Markdown解析器设计与实现

# 定义标识符

因为“ ”空格表示的特殊性，这里将“ ”空格用单词 Space 代替

|  |  |  |
| --- | --- | --- |
| 类型 | 正则表达式定义 | 说明 |
| TEXT | [^\n\r]\* | 除换行符和回车符之外的任意字符的闭包都是TEXT |
| NEWLINE | \n | 换行符当作切分单词的分隔符 |
| ENTER | \r | 回车符当作切分单词的分隔符 |
| &amp; | & | HTML实体字符 |
| &lt; | < | HTML实体字符 |
| ESC | \[\`\*\_{}[]()#+-.!] | 转义字符起始标志 |
| HEAD标题组 | | |
| H1 | # | 标题一 |
| H2 | ## | 标题二 |
| H3 | ### | 标题三 |
| H4 | #### | 标题四 |
| H5 | ##### | 标题五 |
| H6 | ###### | 标题六 |
| DIVIDELINE分隔线组 | | |
| DIVIDELINE | -(Space-){2,}Space- | 分隔线 |
| DIVIDELINE | \\*(Space\\*){2,}Space\* | 分隔线 |
| QUOTE引用组 | | |
| QUOTE | (>Space)+ | 引用和嵌套引用 |
| LIST列表组 | | |
| ULIST | -Space | 无序列表 |
| ULIST | +Space | 无序列表 |
| ULIST | \*Space | 无序列表 |
| OLIST | [0-9].Space | 有序列表 |
| STYLE强调组 | | |
| EM | \*TEXT\* | 斜体 |
| EM | \_TEXT\_ | 斜体 |
| STRONG | \*\*TEXT\*\* | 加粗 |
| STRONG | \_\_TEXT\_\_ | 加粗 |
| CODE代码组 | | |
| INLINECODE | `TEXT` | 行级代码 |
| LAG | javascript | java | python | c++ | c | go | php | html | css | shell | 块级代码的语言标识 |
| BLOCKCODE | ```LAG \n\r[TEXT\n\r]+\n\r``` | 块级代码 |
| LINK组 | | |
| URL | /^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]\*)\*\/?$/ | 定义URL的正则表示 |
| LINK | <URL> | 可跳转URL |
| ALTLINK | [TEXT](URL) | 带文字说明的可跳转URL |
| IMG | ![TEXT](URL) | 引入图片资源 |
| LINKLEFT | < | LINK的左符号 |
| LINKRIGHT | > | LINK的右符号 |
| ALTLINKLEFT | [ | ALTLINK的左符号 |
| ALTLINKCENTER | ]( | ALTLINK的中符号 |
| ALTLINKRIGHT | ) | ALTLINK的右符号 |
| IMGBEGIN | ! | IMG的开始标识 |

# 定义状态机

为了方便表述，定义 (char)[[opt]]->state[(char)] 表示识别char经过opt操作达到状态state , opt默认为add “加入临时token序列”，这是为了后续识别的方便。除此之外，opt=emit in [type] “提交type类型到token流”；opt=transfer to [tag]“转换为tag 的html实体。其中[(char)]可以有，也可以没有，表示回流，比如(other)->state(other)表示识别other字符后回流字符到state状态重新识别。

|  |  |  |
| --- | --- | --- |
| 状态名称 | 状态入度 | 状态出度 |
| Start | 1. ()-> | 1. (#)->Head1 2. (>)->Quote 3. ([0-9])->Olist1 4. (+)->Ulist1 5. (\_)->Em 6. (\*)->EmUlistDivideline 7. (-)->UlistDivideline 8. (`)->Code1 9. (<)->LinkLeft 10. ([)->AltlinkLeft 11. (!)->ImgBegin 12. (\n)[emit]->Start 13. (\r)[emit]->Start 14. (&)[emit]->Start 15. (\)->Esc 16. (other)->Text |
| Head1 | 1. Start->(#) | 1. (#)->Head2 2. (other)[emit]->Start(other) |
| Head2 | 1. Head1->(#) | 1. (#)->Head3 2. (other)[emit]->Start(other) |
| Head3 | 1. Head2->(#) | 1. (#)->Head4 2. (other)[emit]->Start(other) |
| Head4 | 1. Head3->(#) | 1. (#)->Head5 2. (other)[emit]->Start(other) |
| Head5 | 1. Head4->(#) | 1. (#)[emit to H6]->Start 2. (other)[emit to H5]->Start(other) |
| Quote | 1. Start->(>) | 1. (Space)[emit]->Start 2. (other)[emit to TEXT]->Start(other) |
| Olist1 | 1. Start->([0-9]) | 1. (.)->Olist2 2. (other)[emit]->Start(other) |
| Olist2 | 1. Olist1->(.) | 1. (Space)[emit]->Start 2. (other)[emit]->Start(other) |
| Ulist1 | 1. Start->(+) | 1. (Space)[emit]->Start 2. (other)[emit]->Start(other) |
| Em | 1. Start->(\_) | 1. (\_)[emit to STRONG]->Start 2. (other)[emit to EM]->Start(other) |
| EmUlistDivideline | 1. Start->(\*) | 1. (\*)->EmUlistDivideline 2. (Space) -> EmUlistDivideline 3. (other)[emit by RegExp]->Start(other) |
| UlistDivideline | 1. Start->(-) | 1. (Space)-> UlistDivideline 2. (-)->UlistDivideline 3. (other)->[emit by RegExp]->Start(other) |
| Code1 | 1. Start->(`) | 1. (`)->Code2 2. (other)[emit to INLINECODE]->Start(other) |
| Code2 | 1. Code1->(`) | 1. (`)[emit to BLOCKCODE]->Start 2. (other)[emit] ->Start(other) |
| LinkLeft | 1. Start->(<) | 1. (TEXT)->TEXT 2. (other)[emit]->Start(other) |
| ImgBegin | 1. Start->(!) | 1. ([)->AltlinkLeft 2. (other)[emit]->Start(other) |
| AltlinkLeft | 1. Start->([) 2. ImgBegin->([) | 1. (TEXT)->Text 2. (other)[emit]->Start(other) |
| AltlinkCenter | 1. Text->(]) | 1. (()->Text 2. (other)[emit]->Start(other) |
| Esc | 1. Start->(\) | 1. ([\`\*\_{}[]()#+-.!])[emit]->Start 2. (other)[emit]->Start(other) |
| Text | 1. Start->(other) 2. Altlink->(TEXT) | 1. (TEXT)->Text 2. (>)[emit by RegExp]->Start 3. (])->AltLinkCenter 4. ())[emit by RegExp]->Start 5. (other)[emit]->Start(other) |

# 程序实现思路

程序分为两个部分：状态函数和主控程序

状态函数由当前状态根据字符输入到达的下一个状态的分支条件语句构成，比如：

let tokens = [];

let token = [];

state1 = function(char){

if(char == ‘\*){

token.push(char);

return state2;

}else{

emitToken(type , token.join(‘’));

token = [] ;

start(char);

}

}

function emitToken(type , token){

tokens.push({

type : type,

token : token

})

}

主控程序就是一个while循环，负责一个字符一个字符的向后读取原Markdown文章：

let state = start;

for(let char of input.split(‘’)){

state = state(char);

}