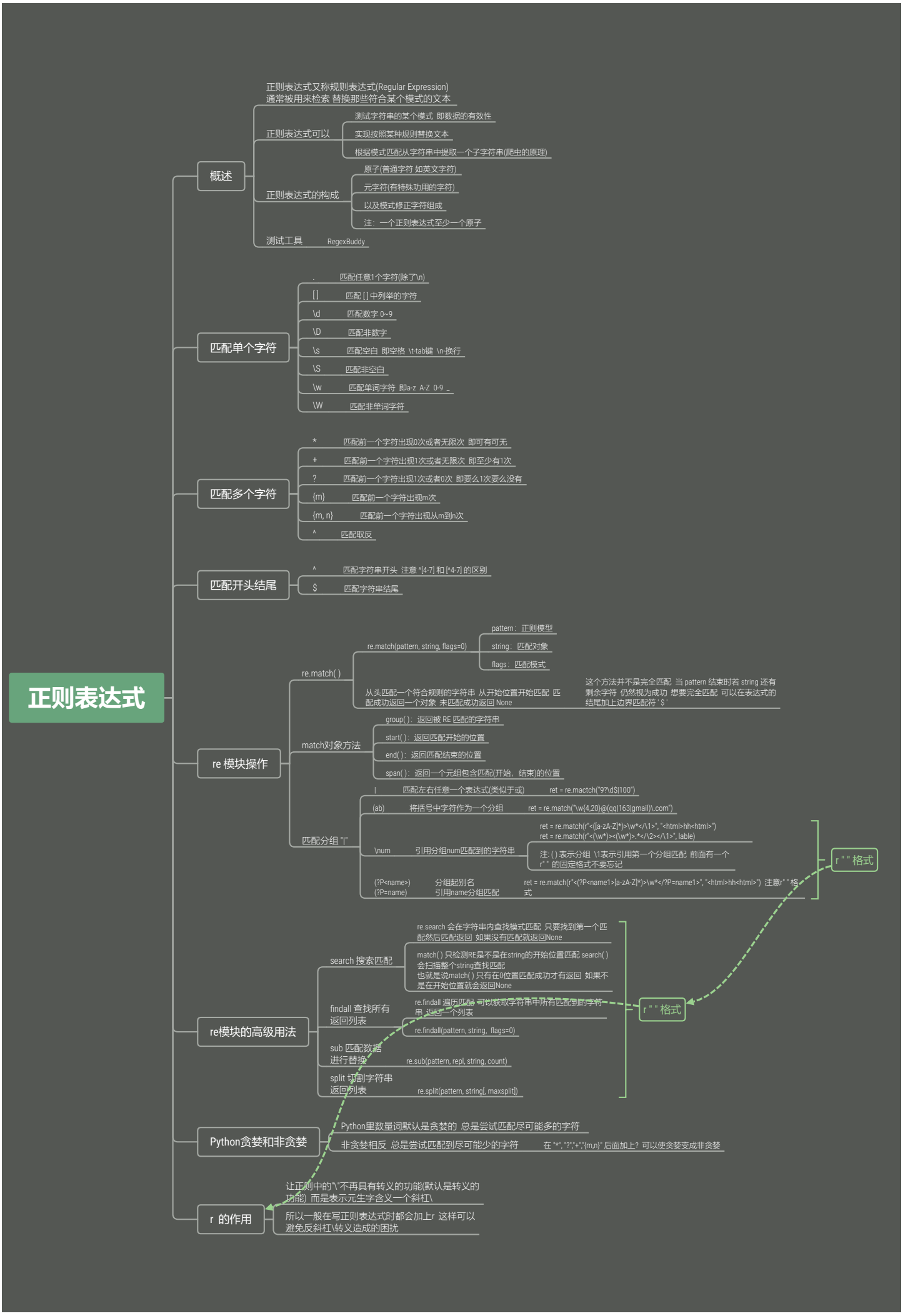
正则表达式
概述
正则表达式又称规则表达式(Regular Expression)
通常被用来检索 替换那些符合某个模式的文本
正则表达式可以
测试字符串的某个模式 即数据的有效性
实现按照某种规则替换文本
根据模式匹配从字符串中提取一个子字符串(爬虫的原理)
正则表达式的构成
原子(普通字符 如英文字符)
元字符(有特殊功用的字符)
以及模式修正字符组成
注：一个正则表达式至少一个原子
测试工具 RegexBuddy
匹配单个字符
. 匹配任意1个字符(除了\n)
[ ] 匹配 [ ] 中列举的字符
\d 匹配数字 0~9
\D 匹配非数字
\s 匹配空白 即空格 \t-tab键 \n-换行
\S 匹配非空白
\w 匹配单词字符 即a-z A-Z 0-9 _
\W 匹配非单词字符
匹配多个字符
* 匹配前一个字符出现0次或者无限次 即可有可无
+ 匹配前一个字符出现1次或者无限次 即至少有1次
? 匹配前一个字符出现1次或者0次 即要么1次要么没有
{m} 匹配前一个字符出现m次
{m, n} 匹配前一个字符出现从m到n次
^ 匹配取反
匹配开头结尾
^ 匹配字符串开头 注意 *[4-7] 和 [^4-7] 的区别
$ 匹配字符串结尾
re 模块操作
re.match( )
re.match(pattern, string, flags=0)
pattern：正则模型
string：匹配对象
flags：匹配模式
从头匹配一个符合规则的字符串 从开始位置开始匹配 匹配成功返回一个对象 未匹配成功返回 None
这个方法并不是完全匹配 当 pattern 结束时若 string 还有剩余字符 仍然视为成功 想要完全匹配 可以在表达式的结尾加上边界匹配符 '$'
match对象方法
group( )：返回被 RE 匹配的字符串
start( )：返回匹配开始的位置
end( )：返回匹配结束的位置
span( )：返回一个元组包含匹配(开始，结束)的位置
匹配分组 "|"
| 匹配左右任意一个表达式(类似于或) ret = re.mactch("9?\d$|100")
(ab) 将括号中字符作为一个分组 ret = re.match("\w{4,20}@(qq|163|gmail)\.com")
\num 引用分组num匹配到的字符串
ret = re.match(r"<([a-zA-Z]*)>\w*</\1>", "<html>hh<html>")
ret = re.match(r"<(\w*)><(\w*)>.*</\2></\1>", lable)
注: ( ) 表示分组 \1表示引用第一个分组匹配 前面有一个r" " 的固定格式不要忘记
(?P<name>) 分组起别名
(?P=name) 引用name分组匹配
ret = re.match(r"<(?P<name1>[a-zA-Z]*)>\w*</?P=name1>", "<html>hh<html>") 注意r" " 格式
r" " 格式
re模块的高级用法
search 搜索匹配
re.search 会在字符串内查找模式匹配 只要找到第一个匹配然后匹配返回 如果没有匹配就返回None
match( ) 只检测RE是不是在string的开始位置匹配 search( ) 会扫描整个string查找匹配
也就是说match( ) 只有在0位置匹配成功才有返回 如果不是在开始位置就会返回None
findall 查找所有返回列表
re.findall 遍历匹配，可以获取字符串中所有匹配到的字符串 返回一个列表
re.findall(pattern, string, flags=0)
sub 匹配数据进行替换
re.sub(pattern, repl, string, count)
split 切割字符串返回列表
re.split(pattern, string[, maxsplit])
r" " 格式
Python贪婪和非贪婪
Python里数量词默认是贪婪的 总是尝试匹配尽可能多的字符
非贪婪相反 总是尝试匹配到尽可能少的字符 在 "*", "?","+","{m,n}" 后面加上? 可以使贪婪变成非贪婪
r 的作用
让正则中的"\"不再具有转义的功能(默认是转义的功能) 而是表示元生字含义一个斜杠\
所以一般在写正则表达式时都会加上r 这样可以避免反斜杠\转义造成的困扰