链滴

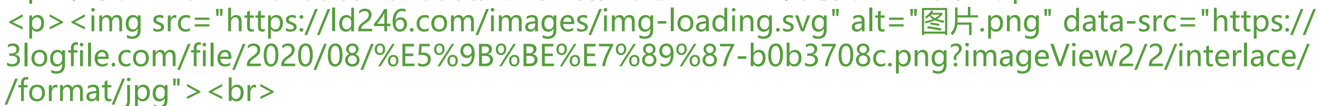
Abstract Syntax Code

作者：someone27889

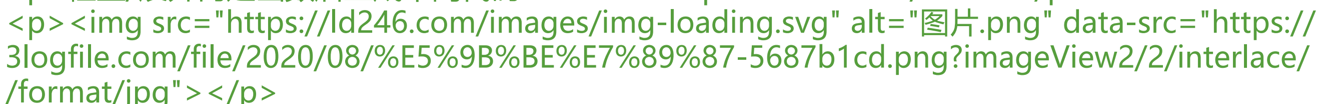
原文链接：https://ld246.com/article/1598364402210

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

<h2 id="抽象语法树-AST">抽象语法树 AST</h2>
<p>将代码进行词法分析语法分析后变成树结构以便于生成中间代码的环节</p>

<p><img src="https://ld246.com/images/img-loading.svg" alt="图片.png" data-src="https://
3logfile.com/file/2020/08/%E5%9B%BE%E7%89%87-b0b3708c.png?imageView2/2/interlace/
/format/jpg">
如图所示，该树为 <code>2*3+7</code></p>
<h2 id="词法分析">词法分析</h2>
<p>比如一段代码</p>
<pre><code class="highlight-chroma"><span class="highlight
cl">const a = 5;
</code></pre>
<p>经过词法分析</p>
<pre><code class="highlight-chroma"><span class="highlight
cl">[{value: 'const', type: 'keyword'}, {value: 'a', type: 'identifier'}, ...]
</code></pre>
<h2 id="语法分析">语法分析</h2>
<p>词法分析的输出是语法分析的输入，将词法分析的结果转化为树</p>
<pre><code class="highlight-chroma"><span class="highlight
cl">{
 type: "VariableD
clarator",
 id: {
 type: "Identifi
r",
 name: "a"
 },
 ...
}
</code></pre>
<p>当生成树的时候，解析器会删除一些没必要的标识 tokens（比如：不完整的括号），因此 AST
是 100% 与源码匹配的。</p>
<p>与源代码互相匹配的叫做 <code>具体语法树</code></p>
<h2 id="关于研究用途拆解包">关于研究用途拆解包</h2>
<p>recast 包可以在 nodejs 中拆解 code 输出词法语法分析后的结果，并且可以进行 inspect 操作
等</p>
<h2 id="拿到抽象语法树之后">拿到抽象语法树之后</h2>
<p>Go 语言的编译器会对语法树中定义和使用的类型进行检查</p>

常量、类型和函数名及类型；
变量的赋值和初始化；
函数和闭包的主体；
哈希键值对的类型；
导入函数体；
外部的声明；

<p>检查过程中展开内建函数</p>
<p>比如 go 中的 make() 转为具体的 make 类型 <code>makeslice makechan</code> 等</p>
<h2 id="中间代码生成">中间代码生成</h2>
<p>检查/展开内建函数后生成中间代码 <code>compileFunctions</code></p>

<p><img src="https://ld246.com/images/img-loading.svg" alt="图片.png" data-src="https://
3logfile.com/file/2020/08/%E5%9B%BE%E7%89%87-5687b1cd.png?imageView2/2/interlace/
/format/jpg"></p>
<p>将函数 放入编译队列 Queue 等待编译 后端 携程消费</p>
<h2 id="编译">编译</h2>

<p>1.编译入口在 [<code>src/cmd/compile/internal/gc/main.go</code>]</p>
<p>2.先获取命令行传入的参数并更新编译选项和配置</p>
<p>3.运行 <code>parseFiles</code> 函数对输入的所有文件进行词法与语法分析得到文件对应的象语法树</p>

检查常量、类型和函数的类型；
处理变量的赋值；
对函数的主体进行类型检查；
决定如何捕获变量；
检查内联函数的类型；
进行逃逸分析；
将闭包的主体转换成引用的捕获变量；
编译顶层函数；
检查外部依赖的声明；

<p>最后生成中间代码 main 最后根据 机器类型生成对应的机器码</p>
<h2 id="感谢">感谢</h2>
<p>内容部分来自:
draveness</p>