

# nhwc compiler

---

nhwc compiler 的目标是成为一个简化版的类llvm的编译器框架，用于生成 nhwc ir、riscv assembly。这个项目专注于代码可读性，因此使用了大量贴合英语语法的宏和 graphviz 的可视化功能生成图样来更好的debug

## nhwc 是什么意思？

---

nhwc 的意思是 Never Happy When Coding，出自动画「BanG Dream! It's Mygo!!!」中台词:"我从来没有觉得玩乐队开心过。"

## 进度

- ☒ 基础设施
  - ☒ 符号表
    - ☒ 支持快速查找符号(使用平衡树套哈希表)
    - ☒ 支持对符号动态添加field信息
  - ☒ 图
    - ☒ 定义简洁的宏来完成访问和插入节点操作
  - ☒ nhwc ir
    - ☒ 支持 phi-node
    - ☒ 支持简单的的 llvm ir 功能
    - ☐ 支持 对数组访问和修改命令
  - ☒ riscv instr
    - ☒ 支持常见的32位操作
  - ☒ Pass形式组织的框架
- ☐ 编译总流程
  - ☒ code2ast\_pass 由c语言代码构造ast树
    - ☒ antlr 自带的图结构转 petgraph 图结构
  - ☒ ast2st\_pass 由 ast 树构造 scope tree 解析作用域
    - ☒ 支持 ast node 与 st node 互相映射
  - ☐ ast2et\_debug\_pass 这个 Pass 仅用于绘制 expression tree 图片，测试toolkit::gen\_et
    - ☒ 支持数组相关 Operator
    - ☒ 支持对节点添加 def 或 use 信息
    - ☒ 支持简单的常量子树计算优化
    - ☒ 支持解析单个 statement
    - ☐ 支持直接解析整个basic block
  - ☒ ast2cfg\_pass 用于将 ast 树转化为 cfg (不带nhwc ir)
  - ☐ cfg2ncfg\_pass 用于从 cfg(不带nhwc ir) 构造 ncfg(带nhwc ir 的cfg)
    - ☒ 符号作用域检查
    - ☒ 支持遍历作为 树的 et
    - ☐ 支持遍历作为 DAG的 et
  - ☐ ncfg2ssa\_ncfg 将 ncfg(non-ssa) 转化为 ssa形式的ncfg
    - ☐ 插入 phi-node
    - ☐ 变量重命名

- ☐ nhwc2riscv\_pass 由nhwc ir 转化为 riscv
  - ☐ 完成nhwc ir 到riscv assembly 的简单对应
- ☐ riscv2binary\_pass 将riscv 汇编文件编译为二进制文件
- ☐ 高级优化
  - ☐ 支持循环展开
  - ☐ 支持对表达式进行公共子表达式消除
  - ☐ 支持向量化
  - ☐ 支持常量传播

更多内容，敬请期待

## Pass 形式组织的框架

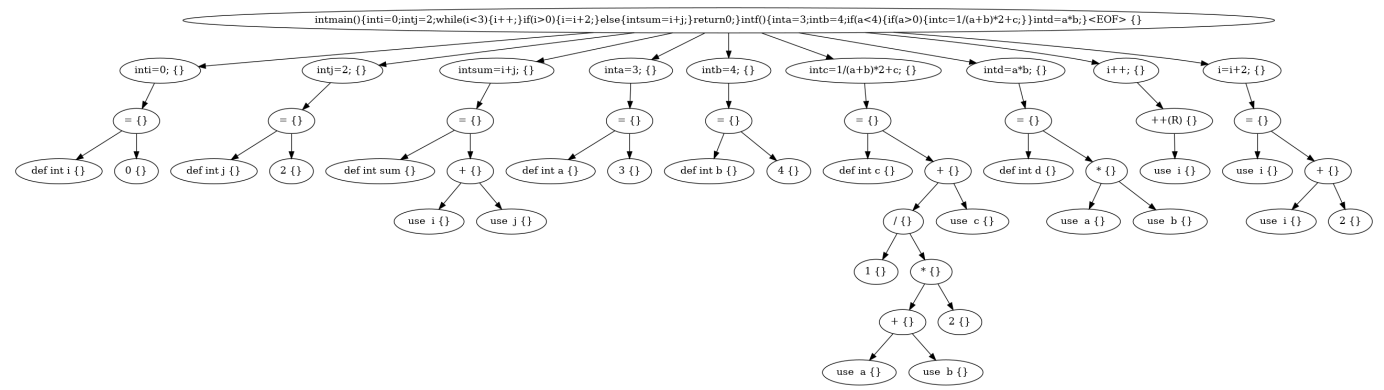
```
fn main() {

    // 读取命令选项，诸如 -c 表示代码文件地址
    // 你也可以通过运行 cargo run -- --help 来查看所有可用选项
    let args = Args::parse();
    // args.c_file_path = PathBuf::from_str("./demos/demo1.c").unwrap();
    let mut pass_manager = PassManager::new(args);
    let code2ast_pass = Code2AstPass::new(true);
    let ast2cfg_pass = Ast2CfgPass::new(true);
    let ast2et_debug_pass = Ast2EtDebugPass::new(true);
    let cfg2ncfg_pass = Cfg2NcfgPass::new(true);
    let ast2st_pass = Ast2StPass::new(true);
    add_passes!(
        code2ast_pass
        then ast2et_debug_pass
        then ast2cfg_pass
        then ast2st_pass
        then cfg2ncfg_pass
        to pass_manager
    );
    timeit!({pass_manager.execute_passes()}, "all passed finish");
}
```

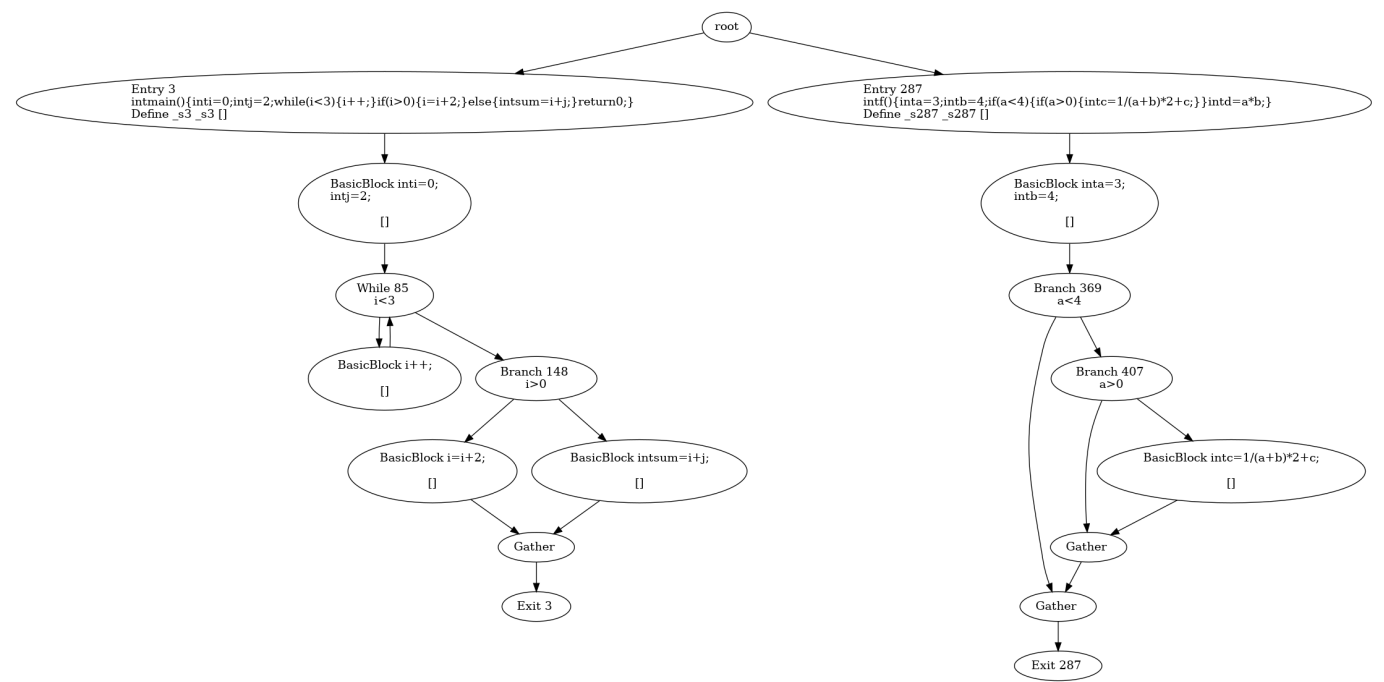
生成ast树的具体pass代码参见 [code2ast\\_pass.rs](#)，其他以此类推

```
impl Pass for Code2AstPass{
    // 运行这个pass
    fn run(&mut self, ctx:&mut Context) {
        println!("pass Code2AstPass run");
        ctx.code =
        read_file_content(ctx.args.c_file_path.to_string_lossy().into_owned());
        parse_as_ast_tree(ctx);
        // 生成对应的png
        if self.is_gen_png{
            let ast_tree = &mut ctx.ast_tree;
            generate_png_by_graph(&ast_tree, "ast_tree".to_string(), &
```





cfg



ncfg

