第31章 属性说明符和标准属性

《现代C++语言核心特性解析》 谢丙堃

GCC和MSVC的属性语法

- GCC
  - __attribute__((attribute-list))

- MSVC
  - __declspec(attribute-list)

标准属性说明符语法

- 基本语法

```
[[attr]] [[attr1, attr2, attr3(args)]] [[namespace::attr(args)]]
```

- 例子

```
[[attr1]] class [[attr2]] X { int i; } a, b[[attr3]];
```

使用using打开属性的命名空间

- 有命名空间的情况

```
[[gnu::always_inline]] [[gnu::hot]] [[gnu::const]] [[nodiscard]]
inline int f();
// 或者
[[gnu::always_inline, gnu::hot, gnu::const, nodiscard]]
inline int f();
```

- 使用using

```
[[using gnu: always_inline, hot, const]][[nodiscard]]
inline int f()
```

标准属性

- noreturn
  - 声明函数不会返回

```
void foo() {}
void bar() {}
int main()
{
    foo();
    bar();
}

main:
.LFB2:
        push    rbp
        mov     rbp, rsp
        call    foo()
        call    bar()
        mov     eax, 0
        pop     rbp
        ret
.LFE2:
```

```
[[noreturn]] void foo() {}
void bar() {}
int main()
{
    foo();
    bar();
}

main:
.LFB2:
        push    rbp
        mov     rbp, rsp
        call    foo()
.LFE2:
```

标准属性

- carries_dependency
  - 允许跨函数传递内存依赖项，它通常用于弱内存顺序架构平台上多线程程序的优化，避免编译器生成不必要的内存栅栏指令。

标准属性

- deprecated
  - 带有此属性实体被声明为弃用的

```
[[deprecated]] void foo() {}
class [[deprecated]] X {};
int main()
{
    X x;
    foo();
}

[[deprecated("foo was deprecated, use bar instead")]]
void foo() {}
int main()
{
    foo();
}
```

标准属性

- fallthrough
  - 在switch语句的上下文中提示编译器直落行为是有意的

```
void bar() {}
void foo(int a) {
    switch (a) {
    case 0: break;
    case 1:
        bar();
        [[fallthrough]];
    case 2:
        bar();
        break;
    default: break;
    }
}
```

标准属性

- nodiscard
  - 声明函数的返回值不应该被舍弃

```
class [[nodiscard]] X {};
[[nodiscard]] int foo() { return 1; }
X bar() { return X(); };

int main()
{
    X x;
    foo();
    bar();
}
```

标准属性

- maybe_unused
  - 声明实体可能不会被使用

```
int foo(int a [[maybe_unused]], int b
[[maybe_unused]])
{
    return 5;
}

int main()
{
    foo(1, 2);
}
```

标准属性

- likely和unlikely
  - 声明在标签或者语句上
  - likely属性允许编译器对该属性所在的执行路径相对于其他执行路径更可能的进行优化
  - unlikely允许编译器对该属性所在的执行路径相对于其他执行路径更不可能的进行优化

标准属性

- no_unique_address
  - 指示编译器该数据成员不需要唯一地址

```cpp
struct Empty {};
struct X {
    int i;
    Empty e;
};
```

```cpp
struct Empty {};
struct X {
    int i;
    [[no_unique_address ]] Empty e;
};
```

```cpp
// main函数
std::cout << "sizeof(X) = " << sizeof(X) << std::endl
    << "X::i address = " <<  &((X*)0)->i << std::endl
    << "X::e address = " << &((X*)0)->e;
```

```
sizeof(X) = 8
X::i address = 0
X::e address = 0x4

sizeof(X) = 4
X::i address = 0
X::e address = 0
```

感谢您的观看

欢迎关注