

(12) 发明专利

(10) 授权公告号 CN 101901268 B

(45) 授权公告日 2011. 12. 21

(21) 申请号 201010244613. 1

(22) 申请日 2010. 08. 02

(73) 专利权人 华为技术有限公司

地址 518129 广东省深圳市龙岗区坂田华为  
总部办公楼

(72) 发明人 曾佳 郭智 吴富强 孙灵燕

(74) 专利代理机构 深圳市深佳知识产权代理事  
务所(普通合伙) 44285

代理人 彭愿洁 李文红

(51) Int. Cl.

G06F 17/30(2006. 01)

审查员 史江峰

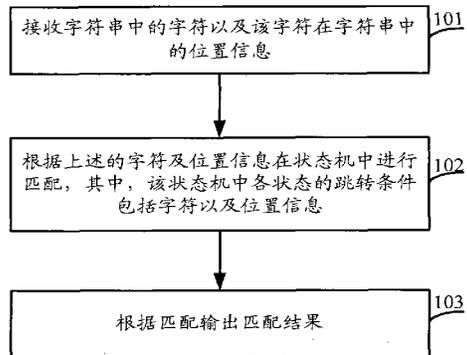
权利要求书 2 页 说明书 14 页 附图 5 页

(54) 发明名称

一种规则匹配方法及装置

(57) 摘要

本发明实施例公开了一种规则匹配方法及装置,用于提高规则匹配的效率。该方法包括:接收字符串中的字符以及该字符在上述字符串中的位置信息;根据上述字符及位置信息在状态机中进行匹配,其中,状态机中各状态的跳转条件包括字符以及位置信息;根据匹配输出匹配结果。本发明实施例可以减少无效匹配结果的输出,提高规则匹配的效率。



1. 一种规则匹配方法,其特征在于,包括:

接收字符串中的字符以及所述字符在所述字符串中的位置信息;

根据所述字符及位置信息在状态机中进行匹配,所述状态机中各状态的跳转条件包括字符以及位置信息;

根据匹配输出匹配结果;

所述根据所述字符及位置信息在状态机中进行匹配包括:

判断接收的字符与跳转条件中包括的字符是否相同,且接收的位置信息与跳转条件中包括的位置信息是否相同,如果均相同,判断所述跳转条件指向的状态是否为接收态,若是,则确定匹配到规则;若否,则确定匹配不到规则。

2. 根据权利要求1所述的方法,其特征在于,所述接收字符串中的字符以及所述字符在所述字符串中的位置信息之前,还包括:

生成所述状态机;

所述生成所述状态机包括:

将输入规则进行语法解析,并改写成状态机构建所需的状态机规则;

根据所述状态机规则进行状态机构建;

将构建的状态机转换成匹配引擎所需要的格式并存储;

所述根据所述状态机规则进行状态机构建包括:

根据所述状态机规则中的每个字符以及所述每个字符在所述状态机规则中的位置信息构建状态机;

其中,所述状态机中的初始状态至第二状态的跳转条件包括所述状态机规则中的首字符以及所述首字符的位置信息;所述状态机中的第二状态至第三状态的跳转条件包括所述状态机规则中的次字符以及所述次字符的位置信息;依此类推,所述状态机中的倒数第二个状态至最后一个状态的跳转条件包括所述状态机规则中的最后一个字符以及所述最后一个字符的位置信息。

3. 根据权利要求2所述的方法,其特征在于,所述将输入规则进行语法解析,并改写成状态机构建所需的状态机规则包括:

将输入规则进行语法解析,若所述输入规则是带有起始位置的纯字符串,则将所示输入规则改写成状态机构建所需的状态机规则“. {m} XXX”,其中,所述“. {m}”表示起始位置,“XXX”表示字符。

4. 根据权利要求2所述的方法,其特征在于,所述将输入规则进行语法解析,并改写成状态机构建所需的状态机规则包括:

将输入规则进行语法解析,若所述输入规则是以起始位置“. {m}”开始的,且包含“[]”,“()”,“|”中至少一个语法的正则表达式,则将所述正则表达式改写成与所述至少一个语法对应的若干个状态机规则;其中,每一个状态机规则以所述起始位置“. {m}”开始,且包含所述正则表达式的部分字符。

5. 一种规则匹配装置,其特征在于,包括:

接收单元,用于接收字符串中的字符以及所述字符在所述字符串中的位置信息;

匹配单元,用于根据所述字符及位置信息在状态机中进行匹配,并根据匹配输出匹配结果;所述状态机中各状态的跳转条件包括字符以及位置信息;

所述匹配单元包括：

判断子单元，用于判断接收的字符与跳转条件中包括的字符是否相同，且接收的位置信息与跳转条件中包括的位置信息是否相同，如果均相同，判断所述跳转条件指向的状态是否为接收态，若是，则确定匹配到规则；若否，则确定匹配不到规则。

6. 根据权利要求 5 所述的装置，其特征在于，还包括：

状态机生成单元，用于生成所述状态机；

所述状态机生成单元包括：

解析子单元，用于将输入规则进行语法解析，并改写成状态机构建所需的状态机规则；

构建子单元，用于根据所述状态机规则进行状态机构建；

存储子单元，用于将构建的状态机转换成匹配引擎所需要的格式并存储；

所述构建子单元具体用于根据所述状态机规则中的每个字符以及所述每个字符在所述状态机规则中的位置信息构建状态机；其中，所述状态机中的起始状态至第二状态的跳转条件包括所述状态机规则中的首字符以及所述首字符的位置信息；所述状态机中的第二状态至第三状态的跳转条件包括所述状态机规则中的次字符以及所述次字符的位置信息；依此类推，所述状态机中的倒数第二个状态至最后一个状态的跳转条件包括所述状态机规则中的最后一个字符以及所述最后一个字符的位置信息。

7. 根据权利要求 6 所述的装置，其特征在于，

所述解析子单元具体用于将输入规则进行语法解析，若所述输入规则是带有起始位置的纯字符串，则将所示输入规则改写成状态机构建所需的状态机规则“{m} XXX”，其中，所述“{m}”表示起始位置，“XXX”表示字符。

8. 根据权利要求 6 所述的装置，其特征在于，

所述解析子单元具体用于将输入规则进行语法解析，若所述输入规则是以起始位置“{m}”开始的，且包含“[]”，“()”，“|”中至少一个语法的正则表达式，则将所述正则表达式改写成与至少一个语法对应的若干个状态机规则；其中，每一个状态机规则以所述起始位置“{m}”开始，且包含所述正则表达式的部分字符。

## 一种规则匹配方法及装置

### 技术领域

[0001] 本发明涉及通信技术领域，具体涉及一种规则匹配方法及装置。

### 背景技术

[0002] 规则匹配技术作为一种关键的特征字识别技术，广泛地应用于 IP 网络深度报文检测 (Deep Packet Inspection, DPI) 当中。其中，特征字识别即对报文中一些特定的字符串进行识别，这些特定的字符串会用一些特殊的规则来表示，通常会使用正则表达式 (Regular Expression) 来表示这些特定的字符串，例如，正则表达式 `to(nite|knight|night)` 用于表示需要匹配的字符串为 `tonite` 或 `toknight` 或 `tonight`。所谓规则匹配即通过在给定的字符串当中查找是否存在符合这些正则表达式的目标字符串。

[0003] 现有技术中提供了一种规则匹配方法，该方法包括：

[0004] 1、接收输入的规则，该规则包括偏移位置及字符串；

[0005] 2、状态机匹配引擎根据上述的字符串在状态机中完成规则匹配并输出匹配结果；

[0006] 3、位置计数器根据上述的偏移位置计算该规则的偏移量；

[0007] 例如，该规则的偏移位置为 0，当输入该规则的第一个字符时，位置计数器计算值为 0；当接收到字符串中的第二个字符时，位置计数器计算值为 1；……；依此类推，直到接收完该规则所有的字符为止，位置计数器的累计值即为该规则的偏移量。

[0008] 4、规则匹配结果模块将位置计数器计算的该规则的偏移量与状态机匹配引擎匹配到的规则的偏移量进行比较，如果相等，则整个规则匹配成功，否则，整个规则匹配失败。

[0009] 发明人在实现本发明的过程中，发现现有技术至少存在如下问题：

[0010] 当输入的规则有大量的较短的规则时，例如 `\x00\x00\x00`（代表 16 进制的三个零，因为不是可文本显示的字符，所以使用 `\x` 进行转义表达），在实际规则匹配中极易匹配到，状态机匹配引擎会有大量的匹配结果输出，而匹配结果经过偏移量的过滤后剩下的有效匹配却很少，导致了大量的无效的匹配。

### 发明内容

[0011] 本发明实施例中提供一种规则匹配方法及装置，能够提高规则匹配的效率。

[0012] 一种规则匹配方法，包括：

[0013] 接收字符串中的字符以及所述字符在所述字符串中的位置信息；

[0014] 根据所述字符及位置信息在状态机中进行匹配，所述状态机中各状态的跳转条件包括字符以及位置信息；

[0015] 根据匹配输出匹配结果。

[0016] 一种规则匹配装置，包括：

[0017] 接收单元，用于接收字符串中的字符以及所述字符在所述字符串中的位置信息；

[0018] 匹配单元，用于根据所述字符及位置信息在状态机中进行匹配，并根据匹配输出

匹配结果；所述状态机中各状态的跳转条件包括字符以及位置信息。

[0019] 与现有的技术相比，本发明实施例具有以下有益效果：

[0020] 本发明实施例中，将字符以及位置信息作为状态机中各状态的跳转条件，进而在接收到字符以及该字符在字符串中的位置信息后，可以在状态机中进行匹配并输出匹配结果。利用本发明实施例提供的规则匹配方法，即使字符串中包含多个相同的规则，也无需输出多个匹配结果，而仅输出与跳转条件中的字符以及位置信息对应的规则的匹配结果即可。与现有技术相比，本发明实施例可以减少无效匹配结果的输出，提高规则匹配的效率。

## 附图说明

[0021] 为了更清楚地说明本发明实施例或现有技术中的技术方案，下面将对实施例中所需要使用的附图作简单地介绍，显而易见地，下面描述中的附图仅仅是本发明的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动的前提下，还可以根据这些附图获得其他的附图。

[0022] 图 1 为本发明实施例中提供的一种规则匹配方法的流程图；

[0023] 图 2 为本发明实施例中提供的一种状态机生成方法的流程图；

[0024] 图 3 为本发明实施例中提供的一种状态机的示意图；

[0025] 图 4 为本发明实施例中提供的一种状态机生成方法的流程图；

[0026] 图 5 为本发明实施例中提供的一种规则匹配方法的流程图；

[0027] 图 6 为本发明实施例中提供的另一种状态机的示意图；

[0028] 图 7 为本发明实施例中提供的另一种状态机的示意图；

[0029] 图 8 为本发明实施例中提供的另一种状态机的示意图；

[0030] 图 9 为本发明实施例中提供的一种规则匹配装置的结构图；

[0031] 图 10 为本发明实施例中提供的另一种规则匹配装置的结构图。

## 具体实施方式

[0032] 下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例仅仅是本发明一部分实施例，而不是全部的实施例。基于本发明中的实施例，本领域普通技术人员在没有做出创造性劳动前提下所获得的所有其他实施例，都属于本发明保护的范围。

[0033] 实施例一：

[0034] 请参阅图 1，图 1 为本发明实施例中提供的一种规则匹配方法的流程图。如图 1 所示，该方法可以包括以下步骤：

[0035] 101、接收字符串中的字符以及该字符在字符串中的位置信息；

[0036] 其中，字符串的输入可以是字符格式的，例如：“Example”，这类字符串可以表示所有的可显示的 ASCII 码；另外，字符串的输入也可以是用 \xhh 转义的 ASCII 码，例如：“\x01\x02\x0a”，这类字符串主要是用于表示无法显示的 ASCII 码，例如，回车，换行等。

[0037] 其中，位置信息的输入可以是任意的进制。例如，例如 \d(ddd) 代表十进制数 (ddd 代表具体的数值)，\x(hhh) 代表 16 进制数 (hhh 代表具体的数值)。如果只有两位，可以不用“()”。例如 \d10 代表十进制数 10。

[0038] 下面举例说明字符在字符串中的位置信息。假设字符串为 abc,则第一个字符 a 在字符串中的位置信息可以设置为 0,第二个字符 b 在字符串中的位置信息设置为 1,第三个字符 c 在字符串中的位置信息设置为 2。换句话说,在一个字符串中,后一字符的位置信息等于前一字符的位置信息加 1,第一个字符的位置信息一般设置为 0。

[0039] 本实施例中,可以配置一个位置计数器,用于为每个接收到的字符串中的字符进行计数,生成该字符在字符串中的位置信息。例如,位置计数器接收到字符串中的第一个字符时,位置计数器计算值为 0(0 表示第一个字符在字符串中的位置信息);当接收到字符串中的第二个字符时,位置计数器计算值为 1(1 表示第二个字符在字符串中的位置信息);……;依此类推,直到接收完字符串所有的字符为止。

[0040] 102、根据上述的字符及位置信息在状态机中进行匹配,其中,该状态机中各状态的跳转条件包括字符以及位置信息;

[0041] 其中,根据接收的字符及位置信息在状态机中进行匹配具体可以为:

[0042] 判断接收的字符与跳转条件中包括的字符是否相同,并且接收的位置信息与跳转条件中包括的位置信息是否相同,如果均相同,再判断该跳转条件指向的状态是否为接收态,如果是接收态,则确定匹配到规则;如果不是接收态,则确定尚未匹配到规则。

[0043] 反之,如果接收的字符与跳转条件中包括的字符不相同,或者接收的位置信息与跳转条件中包括的位置信息不相同,又或者接收的字符与跳转条件中包括的字符不相同,并且接收的位置信息与跳转条件中包括的位置信息不相同,则将接收的字符及其位置信息丢弃,继续接收下一字符及其位置信息。

[0044] 其中,根据实际应用载体不同,状态机可以采用数据表项、链表、指令表项、状态图等方式来表示,本实施例不作限定。

[0045] 103、根据匹配输出匹配结果。

[0046] 其中,如果匹配到规则,可以输出匹配到的规则的标识,如果匹配不到规则,可以输出匹配错误提示。

[0047] 本实施例中,可以在上述步骤 101 之前生成状态机。如图 2 所示,生成状态机具体可以包括以下步骤:

[0048] 201、将输入规则进行语法解析,并改写成状态机构建所需的状态机规则;

[0049] 其中,将输入规则进行语法解析,若该输入规则是带有起始位置的纯字符串,则可以将该输入规则改写成状态机构建所需的状态机规则“. {m} XXX”,其中,“. {m}”表示起始位置,“XXX”表示字符。例如,起始位置为 \d60,字符串为“http”的输入规则可以改写成状态机构建所需的状态机规则:“. {60} http”。

[0050] 其中,将输入规则进行语法解析,若该输入规则是以起始位置“. {m}”开始的,且包含“[]”,“()”,“|”中至少一个语法的正则表达式,则将该输入规则改写成与上述的至少一个语法对应的若干个状态机规则;其中,每一个状态机规则以起始位置“. {m}”开始,且包含上述的输入规则的部分字符。

[0051] 具体地,语法“[]”表示范围的意思,即方括号所示字符集中的任意一个字符,例如 [a-z] 表示 a 到 z 中的任意一个;“()”表示小括号中的字符串作为一个整体,不可分割,例如 (abc) 表示 abc 作为一个不可分割的整体;“|”表示“或”的意思,例如 a|b 代表是 a 或 b。

[0052] 下面举例说明输入规则为正则表达式时,如何将输入规则进行语法解析,并改写成状态机构建所需的状态机规则:

[0053] 举例一、将输入规则进行语法解析,若该输入规则是以起始位置“ $\cdot$ ”开始的,且包含语法“ $[]$ ”的正则表达式,则将该输入规则改写成若干个状态机规则;其中,若干个状态机规则的数量与语法“ $[]$ ”范围相对应;每一个状态机规则以起始位置“ $\cdot$ ”开始,并包含上述输入规则的部分字符。

[0054] 例如,输入规则为正则表达式:“ $\cdot \{10\} [abc]$ ”,则将该输入规则改写成与语法“ $[]$ ”范围相对应的 3 个状态机规则,分别是“ $\cdot \{10\} a$ ”,“ $\cdot \{10\} b$ ”以及“ $\cdot \{10\} c$ ”。其中,每一个状态机规则分别包含上述输入规则的部分字符。

[0055] 举例二,将输入规则进行语法解析,若该输入规则是以起始位置“ $\cdot$ ”开始的,且包含语法“ $|$ ”的正则表达式,则将该输入规则改写成若干个状态机规则;其中,若干个状态机规则的数量等于输入规则中出现的语法“ $|$ ”的数量加 1;其中,每一个状态机规则以起始位置“ $\cdot$ ”开始,并包含上述输入规则的部分字符。

[0056] 例如,输入规则为正则表达式:“ $\cdot \{10\} a|b|c$ ”,则将该输入规则改写成 3 个状态机规则,分别为:“ $\cdot \{10\} a$ ”、“ $\cdot \{10\} b$ ”、“ $\cdot \{10\} c$ ”。其中,状态机规则的数量为 3,等于 2 个语法“ $|$ ”的数量加 1,且每一个状态机规则分别包含上述输入规则的部分字符。

[0057] 例如,输入规则为正则表达式:“ $\cdot \{3\} abc| \cdot \{6\} mnx$ ”,则将该输入规则改写成 2 个状态机规则,分别为:“ $\cdot \{10\} abc$ ”和“ $\cdot \{6\} mnx$ ”。其中,状态机规则的数量为 2,等于 1 个语法“ $|$ ”的数量加 1,且每一个状态机规则分别包含上述输入规则的部分字符。

[0058] 举例三,将输入规则进行语法解析,若该输入规则是以起始位置“ $\cdot$ ”开始的,且包含语法“ $[]$ ”,“ $()$ ”的正则表达式,则将该输入规则改写成与上述语法“ $[]$ ”,“ $()$ ”对应的若干个状态机规则;其中,每一个状态机规则以起始位置“ $\cdot$ ”开始,且包含上述的输入规则的部分字符。

[0059] 例如,输入规则为正则表达式:“ $\cdot \{10\} [abc] (ef)$ ”,则将该输入规则改写成与上述语法“ $[]$ ”,“ $()$ ”对应的 3 个状态机规则,分别为:“ $\cdot \{10\} aef$ ”,“ $\cdot \{10\} bef$ ”,“ $\cdot \{10\} cef$ ”,其中,每一个状态机规则分别包含上述输入规则的部分字符。

[0060] 举例四,将输入规则进行语法解析,若该输入规则是以起始位置“ $\cdot$ ”开始的,且包含语法“ $[]$ ”,“ $()$ ”,“ $|$ ”的正则表达式,则将该输入规则改写成与上述语法“ $[]$ ”,“ $()$ ”,“ $|$ ”对应的若干个状态机规则;其中,每一个状态机规则以起始位置“ $\cdot$ ”开始,且包含上述的输入规则的部分字符。

[0061] 例如,输入规则为正则表达式:“ $\cdot \{10\} [abc] (ef)g|h$ ”,则将该输入规则改写成与上述语法“ $[]$ ”,“ $()$ ”,“ $|$ ”对应的 6 个状态机规则,分别为:“ $\cdot \{10\} aefg$ ”、“ $\cdot \{10\} befg$ ”、“ $\cdot \{10\} cefg$ ”、“ $\cdot \{10\} aefh$ ”、“ $\cdot \{10\} befh$ ”、“ $\cdot \{10\} cefh$ ”,且每一个状态机规则分别包含上述输入规则的部分字符。

[0062] 202、根据状态机规则进行状态机构建;

[0063] 其中,根据状态机规则进行状态机构建具体可以为:

[0064] 根据状态机规则中的每个字符以及每个字符在状态机规则中的位置信息构建状态机;其中,状态机中的起始状态至第二状态的跳转条件包括状态机规则中的首字符以及首字符的位置信息;状态机中的第二状态至第三状态的跳转条件包括状态机规则中的次字

符以及次字符的位置信息；……；依此类推，状态机中的倒数第二个状态至最后一个状态的跳转条件包括状态机规则中的最后一个字符以及最后一个字符的位置信息。

[0065] 下面举例说明如何根据状态机规则进行状态机构建：

[0066] 假设，初始条件下存在以起始位置  $\cdot\{0\}$  开始的 3 个状态机规则，分别是“ $\cdot\{0\}$  unset”，“ $\cdot\{2\}$  setup”以及“ $\cdot\{8\}$  test”。其中，状态机规则“ $\cdot\{0\}$  unset”的标识为  $r\langle 1\rangle$ ，状态机规则“ $\cdot\{2\}$  setup”的标识为  $r\langle 2\rangle$ ，状态机规则“ $\cdot\{8\}$  test”的标识为  $r\langle 3\rangle$ ，则分别根据这 3 个状态机规则进行状态机构建具体可以为：

[0067] 请参阅图 3，在状态 0（又称起始状态）下，接收状态机规则“ $\cdot\{0\}$  unset”输入的第一个字符“u”及其位置信息“0”，并将字符“u”及其位置信息“0”作为状态 0 到状态 1 的跳转条件；如图 3 所示，状态 0 到状态 1 的跳转条件可以表示为： $\{0, u\}$ ；本实施例中，字符“u”也可以用 ASCII 码“0x75”表示；

[0068] 在状态 1 下，接收状态机规则“ $\cdot\{0\}$  unset”输入的第二个字符“n”及其位置信息“1”，并将字符“n”及其位置信息“1”作为状态 1 到状态 2 的跳转条件；如图 3 所示，状态 1 到状态 2 的跳转条件可以表示为： $\{1, n\}$ ；本实施例中，字符“n”也可以用 ASCII 码“0x6e”表示；

[0069] 在状态 2 下，接收状态机规则“ $\cdot\{0\}$  unset”输入的第三个字符“s”及其位置信息“2”，并将字符“s”及其位置信息“2”作为状态 2 到状态 3 的跳转条件；如图 3 所示，状态 2 到状态 3 的跳转条件可以表示为： $\{2, s\}$ ；本实施例中，字符“s”也可以用 ASCII 码“0x73”表示；

[0070] 在状态 3 下，接收状态机规则“ $\cdot\{0\}$  unset”输入的第四个字符“e”及其位置信息“3”，并将字符“e”及其位置信息“3”作为状态 3 到状态 4 的跳转条件；如图 3 所示，状态 3 到状态 4 的跳转条件可以表示为： $\{3, e\}$ ；本实施例中，字符“e”也可以用 ASCII 码“0x65”表示；

[0071] 在状态 4 下，接收状态机规则“ $\cdot\{0\}$  unset”输入的第五个字符“t”及其位置信息“4”，并将字符“t”及其位置信息“4”作为状态 4 到状态 5 的跳转条件；如图 3 所示，状态 4 到状态 5 的跳转条件可以表示为： $\{4, t\}$ ；本实施例中，字符“t”也可以用 ASCII 码“0x74”表示。

[0072] 至此，根据状态机规则“ $\cdot\{0\}$  unset”构建状态机完毕。如图 3 所示，状态 5 中包含标识  $r\langle 1\rangle$ ，用于表示状态 5 为接收态，即在状态 5 下可以匹配到标识为  $r\langle 1\rangle$  的规则“ $\cdot\{0\}$  unset”。

[0073] 请参阅图 3，在状态 0（又称起始状态）下，接收状态机规则“ $\cdot\{2\}$  setup”输入的第一个字符“s”及其位置信息“2”，并将字符“s”及其位置信息“2”作为状态 0 到状态 6 的跳转条件；如图 3 所示，状态 0 到状态 6 的跳转条件可以表示为： $\{2, s\}$ ；本实施例中，字符“s”也可以用 ASCII 码“0x73”表示；

[0074] 在状态 6 下，接收状态机规则“ $\cdot\{2\}$  setup”输入的第二个字符“e”及其位置信息“3”，并将字符“e”及其位置信息“3”作为状态 6 到状态 7 的跳转条件；如图 3 所示，状态 6 到状态 7 的跳转条件可以表示为： $\{3, e\}$ ；本实施例中，字符“e”也可以用 ASCII 码“0x65”表示；

[0075] 在状态 7 下，接收状态机规则“ $\cdot\{2\}$  setup”输入的第三个字符“t”及其位置信息

“4”，并将字符“t”及其位置信息“4”作为状态 7 到状态 8 的跳转条件；如图 3 所示，状态 7 到状态 8 的跳转条件可以表示为： $\{4, t\}$ ；本实施例中，字符“t”也可以用 ASCII 码“0x74”表示；

[0076] 在状态 8 下，接收状态机规则“. {2} setup”输入的第四个字符“u”及其位置信息“5”，并将字符“u”及其位置信息“5”作为状态 8 到状态 9 的跳转条件；如图 3 所示，状态 8 到状态 9 的跳转条件可以表示为： $\{5, u\}$ ；本实施例中，字符“u”也可以用 ASCII 码“0x75”表示；

[0077] 在状态 9 下，接收状态机规则“. {0} setup”输入的第五个字符“p”及其位置信息“6”，并将字符“p”及其位置信息“6”作为状态 9 到状态 10 的跳转条件；如图 3 所示，状态 9 到状态 10 的跳转条件可以表示为： $\{6, p\}$ ；本实施例中，字符“p”也可以用 ASCII 码“0x70”表示。

[0078] 至此，根据状态机规则“. {2} setup”构建状态机完毕。如图 3 所示，状态 10 中包含标识  $r\langle 2 \rangle$ ，用于表示状态 10 为接收态，即在状态 10 下可以匹配到标识为  $r\langle 2 \rangle$  的规则“. {2} setup”。

[0079] 请参阅图 3，在状态 0（又称起始状态）下，接收状态机规则“. {8} test”输入的第一个字符“t”及其位置信息“8”，并将字符“t”及其位置信息“8”作为状态 0 到状态 11 的跳转条件；如图 3 所示，状态 0 到状态 11 的跳转条件可以表示为： $\{8, t\}$ ；本实施例中，字符“t”也可以用 ASCII 码“0x74”表示；

[0080] 在状态 11 下，接收状态机规则“. {8} test”输入的第二个字符“e”及其位置信息“9”，并将字符“e”及其位置信息“9”作为状态 11 到状态 12 的跳转条件；如图 3 所示，状态 11 到状态 12 的跳转条件可以表示为： $\{9, e\}$ ；本实施例中，字符“e”也可以用 ASCII 码“0x65”表示；

[0081] 在状态 12 下，接收状态机规则“. {8} test”输入的第三个字符“s”及其位置信息“10”，并将字符“s”及其位置信息“10”作为状态 12 到状态 13 的跳转条件；如图 3 所示，状态 12 到状态 13 的跳转条件可以表示为： $\{10, s\}$ ；本实施例中，字符“s”也可以用 ASCII 码“0x73”表示；

[0082] 在状态 13 下，接收状态机规则“. {8} test”输入的第四个字符“t”及其位置信息“11”，并将字符“t”及其位置信息“11”作为状态 13 到状态 14 的跳转条件；如图 3 所示，状态 13 到状态 14 的跳转条件可以表示为： $\{11, t\}$ ；本实施例中，字符“t”也可以用 ASCII 码“0x74”表示。

[0083] 至此，根据状态机规则“. {8} test”构建状态机完毕。如图 3 所示，状态 14 中包含标识  $r\langle 3 \rangle$ ，用于表示状态 14 为接收态，即在状态 14 下可以匹配到标识为  $r\langle 3 \rangle$  的规则“. {8} test”。

[0084] 假设初始条件下还存在以起始位置 . {0} 开始的状态机规则“. {0} unsetup”，则可以在状态 5 下，继续接收状态机规则“. {0} unsetup”输入的第六个字符“u”及其位置信息“5”，并将字符“u”及其位置信息“5”作为状态 5 到状态 9 的跳转条件；如图 3 所示，状态 5 到状态 9 的跳转条件可以表示为： $\{5, u\}$ ；本实施例中，字符“u”也可以用 ASCII 码“0x75”表示。即状态机规则“. {0} unsetup”与状态机规则“. {0} setup”在构建状态机时，存在状态机的尾部合并 (MergeTail)，可以进一步优化状态机。

[0085] 203、将构建的状态机转换成匹配引擎所需要的格式并存储。

[0086] 实际应用中,匹配引擎的实现方式是多种多样的。例如匹配引擎可以使用软件,现场可编程门阵列(Field-Programmable Gate Array, FPGA),或专用商业芯片等实现匹配。不同的实现方式需要不同的状态机格式。因此,根据匹配引擎的实现方式的不同,可生成以下状态机格式并在存储在存储器(Memory)中:

[0087] (1) 如果匹配引擎直接使用软件来实现匹配,则 Memory 里可以直接以链表形式将状态机存储;

[0088] (2) 如果匹配引擎使用 FPGA 来实现匹配,则 Memory 可以根据 FPGA 匹配特点,转换成特定格式存储;

[0089] (3) 如果匹配引擎使用商用芯片来实现匹配,则 Memory 可以以商业芯片认识的指令将状态机存储。

[0090] 以图 3 所示的状态机为例,当需要对输入的字符串 unset 进行匹配时,具体操作如下:

[0091] 第一个输入:字符为 u,位置信息为 0,匹配引擎从状态 0 跳到状态 1;

[0092] 第二个输入:字符为 n,位置信息为 1,匹配引擎从状态 1 跳到状态 2;

[0093] 第三个输入:字符为 s,位置信息为 2,匹配引擎从状态 2 跳到状态 3;

[0094] 第四个输入:字符为 e,位置信息为 3,匹配引擎从状态 3 跳到状态 4;

[0095] 第五个输入:字符为 t,位置信息为 4,匹配引擎从状态 4 跳到状态 5。

[0096] 因为状态 5 是接收态,匹配引擎上报匹配到了规则“. {0} unset”。

[0097] 本实施例中,将字符以及位置信息作为状态机中各状态的跳转条件,进而在接收到字符以及该字符在字符串中的位置信息后,可以在状态机中进行匹配并输出匹配结果。例如对于规则“. {2} abc”,如果利用现有技术的匹配方法进行匹配,当输入字符串包含 4 个相同的规则“abc”,即输入字符串为“abcabcabcabc”时,则可以匹配到 4 次“abc”并输出匹配结果,而仅有第二次匹配到的“abc”符合规则“. {2} abc”的起始位置,真正成功匹配到规则“. {2} abc”。利用本发明实例提供的方法,在第一个 abc 输入的时候,因为位置信息不对,状态机不会启动;第二个 abc 输入的时候,位置信息正确,状态机启动且匹配成功;第三、第四个 abc 输入的时候因为位置信息不对,状态机不会启动。所以,本发明实施例可以根据接收的字符以及位置信息在状态机中进行匹配,减少无效匹配结果的输出,提高规则匹配的效率。

[0098] 实施例二:

[0099] 请参阅图 4,图 4 为本发明实施例中提供的一种状态机生成方法的流程图。在图 4 所示的状态机生成方法中,采用状态机编译器来生成状态机。如图 4 所示,该状态机生成方法可以包括以下步骤:

[0100] 401、状态机编译器获取规则;

[0101] 本实施例中,规则可以预先存储在服务器或本地存储器的规则库中。状态机编译器可以接收服务器或本地存储器输出的规则,例如,服务器或本地存储器可以周期性地输入规则;或者,状态机编译器也可以通过其他方式主动从服务器或本地存储器中读取规则。

[0102] 本实施例中,规则是具有起始位置的。例如,起始位置+纯字符串的规则、正则表达式“. {10} a|b|c”、“. {3} abc|. {6} mnx”等等。

[0103] 402、状态机编译器将获取的规则进行语法解析,如果语法正确,则执行步骤 403;如果语法错误,则上报错误提示;

[0104] 其中,状态机编译器可以通过其解析模块来实现输入规则的语法解析。如果解析模块解析发现输入规则的语法与系统采用的正则语法(如 PCRE 语法)不符,则说明输入规则的语法错误;反之,如果相符,则说明输入规则的语法正确。

[0105] 403、状态机编译器将获取的规则进行语法解析,并改写成状态机构建所需的状态机规则;

[0106] 其中,状态机编译器获取的规则可能是带有起始位置的纯字符串,也可能是以起始位置“. {m}”开始的,且包含“[]”,“()”,“|”这些语法的正则表达式。

[0107] 具体地,若状态机编译器获取的规则是带有起始位置的纯字符串,则可以将该输入规则改写成状态机构建所需的状态机规则“. {m} XXX”,其中,“. {m}”表示起始位置,“XXX”表示字符。例如,起始位置为\d60,字符串为“http”的输入规则可以改写成状态机构建所需的状态机规则:“. {60} http”。

[0108] 具体地,若状态机编译器获取的规则是以起始位置“. {m}”开始的,且包含“[]”,“()”,“|”这些语法的正则表达式,则将该正则表达式改写成与这些语法对应的若干个状态机规则;其中,每一个状态机规则以起始位置“. {m}”开始,且包含上述的正则表达式的部分字符。上述的实施例一中,已经详细举例说明如何将起始位置“. {m}”开始的,且包含“[]”,“()”,“|”这些语法的正则表达式改写成状态机构建所需的状态机规则,本实施例不作复述。

[0109] 其中,状态机编译器将获取的规则进行语法解析,并改写成状态机构建所需的状态机规则具体可以采用如下方法实现:

[0110] 1) 状态机编译器将获取的规则进行语法解析后,对于语法错误的规则可以从处理队列中直接删除,而对于语法规则正确的规则可以存放于规则集中,生成规则集 R;

[0111] 2) 在步骤 1) 生成的规则集 R 的基础上,再进一步将规则集 R 中的以起始位置“. {m}”开始的,包含“[]”,“()”,“|”这些语法的正则表达式改写成状态机构建所需的状态机规则,这些状态机规则与规则集 R 其余的规则(即带有起始位置的纯字符串)形成规则集 R1;

[0112] 其中,上述的步骤 2) 具体可以采用如下方法实现:

[0113] A) 建一个空集合 R2;

[0114] B) 对于步骤 1) 生成的规则集 R,继续对规则进行语法分析;

[0115] C) 如果规则中包含有“[]”,“()”,“|”这些语法,则将该规则改写成状态机构建所需的状态机规则后添加到 R2 中,并将该规则从规则集 R 中删除;

[0116] D) 否则,继续遍历规则集 R;

[0117] E) 是否已遍历完规则集 R,如果没有遍历完,则重复步骤 B) 至步骤 D),否则继续后面步骤;

[0118] F) 将 R2 与删除了正则表达式的 R 合并后生成 R1。

[0119] 404、状态机编译器将状态机规则进行状态机构建;

[0120] 其中,状态机编译器可以根据上述的步骤 2) 生成的规则集 R1,构建状态机;

[0121] 其中,规则集 R1 中的规则是状态机构建所需的状态机规则,例如“. {60} http”、

“.{0}abc”以及“.{0}\x00\x00\x00”等等。

[0122] 其中,状态机编译器将状态机规则进行状态机构建具体可以采用如下方法实现:

[0123] (1) 遍历上述步骤 2) 生成的规则集 R1;

[0124] (2) 对于每一条规则,从状态 0 开始根据规则的字符和位置信息建立分支;

[0125] 其中,规则的第一个字符的位置信息与规则的起始位置相同,后一个字符的位置信息等于前一个字符的位置信息 +1。

[0126] (3) 如果当前状态已经存在包括上述字符和位置信息的跳转条件,则沿着已有分支前进,位置信息 +1,字符前移;

[0127] (4) 如果当前状态没有存在包括上述字符和位置信息的跳转条件,则建立新的分支,位置信息 +1,字符前移;

[0128] (5) 如果当前规则已经完成,跳至步骤 (6),否则重复步骤 3),4);

[0129] (6) 如果规则集 R1 已经完成,跳至步骤 (7),否则重复步骤 3),4),5);

[0130] (7) 根据前面构建的状态机,计算失效跳转;

[0131] 其中,所谓的失效跳转是指状态机中从某一规则的一个状态跳转至下一状态时,该下一状态与其他规则的某一个状态重叠,该跳转称为失效跳转。如图 3 所示,状态 5 和状态 9 之间的跳转即为失效跳转。

[0132] (8) 遍历已构建的状态机的状态,如果当前状态 S 除了已有的跳转条件,遍历还可以跳转到其他状态 S1,增加状态 S 到状态 S1 的跳转条件;

[0133] (9) 如果已经遍历完状态机的所有状态,则退出,否则重复步骤 (8)。

[0134] 其中,状态机编译器将状态机规则进行状态机构建之后,可以对状态机进行如下的状态综合处理:

[0135] a) 根据已构建的状态机,对于规则集 R1 中的两个规则 r1 和 r2,且 r1 和 r2 具有相同的规则号;

[0136] B1 代表 r1 在状态机上的一个状态;

[0137] B2 代表 r2 在状态机上的一个状态;

[0138] b) 如果 B1、B2 在状态机上的深度(即距离起始状态 0 的距离)相同,则设置此深度为合并(Merge)的最大值;

[0139] c) 如果 B1 和 B2 都没有失效跳转边,则合并(Merge)的最小值为 0;

[0140] 其中,B1 和 B2 都没有失效跳转边,说明 B1 和 B2 之间没有跳转条件。

[0141] d) 否则,合并(Merge)的最小值为其中一个状态出现失效跳转的深度;

[0142] 如图 3 所示,状态 5 和状态 9 之间存在失效跳转,则合并(Merge)的最小值为 4(即状态 9 距离起始状态 0 的距离为 4 个状态)。

[0143] e) 对于最大值、最小值之间的状态,进行合并。

[0144] 本实施例中,对状态机进行状态综合处理是为了合并整个状态机中的相同状态,减小状态机,降低存储空间。如图 3 所示,合并(Merge)的最小值为 4(即状态 9 距离起始状态 0 的距离为 4 个状态),而合并(Merge)的最大值为 5(即状态 10 距离起始状态 0 的距离为 5 个状态),可以将状态 9 和状态 10 合并,减小状态机,降低存储空间。

[0145] 405、将构建的状态机转换成匹配引擎所需要的格式并存储。

[0146] 请参阅图 5,图 5 为本发明实施例中提供的一种规则匹配方法的流程图。在图 5 所

示的规则匹配方法中,采用状态机匹配引擎来进行规则的匹配。如图 5 所示,该规则匹配方法可以包括以下步骤:

[0147] 501、状态机匹配引擎接收字符串中的字符以及该字符在字符串中的位置信息;

[0148] 其中,状态机匹配引擎基于已构建的状态机工作。

[0149] 本实施例中,状态机匹配引擎的输入模块中可以设置一个专门的位置计数器,用于确定当前输入字符在字符串中的位置信息。其中,位置信息可以是正向或者反向的,反向值等于字符串总长度减去正向值。例如,状态机匹配引擎接收到字符串第一个字符时,位置计数器计算值为 0;接收到字符串第二个字符时,位置计数器计算值为 1;……;依此类推,直到接收完字符串所有的字符为止。当每一个字符串的所有字符都输入完毕,位置计数器可以清零。

[0150] 进一步地,如果位置信息已经大于状态机要求的最大位置信息时,可以终止字符输入,不再做匹配。

[0151] 502、状态机匹配引擎根据接收的字符以及该字符在字符串中的位置信息在状态机中进行匹配;

[0152] 本实施例中,状态机匹配引擎的匹配模块可以根据接收的字符以及该字符在字符串中的位置信息在状态机中进行匹配。

[0153] 假设,状态机规则“ $\cdot \{0\} \backslash x00 \backslash x00 \backslash x00$ ”和“ $\cdot \{5\} abc$ ”构建的状态机如图 6 所示;其中,状态机规则“ $\cdot \{0\} \backslash x00 \backslash x00 \backslash x00$ ”的标识为  $r<1>$ ,状态机规则“ $\cdot \{5\} abc$ ”的标识为  $r<2>$ 。

[0154] 当输入字符串“ $\backslash x00 \backslash x00 \backslash x00 \backslash x01 \backslash x02 abc eg$ ”时,

[0155] 第一个输入:字符为  $\backslash x00$ ,位置信息为 0,匹配引擎从状态 0 跳到状态 1;

[0156] 第二个输入:字符为  $\backslash x00$ ,位置信息为 1,匹配引擎从状态 1 跳到状态 2;

[0157] 第三个输入:字符为  $\backslash x00$ ,位置信息为 2,匹配引擎从状态 2 跳到状态 3。因为状态 3 是接收态,即匹配引擎上报匹配到了规则  $r<1>$ ;

[0158] 第四个输入:字符为  $\backslash x01$ ,位置信息为 3,状态 3 没有这样的跳转,匹配引擎回到状态 0;

[0159] 第五个输入:字符为  $\backslash x02$ ,位置信息为 4,状态 0 没有这样的跳转;引擎继续在状态 0;

[0160] 第六个输入:字符为 a,位置信息为 5,匹配引擎从状态 0 跳到状态 4;

[0161] 第七个输入:字符为 b,位置信息为 6,匹配引擎从状态 4 跳到状态 5;

[0162] 第八个输入:字符为 c,位置信息为 7,匹配引擎从状态 5 跳到状态 6。

[0163] 因为状态 6 是接收态,即匹配引擎匹配到了规则  $r<2>$ ;

[0164] 第九个输入:字符为 e,位置信息为 8,状态 6 没有这样的跳转;引擎回到在状态 0;

[0165] 第十个输入:字符为 g,位置信息为 9,状态 0 没有这样的跳转;引擎继续在状态 0。

[0166] 本实施例中,状态机匹配引擎中设置预先设置最大位置信息,这样当状态机中的位置信息已经超过最大位置信息时,可以停止匹配。

[0167] 例如,状态机匹配引擎中可以设置预先设置最大位置信息为 7,当状态机匹配引擎在状态 6 匹配到规则  $r<2>$  之后,可以停止匹配,因为后续接收的第九个输入的字符为 e 的位置信息超过了预先设置的最大位置信息为 7。

[0168] 假设状态机如图 6 所示,当输入字符串“ $abc \backslash x01 \backslash x02 \backslash x00 \backslash x00 \backslash x03 abc$ ”时,状态

机从状态 0 开始,一直没有对应的跳转,当输入字符的位置信息大于预先设置的最大位置信息时(例如最大位置信息为 7),状态机匹配引擎可以停止匹配。这样,通过“位置信息 + 字符”的控制,就可以去掉多余的无效匹配。例如,输入字符串“abc\x01\x02\x00\x00\x03abc”中有两个“abc”,仅仅从字符条件出发,都是可以满足规则  $r<2>$  的,但因为位置信息的不匹配,可以直接在匹配阶段去掉。

[0169] 另外,由于有些规则匹配时只关注其出现在某一位置时的匹配,对后续出现在其他位置的匹配并不关注;这样,有了位置信息后,就可以根据位置信息定位第一次出现的位置,后续就可以不用匹配了;如原先假设没有位置信息,有一个规则是“abc”,现在有了位置信息后,规则变为“010a,020b,030c”;如果现在有字符串“abcabcabc”,那么,如果按照原来的规则“abc”,这个字符串中的“abcabcabc”需要被匹配三次;假设只关注第一个 abc,那么用了规则“010a,020b,030c”后,“abcabcabc”中的第一个“abc”会被匹配到,而第二个、第三个“abc”就可以不用匹配了,这样就节省了匹配时间。

[0170] 503、状态机匹配引擎根据状态机中的匹配信息对结果进行处理。

[0171] 其中,如果匹配到规则,可以输出匹配到的规则的标识,如果匹配不到规则,可以输出匹配错误提示。

[0172] 本实施例中,如果生成的状态机没有计算失效跳转,则该状态机可以称为确定的有穷自动机(Deterministic finite automaton, DFA);如果生成的状态机计算失效跳转,则该状态机可以称为不确定有穷自动机(Non-deterministic finite automaton, NFA)。与 DFA 相比, NFA 可以减少状态机的存储量(包括状态数和跳转条件),有效抑制状态爆炸。其中,所谓的状态爆炸是指随着规则数量的增加,状态机的存储量呈乘数或级数增加。

[0173] 因此,本实施例中生成的状态机可以通过计算失效跳转生成 NFA,从而减少状态机的存储量,有效抑制状态爆炸。下面举例说明,本实施例生成的 NFA 可以减少状态机的存储量,有效抑制状态爆炸。

[0174] 例如,对于规则 . {3} abc,其对应的 NFA 为 :状态数 + 跳转条件 = 14 ;

[0175] DFA 为 :状态数 + 跳转条件 = 20 ;

[0176] 如果增加一条规则 . {2} facd 后,变为 :

[0177] . {3} abc

[0178] . {2} facd

[0179] 其对应的 NFA 为 :状态数 + 跳转条件 = 22 ;

[0180] 较前面单条规则的 NFA,存储增长  $(22-14)/14 = 57\%$  ;

[0181] 其对应的 DFA 为 :状态数 + 跳转条件 = 46 ;

[0182] 较前面单条规则的 DFA,存储增长  $(46-20)/20 = 130\%$  ;

[0183] 由上述例子可以看出,当规则数增加时,本实施例生成的 NFA 的存储基本是线性增长,而 DFA 的存储往往是乘数或级数增长的。所以随着规则数量的增加,本实施例生成的 NFA 可以减少状态机的存储量,有效抑制状态爆炸。

[0184] 本实施例中,将字符以及位置信息作为状态机中各状态的跳转条件,可以有效地减少失效跳转。

[0185] 例如,存在标识为  $r<1>$  的规则“. {0} abc”和标识为  $r<2>$  规则“. {10} bad”,其中,规则“. {0} abc”和规则“. {10} bad”构建的状态机如图 7 所示。其中,状态 0 与状态 1 之间

的跳转条件包括字符“a”和位置信息“0”，状态 1 与状态 2 之间的跳转条件包括字符“b”和位置信息“1”，状态 2 与状态 3 之间的跳转条件包括字符“c”和位置信息“2”；状态 0 与状态 4 之间的跳转条件包括字符“b”和位置信息“10”，状态 4 与状态 5 之间的跳转条件包括字符“a”和位置信息“11”，状态 5 与状态 6 之间的跳转条件包括字符“d”和位置信息“12”。其中，状态 3 是接收态，即匹配引擎匹配到了规则  $r<1>$ ；状态 6 是接收态，即匹配引擎匹配到了规则  $r<2>$ 。

[0186] 可见，在没有去掉位置信息之前，因为有位置信息的限制，两条规则构建的状态机中没有失效跳转。

[0187] 当去掉位置信息后，上述两个规则构建的状态机如图 8 所示。其中，状态 0 与状态 1 之间的跳转条件包括字符“a”，状态 1 与状态 2 之间的跳转条件包括字符“b”，状态 2 与状态 3 之间的跳转条件包括字符“c”；而状态 0 与状态 4 之间的跳转条件包括字符“b”，状态 4 与状态 5 之间的跳转条件包括字符“a”，状态 5 与状态 6 之间的跳转条件包括字符“d”；其中，状态 3 是接收态，即匹配引擎匹配到了规则  $r<1>$ ；因为状态 6 是接收态，即匹配引擎匹配到了规则  $r<2>$ 。如图 8 所示，去掉位置信息后，规则  $r<1>$  的字符“b”之后的状态就要多添加失效跳转 a，而规则  $r<2>$  的字符“a”之后的状态就要多添加失效跳转 c。当规则数量较大时，失效跳转会成为存储的主要部分，浪费存储空间。

[0188] 本实施例中，将字符以及位置信息作为状态机中各状态的跳转条件，进而在接收到字符以及该字符在字符串中的位置信息后，可以在状态机中进行匹配并输出匹配结果。与现有技术相比，本发明实施例可以减少无效匹配结果的输出，提高规则匹配的效率。

[0189] 实施例三：

[0190] 请参阅图 9，图 9 为本发明实施例中提供的一种规则匹配装置的结构图，用于实现上述的规则匹配方法。如图 9 所示，该规则匹配装置可以包括：

[0191] 接收单元 901，用于接收字符串中的字符以及该字符在字符串中的位置信息；

[0192] 其中，字符串的输入可以是字符格式的，例如：“Example”，这类字符串可以表示所有的可显示的 ASCII 码；另外，字符串的输入也可以是用  $\backslash$ hh 转义的 ASCII 码，例如：“ $\backslash$ x01 $\backslash$ x02 $\backslash$ x0a”，这类字符串主要是用于表示无法显示的 ASCII 码。

[0193] 其中，位置信息的输入可以是任意的进制。例如，例如  $\backslash$ d(ddd) 代表十进制数， $\backslash$ x(hhh) 代表 16 进制数。如果只有两位，可以不用“()”。例如  $\backslash$ d10 代表十进制数 10。

[0194] 下面举例说明字符在字符串中的位置信息。假设字符串为 abc，则第一个字符 a 在字符串中的位置信息可以设置为 0，第二个字符 b 在字符串中的位置信息设置为 1，第三个字符 c 在字符串中的位置信息设置为 2。换句话说，在一个字符串中，后一字符的位置信息等于前一字符的位置信息加 1，第一个字符的位置信息一般设置为 0。

[0195] 匹配单元 902，用于根据接收单元 501 接收的字符及位置信息在状态机中进行匹配，并根据匹配输出匹配结果；其中，状态机中各状态的跳转条件包括字符以及位置信息。

[0196] 如图 5 所示，该规则匹配装置可以包括：

[0197] 状态机生成单元 903，用于生成状态机。

[0198] 其中，匹配单元 902 具体用于根据接收单元 901 接收的字符及位置信息在状态机生成单元 903 生成的状态机中进行匹配，并根据匹配输出匹配结果；其中，状态机中各状态的跳转条件包括字符以及位置信息。

[0199] 请一并参阅图 10,图 10 为本实施例中提供的另一种规则匹配装置的结构示意图。如图 10 所示,在该规则匹配装置中,状态机生成单元 903 可以包括:

[0200] 解析子单元 9031,用于将输入规则进行语法解析,并改写成状态机构建所需的状态机规则;

[0201] 构建子单元 9032,用于根据上述的状态机规则进行状态机构建;

[0202] 存储子单元 9033,用于将构建的状态机转换成匹配引擎所需要的格式并存储。

[0203] 其中,解析子单元 9031 具体用于将输入规则进行语法解析,若输入规则是带有起始位置的纯字符串,则将输入规则改写成状态机构建所需的状态机规则“. {m} XXX”,其中,“. {m}”表示起始位置,“XXX”表示字符。

[0204] 其中,解析子单元 5031 具体将输入规则进行语法解析,若输入规则是以起始位置“. {m}”开始的,且包含“[]”,“()”,“|”中至少一个语法的正则表达式,则将正则表达式改写成与至少一个语法对应的若干个状态机规则;其中,每一个状态机规则以起始位置“. {m}”开始,且包含正则表达式的部分字符。

[0205] 其中,构建子单元 9033 具体用于根据状态机规则中的每个字符以及每个字符在状态机规则中的位置信息构建状态机;

[0206] 其中,在构建子单元 9033 构建的状态机中,起始状态至第二状态的跳转条件包括状态机规则中的首字符以及首字符的位置信息;状态机中的第二状态至第三状态的跳转条件包括状态机规则中的次字符以及次字符的位置信息;……;依此类推,状态机中的倒数第二个状态至最后一个状态的跳转条件包括状态机规则中的最后一个字符以及最后一个字符的位置信息。

[0207] 如图 10 所示,在该规则匹配装置中,匹配单元 902 可以包括:

[0208] 判断子单元 9021,用于判断接收单元 901 接收的字符与存储子单元 9033 存储的状态机的跳转条件中包括的字符是否相同,且接收单元 901 接收的位置信息与存储子单元 9033 存储的状态机的跳转条件中包括的位置信息是否相同,如果均相同,判断该跳转条件指向的状态是否为接收态,若是,则确定匹配到规则;若否,则确定匹配不到规则。

[0209] 本实施例中,状态机生成单元 903 将字符以及位置信息作为状态机中各状态的跳转条件,进而在匹配单元 902 接收到字符以及该字符在字符串中的位置信息后,可以在状态机中进行匹配并输出匹配结果。与现有技术相比,本发明实施例可以减少无效匹配结果的输出,提高规则匹配的效率。

[0210] 在具体实现过程中,上述各匹配单元可以使用各种硬件芯片并结合各种附属电路来实现,如通过 DSP、FPGA 等处理芯片并结合一些附属电路(电源电路、存储电路、接口电路等)实现,具体实现的方法为本领域技术人员所公知的技术,在此不再赘述。

[0211] 本领域普通技术人员可以理解:实现上述方法实施例的全部或部分步骤可以通过程序指令相关的硬件来完成,前述的程序可以存储于一计算机可读取存储介质中,该程序在执行时,执行包括上述方法实施例的步骤;而前述的存储介质包括:只读存储器(ROM)、随机存取器(RAM)、磁碟或者光盘等各种可以存储程序代码的介质。

[0212] 以上对本发明实施例中所提供的一种规则匹配方法及装置进行了详细介绍,本文中应用了具体个例对本发明的原理及实施方式进行了阐述,以上实施例的说明只是用于帮助理解本发明的方法及其核心思想;同时,对于本领域的一般技术人员,依据本发明的思

---

想,在具体实施方式及应用范围上均会有改变之处,综上,本说明书内容不应理解为对本发明的限制。

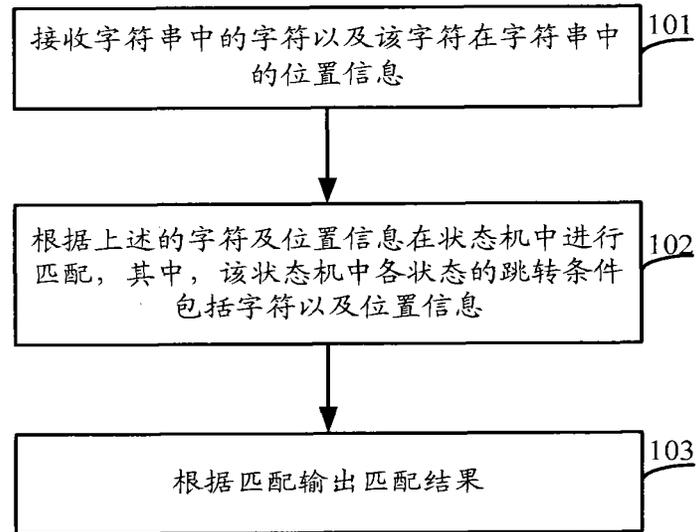


图 1

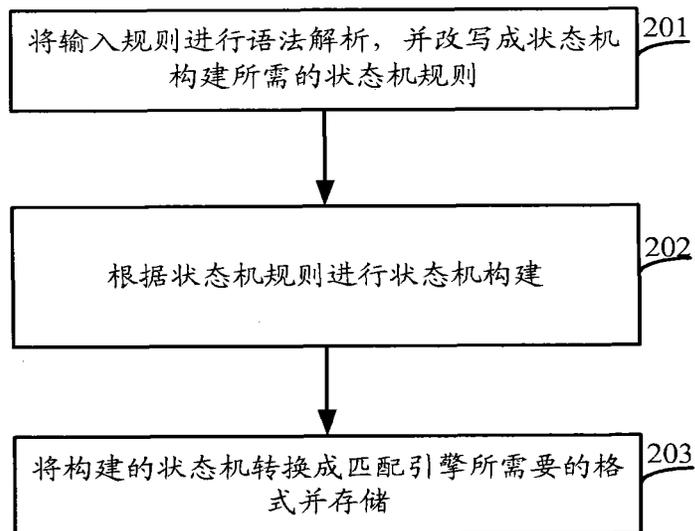


图 2

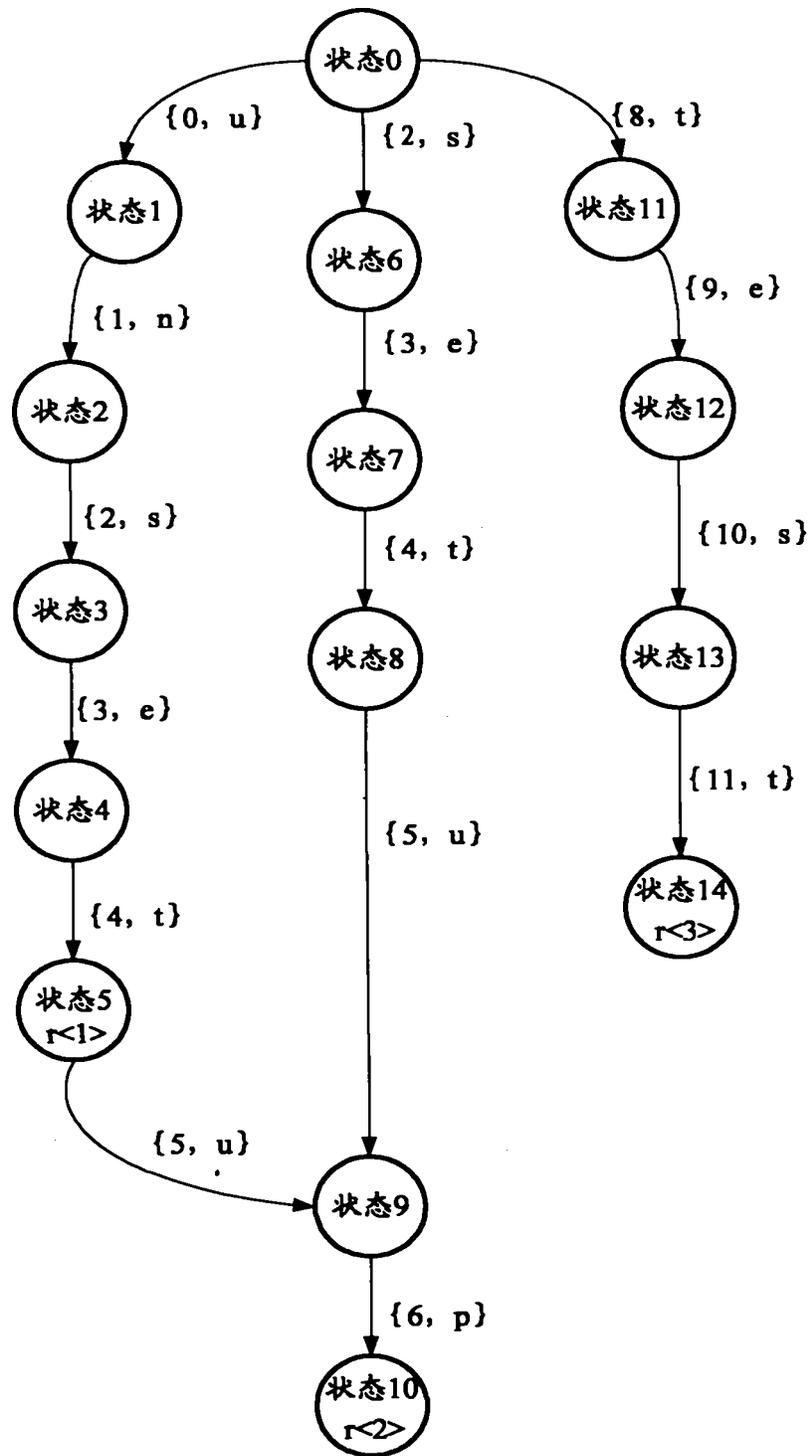


图 3

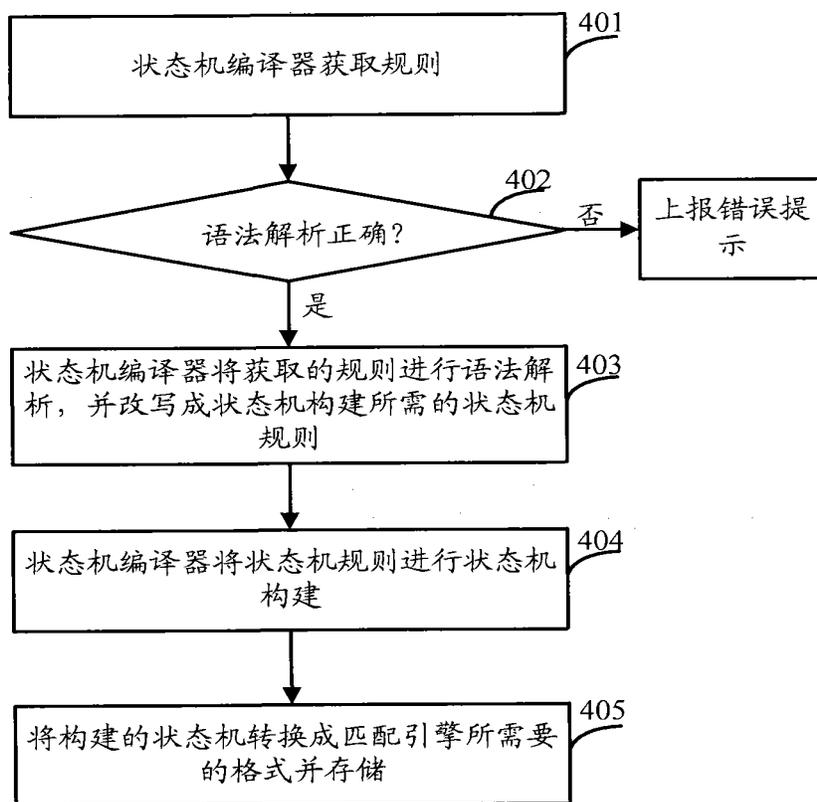


图 4

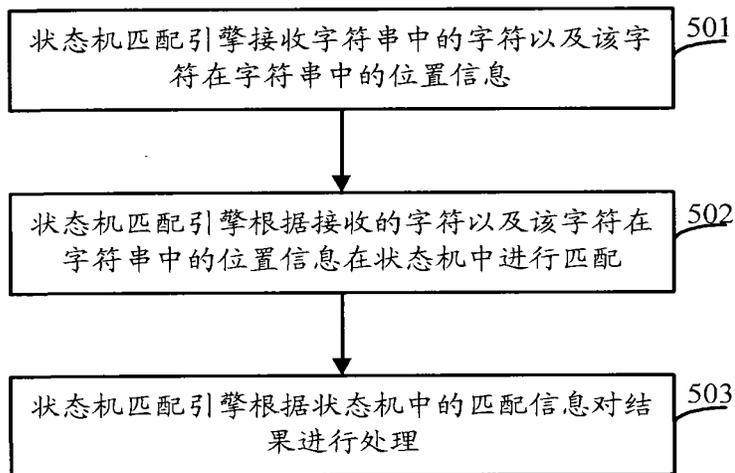


图 5

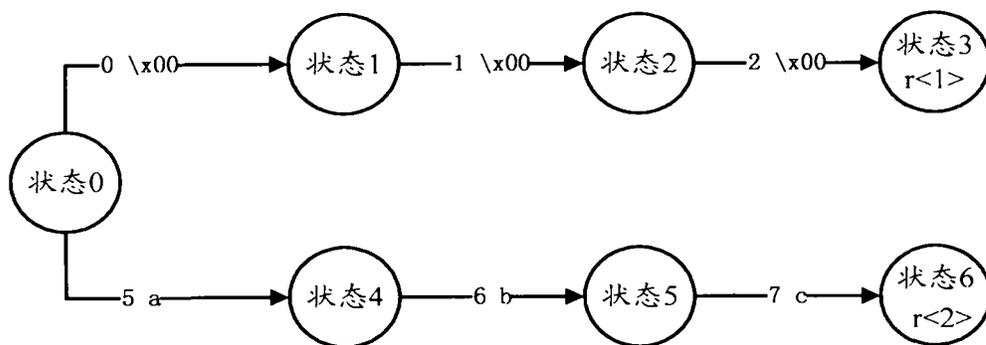


图 6

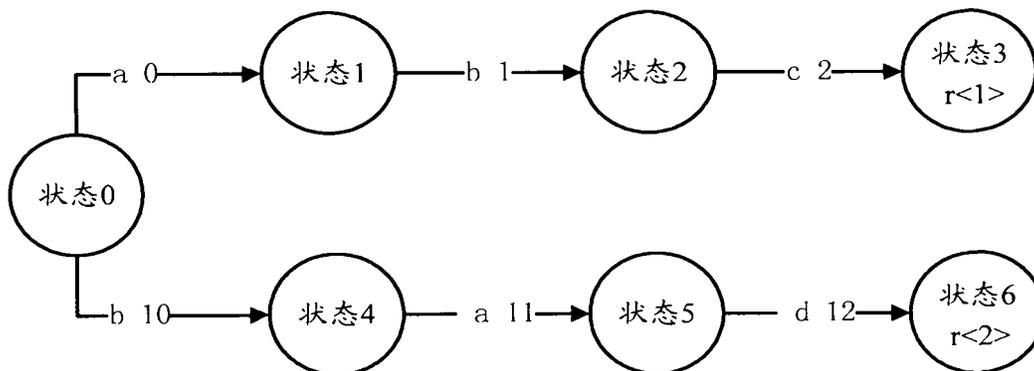


图 7

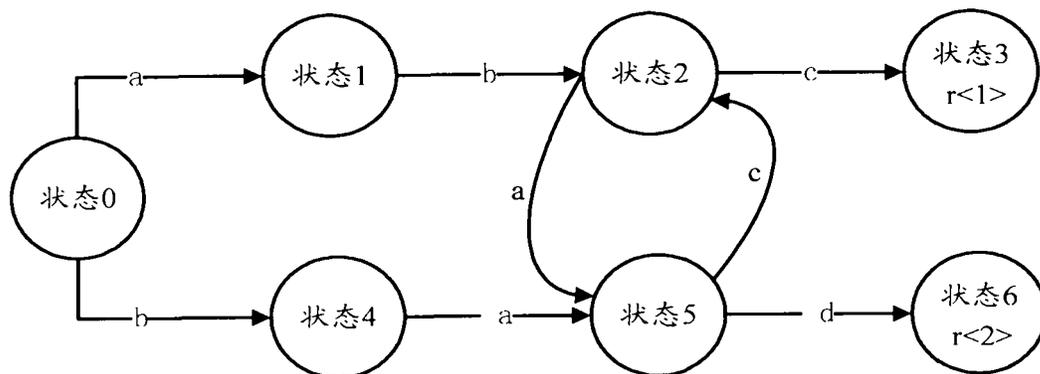


图 8

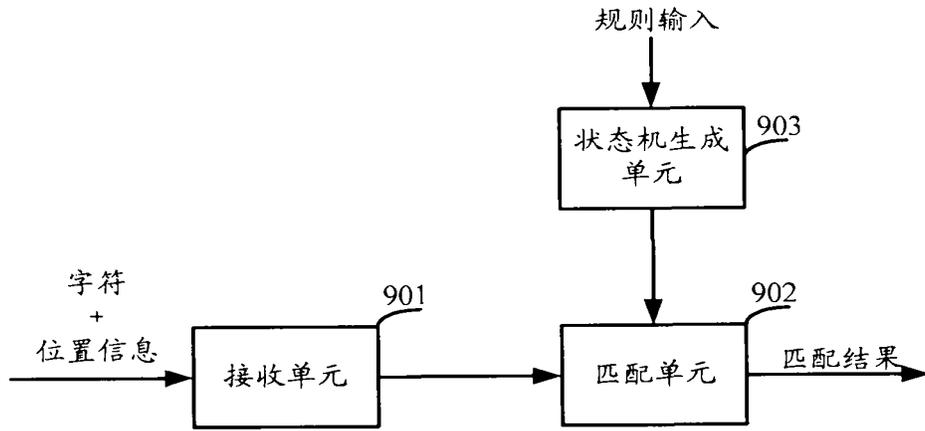


图 9

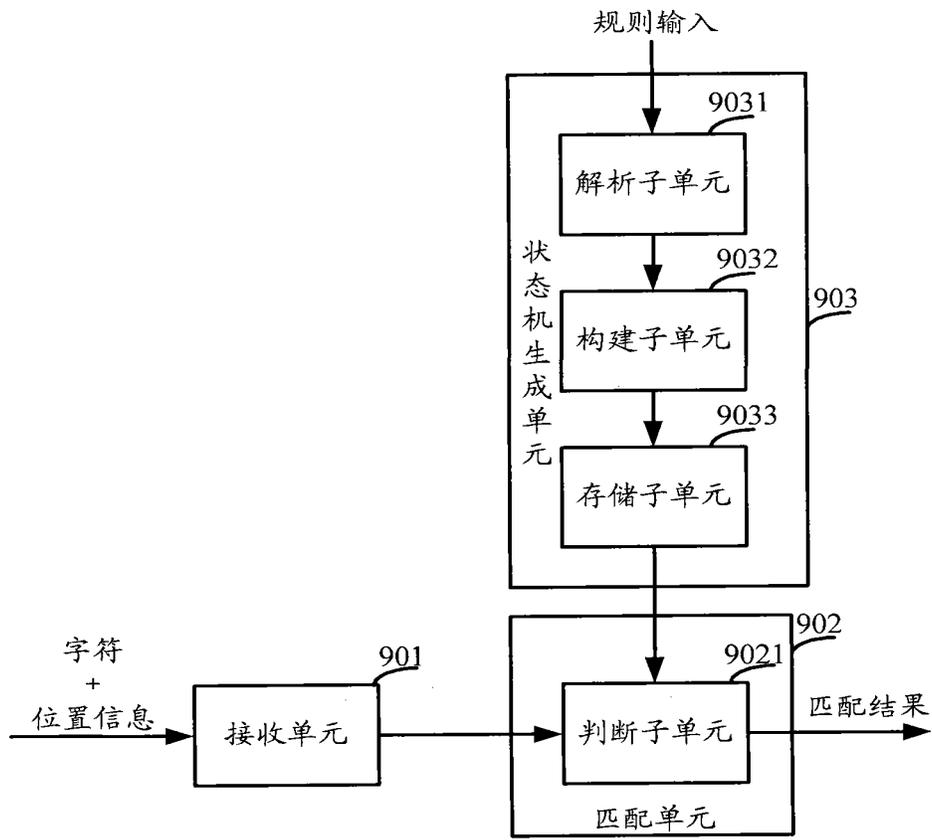


图 10