作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师

语法分类的相同点和不同点

汇编语法，分为 ATT 语法和 Intel 语法。
ATT 语法，起源于 ATT 公司贝尔实验室，主要用于 Unix、Linux 环境。本书讲解 ATT 语法。
Intel 语法，主要用于 Windows 环境。历史上，Windows 和 Intel 结盟，组成 wintel，使得 Intel 语法在 Windows 上面使用广泛。

语法的差异点：
寄存器的前缀,ATT 语法有%前缀,Intel 语法没有%前缀。比如,把 rbp 的值写入栈顶,ATT 语法表示为”pushq %rbp”，Intel 语法表示为”push rbp”。
常量的前缀,ATT 语法有$前缀,Intel 语法没有$前缀。比如,把 rsp 的值减去 32,ATT 语法表示为”subq $32, %rsp”，Intel 语法表示为”sub rsp, 32” 。
指令的参数顺序相反，ATT 语法的格式为”指令 src dest”，Intel 语法的格式为”指令 dest src”。比如，把 rsp 的值写入 rbp，ATT 语法表示为”movq %rsp, %rbp”，Intel 语法表示为”mov rbp, rsp”。
寄存器取地址,ATT 语法使用(),Intel 语法使用[ ] 。比如,把变量 global_age 的值写入 eax,ATT 语法表示为”movl global_age(%rip), %eax”，Intel 语法表示为“mov eax, DWORD PTR global_age[rip]”。
指令的长度，Intel 语法使用”DWORD PTR”，所以更长。相对来说，ATT 语法更紧凑。

语法的相同点：
程序的整体结构相同。主要包括，段 section(代码段、数据段)、符号 symbol(全局变量名、全局方法名等)、函数调用(call/ret 指令)、函数栈(寄存器 rbp/rsp 操作)。
功能等价，可以相互转换。

用代码比较语法分类的差异

编写代码： yufa.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

// 全局变量
int global_age = 300;

// 方法
int print_age(int age)
{
    // 局部变量
    int age_plus = age + 5;
    // 第一个参数是字符串
    printf("age = %d \n", age_plus);
}

// 主函数
```

```
int main()
{
    // 局部变量
    int local_age = 200;

    // 调用方法
    print_age(global_age);

    // 调用方法
    print_age(local_age);

    return 0;
}
```

编译代码：

```
# 编译为可执行文件
gcc yufa.c -o yufa

# 编译为汇编文件，ATT 语法
gcc -S yufa.c -o yufa.ATT.s

# 编译为汇编文件，Intel 语法
gcc -S yufa.c -masm=intel  -o yufa.intel.s
```

ATT 语法的汇编文件： yufa.ATT.s

```
    .file   "yufa.c"
    .globl  global_age
    .data
    .align 4
    .type   global_age, @object
    .size   global_age, 4
global_age:
    .long   300
    .section    .rodata
.LC0:
    .string "age = %d \n"
    .text
    .globl  print_age
    .type   print_age, @function
print_age:
.LFB2:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $32, %rsp
    movl    %edi, -20(%rbp)
```

```
    movl    -20(%rbp), %eax
    addl    $5, %eax
    movl    %eax, -4(%rbp)
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE2:
    .size   print_age, .-print_age
    .globl  main
    .type   main, @function
main:
.LFB3:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $200, -4(%rbp)
    movl    global_age(%rip), %eax
    movl    %eax, %edi
    call    print_age
    movl    -4(%rbp), %eax
    movl    %eax, %edi
    call    print_age
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE3:
    .size   main, .-main
    .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
    .section    .note.GNU-stack,"",@progbits
```

Intel 语法的汇编文件： yufa.intel.s

```
    .file   "yufa.c"
    .intel_syntax noprefix
    .globl  global_age
    .data
    .align 4
    .type   global_age, @object
```

```
    .size   global_age, 4
global_age:
    .long   300
    .section    .rodata
.LC0:
    .string "age = %d \n"
    .text
    .globl  print_age
    .type   print_age, @function
print_age:
.LFB2:
    .cfi_startproc
    push    rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    mov rbp, rsp
    .cfi_def_cfa_register 6
    sub rsp, 32
    mov DWORD PTR [rbp-20], edi
    mov eax, DWORD PTR [rbp-20]
    add eax, 5
    mov DWORD PTR [rbp-4], eax
    mov eax, DWORD PTR [rbp-4]
    mov esi, eax
    mov edi, OFFSET FLAT:.LC0
    mov eax, 0
    call    printf
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE2:
    .size   print_age, .-print_age
    .globl  main
    .type   main, @function
main:
.LFB3:
    .cfi_startproc
    push    rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    mov rbp, rsp
    .cfi_def_cfa_register 6
    sub rsp, 16
    mov DWORD PTR [rbp-4], 200
    mov eax, DWORD PTR global_age[rip]
    mov edi, eax
    call    print_age
    mov eax, DWORD PTR [rbp-4]
```

```
    mov edi, eax
    call    print_age
    mov eax, 0
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE3:
    .size   main, .-main
    .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
    .section    .note.GNU-stack,"",@progbits
```

分析结果：

比较项	ATT 语法	Intel 语法	分析
全局变量 int global_age = 300;	.globl global_age global_age: .long 300	.globl global_age global_age: .long 300	两边相同。 数字，放在 data 区。
字符串 "age = %d \n"	.LC0: .string "age = %d \n"	.LC0: .string "age = %d \n"	两边相同。 字符串，放在 data 区。 没有指定名称，所以分配名称 .LC0 。

方法的定义（开头部分）
int print_age(int age)
	print_age:
	pushq %rbp
	movq %rsp, %rbp
	subq $32, %rsp
		print_age:
		push rbp
		mov rbp, rsp
		sub rsp, 32
			rbp、rsp 是栈寄存器。
			寄存器，ATT 语法有%前缀，Intel 语法没有%前缀。
			mov、sub 指令的参数的顺序相反。

局部变量计算
int age_plus = age + 5;
	movl %edi, -20(%rbp)
	movl -20(%rbp), %eax
	addl $5, %eax
		mov DWORD PTR [rbp-20], edi
		mov eax, DWORD PTR [rbp-20]
		add eax, 5
			Intel 语法，指令更长，包含 DWORD PTR。
			add 指令的参数的顺序相反。
			数字常量，ATT 语法有$前缀，Intel 语法没有$前缀。

调用函数
print_age(global_age);
	movl global_age(%rip), %eax
	movl %eax, %edi
	call print_age
		mov eax, DWORD PTR global_age[rip]
		mov edi, eax
		call print_age
			都使用 call 指令。
			寄存器取地址，ATT 语法使用 ( )，Intel 语法使用[ ]。

调用函数
int local_age = 200;
print_age(local_age);
	movl $200, -4(%rbp)
	movl -4(%rbp), %eax
	movl %eax, %edi
	call print_age
		mov DWORD PTR [rbp-4], 200
		mov eax, DWORD PTR [rbp-4]
		mov edi, eax
		call print_age
			同上

退出函数	leave ret	leave ret	两边相同