strLoop;
67. }
68. }
69. foundIdx = pathIdxStart + i;
70. break;
71. }
73. if (foundIdx == -1) {
74. return false;
75. }
77. pattIdxStart = patIdxTmp;
78. pathIdxStart = foundIdx + patLength;
79. }
81. for (int i = pattIdxStart; i <= pattIdxEnd; i++) {
82. if (!pattDirs[i].equals("\*\*")) {
83. return false;
84. }
85. }
87. return true;
88. }

其实现原理如下:

1) 先把Pattern 和要与之匹配的字符窜 Path 进行 转化为 String Array, 分隔符为  "/"

    例如: /a/\*\*/b/c/d/  ==>["a","\*\*","b","c","d"]

1. String[] pattDirs = StringUtils.tokenizeToStringArray(pattern, this.pathSeparator);
2. String[] pathDirs = StringUtils.tokenizeToStringArray(path, this.pathSeparator);
4. int pattIdxStart = 0;
5. int pattIdxEnd = pattDirs.length - 1;
6. int pathIdxStart = 0;
7. int pathIdxEnd = pathDirs.length - 1;

2) 先匹配 Pattern 中"\*\*" 前的元素 ,这里说的元素是指两数组的元素,

   // Match all elements up to the first \*\*

1. while (pattIdxStart <= pattIdxEnd && pathIdxStart <= pathIdxEnd) {
2. String patDir = pattDirs[pattIdxStart];
3. if ("\*\*".equals(patDir)) {
4. break;
5. }
6. if (!matchStrings(patDir, pathDirs[pathIdxStart], uriTemplateVariables)) {
7. return false;
8. }
9. pattIdxStart++;
10. pathIdxStart++;
11. }

    跳出这个loop的条件是,

     1.pattIdxStart>pattIdxEnd  (没有找到"\*\*")

       说明在还没有找到有 "\*\*"之前Pattern 在循环中溢出,即Pattern的长度比Path的长度要短,则结果肯定不匹配,应直接返回false

1. else if (pattIdxStart > pattIdxEnd) {
2. *// String not exhausted, but pattern is. Failure.*
3. return false;
4. }

   2.pathIdxStart > pathIdxEnd (没有找到"\*\*")

     则在还没有找到 "\*\*" 之前 Path在循环中溢出, 即Path的长度比Pattern的短,或者两者长度一样,

    如果长度一样,则比较两者的最后一个字符是否相等,因为在转化为Array的时候,最后的字符'/' 会被 省略,所以,数组的元素匹配不一定说明最后的一个字符会相等,这时就需要判断是否一样,如果一样,则返回true,如果不一样,则返回false,

  第二种情况就是Pattern的长度比Path长, 那匹配成功的情况就只有两种, 一是Pattern以\* 号或者是\*\* 号结尾,其中如果

 是Pattern是以\* 号结尾, Pattern 必须比Path只大一个元素,而且Path 的最后一样字符需为"/",因为 \* 号只代表 0-n 个字符, 并没有 代表 "Directory" 的意思, Path和Pattern 的Directory 的深度必须一样, 所以判定结果如果 Path是以 "/" 结尾 并而且Pattern只比Path的元素少一个\* 号,则返回true

 如果Pattern比Path多一个以上的元素, 则后面的元素必须为 "\*\*" 号, 否则返回false

     if (pathIdxStart > pathIdxEnd) {

1. *// Path is exhausted, only match if rest of pattern is \* or \*\*'s*
2. if (pattIdxStart > pattIdxEnd) {
3. return (pattern.endsWith(this.pathSeparator) ? path.endsWith(this.pathSeparator) :
4. !path.endsWith(this.pathSeparator));
5. }
6. if (!fullMatch) {
7. return true;
8. }
9. if (pattIdxStart == pattIdxEnd && pattDirs[pattIdxStart].equals("\*") && path.endsWith(this.pathSeparator)) {
10. return true;
11. }
12. for (int i = pattIdxStart; i <= pattIdxEnd; i++) {
13. if (!pattDirs[i].equals("\*\*")) {
14. return false;
15. }
16. }
17. return true;
18. }
19. else if (pattIdxStart > pattIdxEnd) {
20. *// String not exhausted, but pattern is. Failure.*
21. return false;
22. }
23. else if (!fullMatch && "\*\*".equals(pattDirs[pattIdxStart])) {
24. *// Path start definitely matches due to "\*\*" part in pattern.*
25. return true;
26. }

3) 然后后匹配 Pattern  "\*\*" 之后的元素, 如果其中匹配不成功,则返回false

   这里有一个特殊的情况, 就是在没有找到 \*\* 号之前, Path 字符串提前退出,

   特例 /\*\*/\*\*/\*\*/a/b/c  和  /a/b/c  ==>true

        /\*\*/a/b/\*\*/a/b/c   和 /a/b/c  ==> false

   则判断Pattern的pattIdxStart 到pattIdxEnd 之间是否全部为 \*\* 号, 如果不是 , 则返回false

    // up to last '\*\*'

1. while (pattIdxStart <= pattIdxEnd && pathIdxStart <= pathIdxEnd) {
2. String patDir = pattDirs[pattIdxEnd];
3. if (patDir.equals("\*\*")) {
4. break;
5. }
6. if (!matchStrings(patDir, pathDirs[pathIdxEnd], uriTemplateVariables)) {
7. return false;
8. }
9. pattIdxEnd--;
10. pathIdxEnd--;
11. }
12. if (pathIdxStart > pathIdxEnd) {
13. *// String is exhausted*
14. for (int i = pattIdxStart; i <= pattIdxEnd; i++) {
15. if (!pattDirs[i].equals("\*\*")) {
16. return false;
17. }
18. }
19. return true;
20. }

   第二种情况, Pattern和Path都没提前溢出退出循环, 则属于 这种情况

 /a/b/c/d/\*\*/\*\*/e/f/g/h/\*\*/i/j/k/\*\*/l/m/n/\*\*/o/p/q and           /a/b/c/d/a/b/e/f/g/h/a/b/i/j/k/a/b/l/m/n/a/b/o/p/q ==> true

这时候要处理多个 \*\* 号的, 分段处理, \*\* 号之间的字符匹配, 如果 \*\* 号之间的字符不匹配, 则返回false

第一段:

/a/b/c/d/\*\*/\*\*/e/f/g/h/\*\*/

and

a/b/e/f/g/h/a/b/i/j/k/a/b/l/m/n/a/b/o/p/q

==>true

第二段:

/\*\*/i/j/k/\*\*/l

and

/a/b/i/j/k/a/b/l/m/n/a/b/o/p/q

==>true

第三段:

/\*\*/l/m/n/\*\*/

/a/b/l/m/n/a/b/o/p/q

==>true

1. while (pattIdxStart != pattIdxEnd && pathIdxStart <= pathIdxEnd) {
2. int patIdxTmp = -1;
3. for (int i = pattIdxStart + 1; i <= pattIdxEnd; i++) {
4. if (pattDirs[i].equals("\*\*")) {
5. patIdxTmp = i;
6. break;
7. }
8. }
9. if (patIdxTmp == pattIdxStart + 1) {
10. *// '\*\*/\*\*' situation, so skip one*
11. pattIdxStart++;
12. continue;
13. }
14. *// Find the pattern between padIdxStart & padIdxTmp in str between*
15. *// strIdxStart & strIdxEnd*
16. int patLength = (patIdxTmp - pattIdxStart - 1);
17. int strLength = (pathIdxEnd - pathIdxStart + 1);
18. int foundIdx = -1;
20. strLoop:
21. for (int i = 0; i <= strLength - patLength; i++) {
22. for (int j = 0; j < patLength; j++) {
23. String subPat = pattDirs[pattIdxStart + j + 1];
24. String subStr = pathDirs[pathIdxStart + i + j];
25. if (!matchStrings(subPat, subStr, uriTemplateVariables)) {
26. continue strLoop;
27. }
28. }
29. foundIdx = pathIdxStart + i;
30. break;
31. }
33. if (foundIdx == -1) {
34. return false;
35. }
37. pattIdxStart = patIdxTmp;
38. pathIdxStart = foundIdx + patLength;
39. }