第19章 static_assert声明

《现代C++语言核心特性解析》 谢丙堃

运行时断言的使用

- 例子

```
void* resize_buffer(void* buffer, int new_size)
{
    assert(buffer != nullptr);            // OK, 用assert检查函数参数
    assert(new_size > 0);
    assert(new_size <= MAX_BUFFER_SIZE);
    ...
}
bool get_user_input(char c)
{
    assert(c == '\0x0d');                 // 不合适, assert不应该用于检查外部输入
    ...
}
```

静态断言

- 静态断言用于在程序编译阶段评估常量表达式并对返回false的表达式断言。
- 对静态断言的要求：
    1. 所有处理必须在编译期间执行，不允许有空间或时间上的运行时成本。
    2. 它必须具有简单的语法。
    3. 断言失败可以显示丰富的错误诊断信息。
    4. 它可以在命名空间，类或代码块内使用。
    5. 失败的断言会在编译阶段报错。

静态断言的使用

- 例子

```
class A {};
class B : public A {};
class C {};
template<class T>
class E { static_assert(std::is_base_of<A, T>::value, "T is not base of A"); };

int main(int argc, char *argv[]) {
    static_assert(argc > 0, "argc > 0");     // 使用错误，argc>0不是常量表达式
    E<C> x;                                  // 使用正确，但由于A不是C的基类，所以触发断言
    static_assert(sizeof(int) >= 4,          // 使用正确，表达式返回真，不会触发失败断言
        "sizeof(int) >= 4");
    E<B> y;                                  // 使用正确，A是B的基类，不会触发失败断言
}
```

单参数static_assert

- 用宏来实现

```
#define LAZY_STATIC_ASSERT(B) static_assert(B, #B)
```

- C++17标准支持单参数static_assert

```
static_assert(sizeof(int) < 4);
```

感谢您的观看

欢迎关注