# 一、解析源码

代码如下：

|  |
| --- |
| <?php  namespace srcfile\controller;  use app\lib\BaseController;  class Parse extends BaseController {  //  protected $model\_name = "\srcfile\model\File";  // D:\www\docs\helper\shell>d:\www\php-8.2.5-Win32-vs16-x64\php shell.php /api/srcfile/Parse  // 文件内容太大， longblob 类型 不能添加全文索引 ，只能用longtext  // 全文索引的性能还没测试过，是否真的快  function index(){  $where = "path not like '/ext/%' and path not like '%/tests/%' and ext in ('c','h','l','y')";  $sql = "select \* from file where {$where} order by rand() limit 30 ";  $sql = "select \* from file where file='zend\_optimizer.c'";  $files = $this->query($sql);  foreach($files as $v){  e ( "\n ------parseing file: ".$v['path'].'/'.$v['file'] );  // 定义的函数  $ref\_funcs = [];  if(empty($v['content'])) continue;  $funcs = $this->function\_name($v,$ref\_funcs);  $json = empty($v['json']) ? [] : json\_d($v['json']);  $json['funcs'] = $funcs;  $json['func\_count'] = count($funcs);  $json['ref\_funcs'] = $ref\_funcs;  $this->m()->where(['path'=>$v['path'],'file'=>$v['file']])  ->save(['json'=>json\_e($json)]);  }  echo 'over';  }    // 解析所有的函数定义（未确认是否有遗漏，多余的基本过滤了）  // 肯定还要解析不到的，先这样用  function function\_name(&$file,&$ref\_funcs=[]){  $ref\_funcs['\_\_total\_\_'] = [];  $file['content'] = explode("\n",$file['content']);  // 正则很简单，过滤准确率高主要因为源码够规范  $pre\_regex = "@^([a-zA-Z0-9\_\*]+ )+\\*\*([a-zA-Z0-9\_]+)\(@";  $pre\_regex2 = "@^(# ?define +)\\*\*([a-zA-Z0-9\_]+)\(@i";    // 二次过滤是个好方法，第一个正则尽量宽松，过滤出所有可疑选项  # 第二个尽量严格，然后把不符合条件的选项打印出来检查，很准确且高效  // 可以定义虚函数,空函数 强制要求结尾可以过滤大量错误，这里形参要求特别宽松  // 形参没有定义完整  $regex = "@^([a-zA-Z0-9\_\*]+ )+\\*\*([a-zA-Z0-9\_]+)\(.\*\)( | |\\*|\/\\*.\*|\{.\*|;.\*| \*\/\/.+)\*$@";  // 第一个参数是返回类型，第二个是函数名，不会出错  $regex2 = "@^([a-zA-Z0-9\_]+ )+\\*\*([a-zA-Z0-9\_]+)\(([0-9a-zA-Z\_ \*()\[\]\t]+,)\*( | |\\*|\/\\*.\*|\{.\*|;.\*| \*\/\/.+)\*$@";    // 被调用的函数  $ref\_regex = "@[^0-9a-zA-Z\_]\*\\*\*([a-zA-Z0-9\_]+)\(@";    // 逐行匹配  $valid = $invalid = [];  $current\_func = '';  foreach($file['content'] as $line\_no => $line){  // 遇到 } 时表示前一个函数终止  if(substr($line,0,1)=='}'){  $current\_func = '';  }  //  $declare = false;  // 有声明函数，开头可以是#define  if(  preg\_match($pre\_regex,$line,$re)  || preg\_match($pre\_regex2,$line,$re)  ){  // 这些不是返回类型，直接跳过，不计入invalid  $skip = ['return '];  if(in\_array($re[1],$skip)) continue;  $skip = ['if','elif'];  if(in\_array($re[2],$skip)) continue;  // e ($line);  // e ($re[1].': '.$re[2]);  // var\_dump($re);  // 声明函数有效  if(  preg\_match($regex,$line,$re2)  || preg\_match($regex2,$line,$re2)  || preg\_match($pre\_regex2,$line,$re2)  ){  $declare = true;  //e ('valid : '.$line);  // 函数开头，前一个函数可能没有 } 终止，也无所谓  $current\_func = $valid[] = $re2[2];  $ref\_funcs[$current\_func] = [];  }  else{  // 过滤出的无效函数  $invalid[] = [ $re[1], $re[2], $line ];  }  //echo "\n";  }  // 查找被调用的函数  if(!$declare){  //e($line);  if(preg\_match\_all($ref\_regex,$line,$re)){  foreach($re[0] as $k=>$v){  //e($line\_no.' : '.$v);  //e($current\_func.'->'.$re[1][$k]);  $func\_name = $re[1][$k];  if(isset($ref\_funcs[$current\_func][$func\_name])){  $ref\_funcs[$current\_func][$func\_name]++;  }  else{  $ref\_funcs[$current\_func][$func\_name]=1;  }  if(isset($ref\_funcs['\_\_total\_\_'][$func\_name])){  $ref\_funcs['\_\_total\_\_'][$func\_name]++;  }  else{  $ref\_funcs['\_\_total\_\_'][$func\_name]=1;  }  }  //e($re[1][0]);  }  }    }  if(isset($ref\_funcs[''])){  //var\_dump($ref\_funcs);  }  if(!empty($invalid)) {  //var\_dump($valid);  //var\_dump($invalid);  }  //die;  $valid = array\_unique($valid);  return $valid;  }    // d:\www\php-8.2.5-Win32-vs16-x64\php shell.php /api/srcfile/Parse/all  function all(){  $id = 0;  while(1){  // $where = "path not like '/ext/%' and path not like '%/tests/%' and ext in ('c','h','l','y') and id > {$id}";  $where = "ext in ('c','h','l','y') and id > {$id}";  $sql = "select \* from file where {$where} order by id asc limit 1";  $files = $this->query($sql);  if(empty($files)){  e('all over');  die;  }  foreach($files as $v){  $id = $v['id'];  e ( "\n{$id}: ----parseing file: ".$v['path'].'/'.$v['file'] );  // 定义的函数  $ref\_funcs = [];  if(empty($v['content'])) continue;  $funcs = $this->function\_name($v,$ref\_funcs);  //continue;  $json = empty($v['json']) ? [] : json\_d($v['json']);  $json['funcs'] = $funcs;  $json['func\_count'] = count($funcs);  $json['ref\_funcs'] = $ref\_funcs;  $json['ref\_funcs\_total'] = [];  unset($json['ref\_funcs2']);  foreach($json['ref\_funcs']['\_\_total\_\_'] as $k2=>$v2){  $json['ref\_funcs\_total'][] = [$k2,$v2];  }  $this->m()  ->where(['path'=>$v['path'],'file'=>$v['file']])  ->save(['json'=>json\_e($json)]);  }  }  }  // d:\www\php-8.2.5-Win32-vs16-x64\php shell.php /api/srcfile/Parse/one  function one(){  $id = 0;  while(1){  // $where = "path not like '/ext/%' and path not like '%/tests/%' and ext in ('c','h','l','y') and id > {$id}";  $where = "ext in ('c','h','l','y') and file='phpdbg.c' and id > {$id}";  $sql = "select \* from file where {$where} order by id asc limit 1";  $files = $this->query($sql);  //var\_dump($files);  if(empty($files)){  e('all over');  die;  }  //var\_dump(1);  foreach($files as $v){  $id = $v['id'];  e ( "\n{$id}: ----parseing file: ".$v['path'].'/'.$v['file'] );  // 定义的函数  $ref\_funcs = [];  if(empty($v['content'])) continue;  $funcs = $this->function\_name($v,$ref\_funcs);  //var\_dump($funcs,$ref\_funcs);die;  //continue;  $json = empty($v['json']) ? [] : json\_d($v['json']);  $json['funcs'] = $funcs;  $json['func\_count'] = count($funcs);  $json['ref\_funcs'] = $ref\_funcs;  $this->m()  ->where(['path'=>$v['path'],'file'=>$v['file']])  ->save(['json'=>json\_e($json)]);  }  }  }  } |

# 二、查询源码

代码如下：

|  |
| --- |
| <?php  namespace srcfile\controller;  use app\lib\BaseController;  class Query extends BaseController{  //  protected $model\_name = "\srcfile\model\File";  //  function index(){  $keyword = addslashes(input('keyword',''));  $file = addslashes(input('file',''));  $where = " ext in ('c','h','l','y') and path not like '%/tests/%' ";  if(!input('with\_ext','')){  $where .= " and path not like '/ext/%' ";  }  if(!empty($file)){  $where .= " and file='{$file}'";  }  //$where = " ext in ('c','h','l','y')";  $where .= " and match(content) against ('{$keyword}' )";  $sql = "select path,file,ext,content from file where {$where} limit 100 ";  $files = $this->query($sql);  $data = [];  foreach($files as $v){  $data = array\_merge($data,$this->filter($v,$keyword));  }  $this->display\_in\_iframe($data,$sql);  }    //  function filter(&$file,$keyword){  $data = [];  $kw = "<b style='color:red'>{$keyword}</b>";  //  $regex\_keyword = "@(^|[^0-9a-zA-Z\_])(".  str\_replace(['(',')'],['\(','\)'],$keyword)  .")($|[^0-9a-zA-Z\_])@";  $file['content'] = explode("\n\n",$file['content']);  //var\_dump($file['content']);  $flag = false;  foreach($file['content'] as $k=>$v){  if(preg\_match($regex\_keyword,$v,$re)){  $flag = true;  $data[] = [  'file' => "<a target='\_blank' href='/api/srcfile/query/desc?\_ajax=1&file={$file['file']}'>{$file['path']}/{$file['file']}</a>",  'line\_no' => $k+1,  'content' => preg\_replace($regex\_keyword,"$1{$kw}$3",$v)  ];  }  }  if(!$flag){  //var\_dump($regex\_keyword);  //var\_dump($file['content']);die;  //var\_dump(strstr($file['content'][537],'opcode'));  //var\_dump($file['content'][229],preg\_match($regex\_keyword,$file['content'][229],$re), $re);die;  $data[] = [  'file' => $file['path'].'/'.$file['file'],  'line\_no' => 0,  'content' => "<b style='color:blue'>match fail.</b>"  ];  }  return $data;  }  // iframe里显示结果  function display\_in\_iframe($r,$sql){  $table = "<table border=1>";  if($r){  $table .= "<tr>";  foreach($r[0] as $k=>$v){  $table.="<th>{$k}</th>";  }  $table .= "</tr>";  foreach($r as $k=>$v){  $table .= "<tr>";  foreach($v as $k2=>$v2){  $table.="<td>{$v2}</td>";  }  $table .= "</tr>";  }  }  else{  $time = date('Y-m-d H:i:s');  $table .= "<tr><td>&nbsp;执行完毕，无返回记录... [{$time}]&nbsp;</td></tr>";  }  $table.="</table>";  $html = <<<HTML  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <title>title</title>  <link rel="stylesheet" href="/plugins/fontawesome-free/css/all.min.css">  <!-- icheck bootstrap -->  <link rel="stylesheet" href="/plugins/icheck-bootstrap/icheck-bootstrap.min.css">  <!-- Theme style -->  <link rel="stylesheet" href="/dist/css/adminlte.min.css">  <script src="/plugins/jquery/jquery.min.js"></script>  </head>  <body style="padding-top:20px;">  <h5>sql: {$sql}</h5>  {$table}  </body>  </html>  HTML;  echo $html;  }  // 这个查法不会忽略没被调用的函数  function desc2(){  $file = trim(input('file',''));  $f = $this->m()->where('file',$file)->find();  if(empty($f)){  die("<h2>文件不存在：{$file}</h2>");  }  $json = json\_d($f['json']);  //var\_dump($json);  $data = [];  foreach($json['funcs'] as $k=>$v){  $sql = "select path,file from file where ext in ('c','h','l','y') and  match(content) against('{$v}') and id!='{$f['id']}' ";  if(!input('with\_ext','')){  //$sql .= " and path not like '/ext/%' ";  }  $fs = $this->query($sql);  $ref\_files = '';  foreach($fs as $k2=>$v2){  $ref\_files .= $v2['path']."/".$v2['file'].", ";  }    $data[] = [  'func' =>$v,  'ref\_files' => $ref\_files  ];  }  $this->display\_in\_iframe($data,"file='{$file}'");  }    // 这个查法会忽略没被调用的函数  function desc(){  $file = trim(input('file',''));  $sql = "select t2.func,count,file\_count,files from (  select func,sum(count) count,count(distinct(concat(path,'/',file))) file\_count,group\_concat(distinct(concat(path,'/',file)),'<br />') files from (  select path,file,t1.\* from php\_src.file f join json\_table(f.json,\"\$.ref\_funcs\_total[\*]\" columns( func varchar(255) path '$[0]',count int(5) path '\$[1]' ) ) t1  ) t group by func  ) t  right join (  select t1.func from php\_src.file f join json\_table(f.json,\"\$.funcs.\*\" columns( func varchar(255) path '\$' ) ) t1 where file='{$file}'  union all  select t1.func from php\_src.file f join json\_table(f.json,\"\$.funcs[\*]\" columns( func varchar(255) path '\$' ) ) t1 where file='{$file}'  ) t2 on t.func=t2.func  where t2.func is not null  order by file\_count desc,file\_count=1 and files like '%{$file}' asc,count desc";  $data = $this->query($sql);  $this->display\_in\_iframe($data,"file='{$file}'");  }  } |

# 三、编译器

代码如下：

|  |
| --- |
| <?php  // 在 zend\_language\_scanner.cpp 添加方法 int zendparse(){ /\* 业务逻辑加在这里 \*/ return 0;}  header("Content-Type: text/xml; charset=UTF-8");  echo "<wordbook>";  $p = new parser(\_\_DIR\_\_."\\test.php");  $p->show\_token();  //$p->show\_rules();  $p->start\_parse(0);  echo ("<wordbook>");  foreach($p->logic() as $v){  echo $v."\n";  }  echo ("</wordbook>");  echo "</wordbook>";  class parser{  // 文件  protected $file;  // 文件转成token  protected $token;  // 解析规则  protected $rules = [];  // 缓存  protected $cache = [];  // 序号  protected $index = 0;  // 业务逻辑  protected $logic = [];  // 堆栈  protected $stack = [];  //  protected $show\_msg = 0;    function logic(){  return $this->logic;  }    //  function \_\_construct($file){  // 文件名  $this->file = $file;  // 文件解析成token  $tmp = $this->file2token($file);  // 不要开头标签  array\_shift($tmp);  //  $this->token = $tmp;  // 分析规则  $this->make\_rules();  }    //  function show\_token(){  echo "<wordbook>";  foreach($this->token as $i=>$v){  echo "<word>";  echo $i," : ";  if(is\_array($v)){  echo $v[0],' -> ',$v[1],' -> ',$v[2],"\n";  }  else{  echo $v,"\n";  }  echo "</word>";  }  echo "</wordbook>";  }    // 开始解析  function start\_parse($show\_msg = 0){  $this->show\_msg = $show\_msg;  $this->print("<wordbook>");  $tmp = $this->parse('start',0,0,0);  $this->logic = $tmp[2];  $this->print("</wordbook>");  }      // 创建规则  function make\_rules(){  //  $rules = $this->rules();  // 不知是什么，要处理一下  $rules['backup\_fn\_flags']['rules'][0]['tokens'] = [];  $rules['backup\_fn\_flags']['empty']['logic'] = "{}";  //  $this->rules = $rules;  // var\_dump($rules); die;  }    function show\_rules(){  foreach($this->rules as $k=>$v){  echo $k,':',count($v['rules']),"\n";  //var\_dump($v);  }  }    // 业务逻辑，开始位置，token索引号  function add\_logic($index,$logic,$params,$in\_key){  $real\_params = $params;  foreach($real\_params as $k=>$v){  if(!stristr($v,'ast\_') && !stristr($v,'ZSTR\_INIT\_LITERAL')){  //var\_dump($v);  $real\_params[$k] = trim(strval($real\_params[$k]),"'\"");  $real\_params[$k] = '"'.$real\_params[$k].'"';  $real\_params[$k] = "zend\_ast\_create\_zval\_from\_str(ZSTR\_INIT\_LITERAL({$real\_params[$k]}, 0))";  //return $logic;  }  }  $logic = trim($logic,' {}');  if($in\_key=='backup\_doc\_comment'){  $name = "zend\_string \* ast\_".$index;  }  else{  $name = "zend\_ast \* ast\_".$index;  }  $name2 = "ast\_".$index;  foreach($real\_params as $k=> $v){  $k = $k+1;  $logic = str\_replace(['$'.$k,'$<num>'.$k],[$v,trim($v,'"')],$logic);  }  $logic = str\_replace('$$ ',$name.' ',$logic);  $logic = str\_replace('$$-',$name2.'-',$logic);  $logic .= "\t/\* ".$in\_key.':'.json\_encode($params)." \*/";  // var\_dump($logic);  return $logic ;  }    // 解析  // 递归可以，但要小心在原点递归，只要递归元素不在开头，就不会在原点递归，  // 只要不在原地递归，就不可能出现死循环，这才是重点！！！  function parse($in\_key,$num,$direct\_recursion=0,$prev\_index=0){  $index = $this->index++;  $this->print("<item\_{$index}>");  $this->print("<word>匹配开始：{$in\_key} </word>\n");    // 当前要处理的  $record = $this->rules[$in\_key];  $token = $this->token;  // 解析出的参数  $params = [];  //  $flag = false;  // logic  $logic = [];  // 有行号参数  $line\_no = 0;    // 如果到头了  if(empty($this->token[$num])){  if(isset($record['empty'])){  $this->print("{$index}:无内容,empty</item\_{$index}>");  $logic[] = $this->add\_logic($index,$record['empty']['logic'],[],$in\_key);  return [$num,$index,$logic];  }  else{  $this->print("{$index}:无内容</item\_{$index}>");  return [false,$index,$logic];  }  }  // 当前解析进入堆栈,2步  $current\_stack = [$in\_key,$num,$direct\_recursion];  $this->stack[] = $current\_stack;  // 初始为失败  $success = false;  // 失败原因  $fail\_type = "";    // 遍历所有规则  foreach($record['rules'] as $i=>$rule){  // var\_dump($in\_key,$rule);//continue;  // 开始位置  $n = $num;  // 复位，业务逻辑，参数  $tmp\_logic = [];  $params = [];  // 默认为匹配失败  $flag = false;  // 遍历所有关键字  foreach($rule['tokens'] as $k3 => $key){  // 如果在开头递归中，并且是第一个元素  if($direct\_recursion && $k3==0){  // 是递归规则  if($key==$in\_key){  // 跳过第一个，从第二个开始  // var\_dump("处理 开头递归".$key);  // 添加参数  $params[$k3+$line\_no] = 'ast\_'.$prev\_index;  continue;  }  // 开头没有递归规则的  else{  //$fail\_type = '跳过非递归';  break;  }  }  // 无开头递归，中间的递归当普通的处理？  else{  // 不允许间接递归  $step = [$key,$n,0];  $time = 0;  foreach($this->stack as $v){  if($step==$v){  $time++;  }  }  if($time>1){  $fail\_type = '间接递归';  $this->fail\_msg($fail\_type,$in\_key,$rule['tokens'],$key,$num,$n);  //break;  }  // 前置字符  elseif($key=='%prec' || $key=='PREC\_ARROW\_FUNCTION'){  // var\_dump("发现$key", $rule['tokens']);  continue;  }  // 条件是小写  elseif(preg\_match("@[a-z]@",$key)){  // var\_dump("小写条件:".$key);  $cache\_key = $key.'\_'.$n;  // 缓存可减少98%的无用功  if(isset($this->cache[$cache\_key])){  //var\_dump('cache');  $tmp2 = $this->cache[$cache\_key];  }  else{  $tmp2 = $this->parse($key,$n,0,$index) ;  }  $tmp = $tmp2[0];    // 失败  if($tmp===false){  // 参数传入  if($k3!=($direct\_recursion?1:0)){  $fail\_type = '小写条件';  $this->fail\_msg($fail\_type,$in\_key,$rule['tokens'],$key,$num,$n);  }  break;  }  // 成功情况1：  else{  // 不是false算成功，小写匹配成功，要接着匹配后面的  $n=$tmp;  // 尝试添加参数  $params[$k3+$line\_no] = 'ast\_'.$tmp2[1];  // 如果是最后一个元素，成功  if(count($rule['tokens'])-1==$k3){  // var\_dump('bull:'.$key);  $flag = true;  }  // 合并返回的业务逻辑  $tmp\_logic = array\_merge($tmp\_logic,$tmp2[2]);  // 尝试添加业务逻辑  if(isset($rule['logic'][$k3])){  if(strstr($rule['logic'][$k3],'zend\_lineno')){  $line\_no=1;  }  else{  $tmp\_logic[] = $this->add\_logic($index,$rule['logic'][$k3],$params,$in\_key);  }  }  }  }  // 条件不是小写:是大写或字符  else{  $real\_true = isset($token[$n]) && (is\_array($token[$n]) ? $token[$n][0]==$key : $token[$n]==$key) ;    // 特殊情况 T\_NOELSE  $special\_true = false;  if($key=='T\_NOELSE'){  $special\_true = (empty($token[$n]) || !is\_array($token[$n]) || $token[$n][0]!='T\_ELSE') ;  // var\_dump($token[$n],$special\_true, $rule['tokens'],$k3);  }    // 只有这里是真正的比较成功，需要++  if($real\_true || $special\_true){  //echo "匹配token | ".$in\_key." | ".implode(" ",$rule['tokens'])  // ." | ".$this->token\_str($n,$n,1)." \n";  // 成功情况2：  if(count($rule['tokens'])-1==$k3){  $flag = true;  }  // 尝试添加参数,T\_NOELSE要特殊处理  if(!$special\_true){  $params[$k3+$line\_no] = is\_array($this->token[$n])?$this->token[$n][1]:$this->token[$n];  }  // 尝试添加业务逻辑  if(isset($rule['logic'][$k3])){  if(strstr($rule['logic'][$k3],'zend\_lineno')){  $line\_no=1;  }  else{  $tmp\_logic[] = $this->add\_logic($index,$rule['logic'][$k3],$params,$in\_key);  }  }  // 前进  if($real\_true)  $n++;    }  // 失败  else{  if($k3!=($direct\_recursion?1:0)){  $fail\_type = '大写和符号';  $this->fail\_msg($fail\_type,$in\_key,$rule['tokens'],$key,$num,$n);  }  break;  }  // var\_dump($key);die;  }  }  }  // 成功，还是要匹配下一个，为了找到最长的  if($flag===true){  // 匹配成功，要返回序号和业务逻辑  if($success<$n){  $success = $n;  $logic = $tmp\_logic;  }  $this->print("<word>成功 @ {$in\_key} -> ".$this->escape(implode(" ",$rule['tokens']))  ." -> ".$this->token\_str($num,$success>$num?$success-1:$num,1)." </word>\n");  }  }    //if($in\_key=='expr') var\_dump($this->stack);    // 一个规则也没匹配上但有empty  if(!$direct\_recursion && $success===false && !empty($record['empty'])){  $this->print("<word>应用empty: {$in\_key}</word>");  // 添加业务逻辑  $logic = [$this->add\_logic($index,$record['empty']['logic'],[],$in\_key)];  // 查询位置不变但返回成功  $success = $n = $num;  }    // 有递归规则，并且处理成功  if(!empty($record['head\_recursive']) && $success!==false){  // 开头递归  $this->print("<word>处理开头递归:{$in\_key},n:{$success}</word>");  $tmp = $this->parse($in\_key,$success,1,$index);  if($tmp[0]!==false){  if($tmp[0]>$success){  $success = $n = $tmp[0];  // 递归成功需要更新index  $index = $tmp[1];  // 合并业务逻辑  $logic = array\_merge($logic,$tmp[2]);  }  }  }    // 非直接递归，可以缓存，直接递归可能变化  if(!$direct\_recursion){  $catch\_key2 = $in\_key.'\_'.$num;  $this->cache[$catch\_key2] = [$success,$index,$logic];  }    // 退堆栈  array\_pop($this->stack);    $this->print("<word>匹配结束</word></item\_{$index}>");  // 返回  return [$success,$index,$logic];  }    // 解析出错信息  function fail\_msg($fail\_type,$in\_key,$tokens,$key,$num,$n){  $min\_n = isset($\_GET['n']) ? intval($\_GET['n']) : 99999999999 ;  if($n >= $min\_n && $fail\_type){  $token = $this->escape(implode(" ",$tokens));  $key = $this->escape($key);  $this->print("<word>{$fail\_type}\_失败 @ {$in\_key} -＞ ".$token  ." -＞ {$key} -＞ ".$this->token\_str($num,$n>$num?$n-1:$num,1)." </word>\n");  }  }    // 打印消息  function print($str){  $str = str\_replace(['&'],['＆'],$str);  if($this->show\_msg)  echo $str;  }    // 转义  function escape($str){  return str\_replace(['<','>','&'],['＜','＞','＆'],$str);  }    // 打印一串token  function token\_str($start,$end,$index=0){  $token = $this->token;  $str = "";  for($i=$start;$i<=$end;$i++){  if(is\_array($token[$i])){  $str.=" ".$token[$i][$index];  }  else{  $str.=" ".$token[$i];  }  }  $str .= " 序号：".$start.'-'.$end;  $str = $this->escape($str);  return $str;  }    // 文件解析成token  function file2token($path){  $c = file\_get\_contents($path);  $t = token\_get\_all($c);  $re = [];  foreach($t as &$v){  //var\_dump($v);  if(is\_array($v)){  $v[0] = token\_name($v[0]);  // tokenizer\_data.c 中转换  if($v[0]=='T\_DOUBLE\_COLON'){  $v[0] = 'T\_PAAMAYIM\_NEKUDOTAYIM';  }  if(!in\_array($v[0],['T\_WHITESPACE','T\_DOC\_COMMENT','T\_COMMENT'])){  //var\_dump( $v[1] .'=>'. $v[0] );  $re[] = $v;  }    }  else{  $re[] = $v;  }  }  return $re;  }    // 规则解析成数组v2.0  function rules(){  // 未修改过的源文件  $rules = 'D:\www\php-8.2.5-src\Zend\zend\_language\_parser.y';  $r = file\_get\_contents($rules);  $r = preg\_replace("@\\$\\<num\\>[0-9]@is","NULL",$r);  // 防解析出错  $r = str\_replace('{closure}','<closure>',$r);  $r = str\_replace('CG(doc\_comment); CG(doc\_comment) =','',$r);  $r = str\_replace("%prec '~'",'',$r);  $r = str\_replace("(void) zendnerrs;",'',$r);  //$r = str\_replace('$$ = CG(','CG(',$r);  $r = str\_replace('ZEND\_ACC\_PUBLIC','zend\_ast\_create\_zval\_from\_str(ZSTR\_INIT\_LITERAL("public", 0))',$r);  $r = str\_replace('ZEND\_ACC\_PROTECTED','zend\_ast\_create\_zval\_from\_str(ZSTR\_INIT\_LITERAL("protected", 0))',$r);  $r = str\_replace('ZEND\_ACC\_PRIVATE','zend\_ast\_create\_zval\_from\_str(ZSTR\_INIT\_LITERAL("private", 0))',$r);  $r = str\_replace('ZEND\_ACC\_RETURN\_REFERENCE','(1 << 12)',$r);      // 取出规则部分  $r = explode("%%",$r);  $r = $r[1];  // 删除注释  $regex = "@\/\\*[^/]+\\*\/@is";  $r = preg\_replace($regex,"",$r);  // 拆分规则  $regex = "@[a-z\_]+:@is";  // 规则名  preg\_match\_all($regex,$r,$d);  // 规则内容  $r = preg\_split($regex,$r);  // 遍历所有规则  $data = [];  //  foreach($d[0] as $i=>$v){  $v = substr($v,0,-1);  // 原规则名建立数组  $data[$v] = ['rules'=>[]];  // 原规则  //$data[$v]['text'] =  $tmp\_rule = $r[$i+1];  // 拆分子规则, 先去除结尾分号, ':单个字符 c:closure  $tmp\_rule = trim(trim($tmp\_rule),';');  // 2次才能消干净  $tmp\_rule = trim(trim($tmp\_rule),';');  $tmp\_rule = preg\_split("@(\{[^'][^\{\}]\*(?:\{[^'][^\{\}]\*(?:\{[^'][^\{\}]\*[^']\})?[^\{\}]\*[^']\})?[^\{\}]\*[^']\})@is",$tmp\_rule,0,3);  // 分割结果有可以是偶数（最后一个刚刚好），最后一个有可能不是空格（只有一个元素）  // 验证分割没有问题  foreach($tmp\_rule as $i3=>$v3){  $logic = preg\_match("@\{\s|\s\}@",$v3);  // 单数必须有，双数必须没有,过了这里说明逻辑没问题了  if(($i3%2==0)===$logic){  var\_dump($tmp\_rule,\_\_LINE\_\_);die;  }    }  // 整理成规则数组  for($i2=0,$l=count($tmp\_rule);$i2<$l;$i2+=2){  // 跳过，空的  if(!trim($tmp\_rule[$i2])) continue;    $rs = preg\_split("@\s\*\|\s+@isu",trim(trim(trim($tmp\_rule[$i2]),'|')));    foreach($rs as $v3){  if($v3=='%empty'){  $data[$v]['empty'] = [  // 没有处理逻辑，单纯的关键字表  'logic' => $tmp\_rule[$i2+1]??""  ];  }  else  {  $v3 = preg\_split("@\s+@is",$v3);  // 去掉引号  foreach($v3 as $k4=>$v4){  $v3[$k4] = trim($v4,"'");  }  $tmp = [  // 后面还是要有个空，否则 |' 会出错  'tokens' => $v3,  // 有些没有处理逻辑，单纯的关键字表  'logic' =>[ count($v3)-1 => $tmp\_rule[$i2+1]??"" ],  // 与前面逻辑分割开  'concat' => strstr($tmp\_rule[$i2],'|')===false && $i2!=0  ];  // 是否有开头递归  if($tmp['tokens'][0]==$v && !$tmp['concat']){  $data[$v]['head\_recursive'] = 1;  }  // 连接的排序值，不在开头  if($tmp['concat'] && count($data[$v]['rules'])>0 ){  $t = array\_pop($data[$v]['rules']);  $t['tokens'] = array\_merge($t['tokens'],$tmp['tokens']);  $t['logic'][count($t['tokens'])-1] =  $tmp['logic'][count($tmp['tokens'])-1];  $t['concat'] = true;  $data[$v]['rules'][] = $t;  }  // 不连接  else{  $data[$v]['rules'][] = $tmp;  }  }  }    }  }  // die();  return $data;  }  } |

# Change Log

2025.6.28 整理更新。