【软件构造】第六章第三节 面向可维护的构造技术

第六章第三节 面向可维护的构造技术

学了这么多OO设计模式，不外乎都是 delegation + subtying，万变不离其宗。

除了OO，还有什么其他能够提升软件可维护性的构造技术？——本节从委派+子类型跳出来，学习以下三个方面：
(1) 基于状态的构造技术 (2) 表驱动的构造技术 (3) 基于语法的构造技术

Outline

- 基于状态的构造技术
  - 状态模式（State Pattern）
  - 备忘录模式（Memento Pattern）
- 基于语法的构造技术
- 正则语法与正则表达式

Notes

基于状态的构造技术

【状态模式（**State Pattern**）】

【备忘录模式（**Memento Pattern**）】

基于语法的构造技术

【运用场景】

- 有一类应用，从外部读取文本数据， 在应用中做进一步处理。 具体来说，读取的一个字节或字符序列可能是：
- 输入文件有特定格式，程序需读取文件并从中抽取正确的内容。
- 从网络上传输过来的消息，遵循特定的协议。
- 用户在命令行输入的指令，遵顼特定的格式。
- 内存中存储的字符串，也有格式需要。

对于这些类型的序列，语法的概念是设计的一个好选择：

- 使用grammar判断字符串是否合法，并解析成程序里使用的数据结构 。
- 正则表达式
- 通常是递归的数据结构 。

【语法成分】

terminals 终止节点、叶节点

nonterminal 非终止节点（遵循特定规则，利用操作符、终止节点和其他非终止节点，构造新的字符串）

【语法中的操作符】

- 三个基本语法的操作符：
  - 连接，不是通过一个符号，而是一个空间：
    - x ::= y z //x等价于y后跟一个z
  - 重复，以*表示：
    - x ::= y* // x等价于0个或更多个y
  - 联合，也称为交替，如图所示 | ：
    - x ::= y | z //x等价于一个y或者一个z
- 三个基本操作符的组合：
  - 可选（0或1次出现），由？表示：
    - x ::= y? //x等价于一个y或者一个空串
  - 出现1次或多次：以+表示：
    - x ::= y+ //x等价于一个或者更多个y， 等价于 x ::= y y*
  - 字符类[...]，表示长度的字符类，包含方括号中列出的任何字符的1个字符串：
    - x ::= [abc] //等价于 x ::= 'a' | 'b' | 'c'
  - 否定的字符类[^...]，表示长度，包含未在括号中列出的任何字符的1个字符串：
    - x ::= [^abc] //等价于 x ::= 'd' | 'e' | 'f' | ... (all other characters in Unicode)
- 例子：
  - x ::= (y z | a b)* //an x is zero or more y z or a b pairs
  - m ::= a (b|c) d //an m is a, followed by either b or c, followed by d

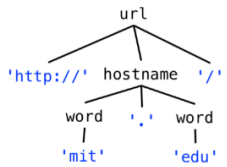
【实例：使用语法构造**URL**】

- 写一个语法表达式，表达如下的若干URL（域名）
  - http://google.com/
  - http://baidu.com/
  - http://stanford.edu/
- 我们可以用如下的一行表达式表示

```
url ::= 'http://' [a-z]+ '.' [a-z]+  '/'
```

- 用语法树可表示为如下形式：

```
url ::= 'http://' hostname '/'
hostname ::= word '.' word
word ::= [a-z]+
```



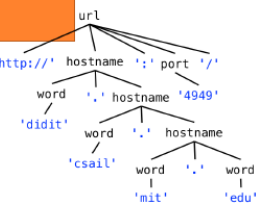
hostname可以有两个以上的部分，并且可以有一个可选的端口号：http://didit.csail.mit.edu:4949/

为了处理这样的字符串，语法可以这样写：

```
url ::= 'http://' hostname (':' port)? '/'
hostname ::= word '.' hostname | word '.' word
port ::= [0-9]+
word ::= [a-z]+
```

- hostname is now defined recursively in terms of itself.
- Another way:

```
hostname ::= (word '.')+ word
```



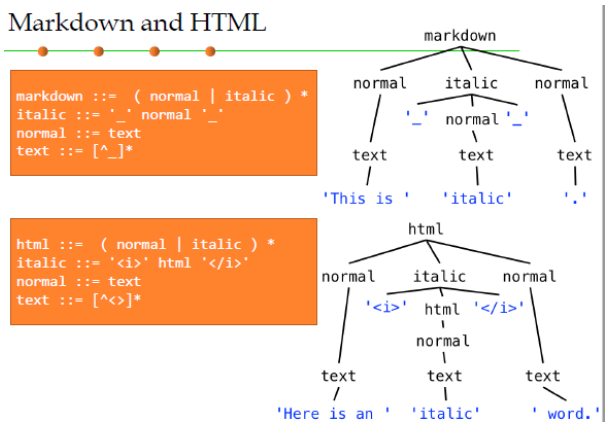
【Markdown 和 HTML的语法】

```
this is _italic_ example

a_b_c_d_e
```

this is *italic* example

abcde



正则语法与正则表达式

- 正则语法：简化之后可以表达为一个产生式而不包含任何非终止节点。
- 正则语法示例：

```
//Rugular!
url ::= 'http://' ([a-z]+ '.')+ [a-z]+ (':' [0-9]+)? '/'

//Regular!
markdown ::= ([^_]* | '_' [^_]* '_' )*

//Not Regular!
html ::= ( [^<>]* | '<i>' html '<i>' )*
```

.	any single character
\d	any digit, same as [0-9]
\s	any whitespace character, including space, tab, newline
\w	any word character, including letters and digits
\., \(, \), *, \+, ...	escapes an operator or special character so that it matches literally

- 正则表达式（regex）：去除引号和空格，从而表达更简洁(更难懂)
- 在Java中使用正则表达式
  - 适用场合：我们用正则表达式匹配字符串（例如 String.split , String.matches , java.util.regex.Pattern）
  - 用一个空格代替所有的多个空格

Replace all runs of spaces with a single space:

```
String singleSpacedString = string.replaceAll(" +", " ");
```

- 匹配一个URL：

```
Pattern regex =
      Pattern.compile("http://([a-z]+\\.)+[a-z]+(:[0-9]+)?/");
Matcher m = regex.matcher(string);
if (m.matches()) { // then string is a url   }
```

- 提取HTML标签的一部分

```
Pattern regex = Pattern.compile("<a href=\"([^\"]*)\">");
Matcher m = regex.matcher(string);
if (m.matches()) {
    String url = m.group(1);
    // Matcher.group(n) returns the nth parenthesized part of
    // the regex
}
```

附：正则语法

Character Classes

Construct	Description
[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z, or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z] (subtraction)

Predefined Character Classes

Construct	Description
.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Quantifiers

Greedy	Reluctant	Possessive	Meaning
X?	X??	X?+	X, once or not at all
X*	X*?	X*+	X, zero or more times
X+	X+?	X++	X, one or more times
X{n}	X{n}?	X{n}+	X, exactly n times
X{n,}	X{n,}?	X{n,}+	X, at least n times
X{n,m}	X{n,m}?	X{n,m}+	X, at least n but not more than m times

Boundary Matchers

Boundary Construct	Description
^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

【解释器模式（Interpreter Pattern）】