17 | 标准库：断言、错误处理与对齐

2022-01-21 于航

《深入C语言和程序运行原理》

课程介绍 >

讲述：于航

时长 11:38　大小 10.66M

你好，我是于航。

这一讲是这门课中关于 C 标准库的最后一讲。通过前面几讲的学习，相信你已经对 C 标准库提供的相关能力有了一个全面的认识。在此基础上，我们便可以使用这些成熟的接口，来更加方便地构建应用程序。这一讲后，在“ C 工程实战篇”的其他篇目中，我会和你一起讨论语言具体功能之外的性能优化、自动化测试、结构化编译等 C 工程化相关内容，并带你手把手实现一个简单的高性能 HTTP Server。

今天，我们来看一看与 C 标准库相关的最后三个话题：断言、错误处理，以及对齐。

断言为我们提供了一种可以静态或动态地检查程序在目标平台上整体状态的能力，与它相关的接口由头文件 assert.h 提供。错误处理则涉及 C 程序如何通过特定方式，判断其运行是否发生错误，以及错误的具体类型，头文件 errno.h 中则定义了与此相关的宏。除此之外，C 语言

还具有自定义数据对齐方式的能力，借助 stdalign.h 头文件提供的宏，我们可以轻松地做到这一点。

断言

在计算机编程中，断言是一种可用于判断程序设计或运行是否符合开发者预期的逻辑判断式。与断言相关的编程接口由标准库头文件 assert.h 提供。

在 C 语言中，断言被分为静态断言与运行时断言。其中，静态断言主要用来约束程序在编译时需要满足的一定要求；运行时断言则可以在程序运行过程中，判断一些支持程序正常运行的假设性条件是否满足。我们来看下面这个例子：

复制代码

```
#include <assert.h>
double sqrt(double x) {
  // 检查函数使用时传入的参数：
  assert(x > 0.0);
  // ...
}
int main(void) {
  // 检查程序的编译要求：
  static_assert(sizeof(int) >= 4,
    "Integer should have at least 4 bytes length.");
  // ...
  return 0;
}
```

可以很直观地看到，我们分别在代码的第 4 行和第 9 行使用到了运行时断言与静态断言。因为这里的重点内容是两种断言的具体使用方式，因此我们没有去实现一个完整的可运行程序，但这并不影响你对相关概念的理解。

接下来，我们进一步看看静态断言和运行时断言在 C 语言中的使用方式和适用场景。

静态断言

从刚才的例子中可以看到，在 main 函数内部，通过名为 static_assert 的宏，我们可以限定程序在被编译时，其所在平台上 `int` 类型的宽度需要大于等于 4 字节。否则，编译会被终止，对应的错误信息也会被打印出来。

实际上，在预处理阶段，static_assert 宏会被展开成名为 `_Static_assert` 的 C 关键字。该关键字以类似“函数调用”的形式在 C 代码中使用，它的第一个参数接收一个常量表达式。程序在被编译时，编译器会对该表达式进行求值，并将所得结果与数字 0 进行比较。若两者相等，则程序终止编译，并会将通过第二个参数指定的错误信息，与断言失败信息合并输出。若两者不相等，程序会被正常编译，且该关键字对应的 C 代码不会生成任何对应的机器指令。

一般来说，我们会在程序运行前使用静态断言，来检查它所需要满足的一系列要求。比如，在上面这个例子中，程序的正常运行便依赖于一个前置条件，即 `int` 类型的宽度需要满足至少 4 字节。而通过静态断言，开发者便可以提前得知，程序如果运行在当前平台上，是否能正常工作。

类似的用例还有很多，比如判断 `char` 类型的默认符号性（借助 CHAR_MIN 宏常量），或是判断指针类型与 `int` 类型的宽度是否相等，或是判断某个结构体的大小是否满足预期要求，等等。这些都是可能影响 C 程序运行正确性的因素，而通过静态断言，它们都可以在编译时被提前检测出来。

运行时断言

还是上面那段代码，在代码中名为 sqrt 的函数实现里，我们使用到了运行时断言。这里，通过名为 assert 的宏，程序可以在函数主要逻辑被调用时，首先判断用于支持函数正常运作的假设性条件是否成立。

这个函数的功能是计算给定数字值的平方根，因此要保证传入参数 x 的值大于 0。而通过运行时断言，我们便能够做到这一点。但与静态断言使用的 static_assert 不同，assert 并不支持自定义错误消息。那么，我在这里留下一个小问题：你知道怎样才能在运行时断言失败时，将我们自定义的错误消息也显示给开发者吗？欢迎在评论区分享你的思考。

另外还需要注意的是，C 程序中的运行时断言是否可用，也会受到宏常量 NDEBUG 的影响。当该宏常量的定义先于 `#include <assert.h>` 语句出现时，编译器会忽略对 assert 宏函数调用代码的编译。反之，它便会在程序运行时进行正常的断言检查。通过这种方式，我们可以相对灵活地控制运行时断言的启用与关闭。

通常来说，运行时断言可被应用于“契约式编程（Design by Contract）”与“防御式编程（Defensive Programming）”这两种软件设计方法中。如果你对这两个概念感兴趣，可以点击链接来参考更多信息。

最后，让我们再从这两种软件设计方法的角度，回顾一下函数 sqrt 内部使用的运行时断言。这里，函数 sqrt 通过运行时断言，保证了它的主要逻辑被执行前，相关必要条件（即传入的参数值大于 0）需要被首先满足。从防御式编程的角度来看，这是预防函数调用时发生错误的一种措施。而从契约式编程的角度来看，这就是被调用函数进行契约（函数调用前置条件）检查的过程。

错误处理

在 C 语言中，名为 errno 的预处理器宏会被展开为一个 `int` 类型的可修改全局左值，也就是说，我们可以直接对它进行赋值操作（这里为了便于描述，下面我再次提及 errno 时，均指代这个左值）。

在这个值中，便存放有程序自上一次调用 C 标准库函数后的状态信息。该宏由标准库头文件 errno.h 提供，在默认情况下，errno 中存放着数字值 0，表示程序正常运行。随着程序不断调用各种标准库函数，当某一时刻某个函数的执行产生了不符合预期的结果时，函数便会通过修改 errno 的值，来向程序传达这一消息。我们来看下面这个例子：

复制代码

```
1 #include <tgmath.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <errno.h>
5 int main(void) {
6   sqrt(-1);
7   fprintf(stderr, "%s\n", strerror(errno));  // "Numerical argument out of doma
8   return 0;
9 }
```

这里，在代码的第 6 行，我们用实参值 “-1” 调用了用于求取平方根的标准库函数 sqrt。可以看到，由于负数在实数域内没有平方根，因此这是一种错误的使用方式。我们在上一小节中使用了运行时断言来终止程序执行，而这里，标准库函数会通过设置 errno 来向程序反馈相应的错误信息，但不会终止程序的运行。

紧接着，在代码的第 7 行，我们可以通过 strerror 函数，来得到当前 errno 中存放的数字值所表示状态对应的可读文本。然后，借由 fprintf 函数，该文本被“发送”到标准错误流中。同样地，使用 string.h 头文件提供的 perror 函数，我们也可以达到类似的效果。

实际上，C11 标准中仅规定了 errno 可能取得的三个枚举值，我将它们的具体值，以及对这些值的描述信息整理在了下面的表格中，供你参考。

枚举值	描述
EDOM	数字参数值超过了函数的可处理范围
EILSEQ	遇到非法的字节序列
ERANGE	函数调用结果值过大

除此之外，POSIX 标准、C++ 标准库，甚至不同的操作系统实现，都可能会为 errno 定义额外的可选枚举值，用来表示更多不同场景下的错误情况。

不仅如此，**C 语言还为 errno 添加了线程本地属性**。这也就意味着，在程序不同线程中发生的错误，将会使用专属于本线程的 errno 来存放相应的错误标识数值。你可以通过下面这段代码来验证这个结论：


```
#include <threads.h>
#include <stdio.h>
#include <errno.h>
#include <tgmath.h>
int run(void* data) {
  log(0.0);
  perror("Run");  // "Run: Numerical result out of range".
  return thrd_success;
}
int main(void) {
#ifndef __STDC_NO_THREADS__
  thrd_t thread;
  thrd_create(&thread, run, NULL);
  thrd_join(thread, NULL);
  perror("Main");  // "Main: Success".
#endif
  return 0;
}
```

这样看起来，errno 在错误检查方面表现得还不错，但在使用时我们仍然需要注意很多问题。通常来说，我们可以把标准库函数在遇到执行错误时的具体行为分为下面四类：

- 设置 errno，并返回仅用于表示执行错误的值，如 ftell；
- 设置 errno，并返回可同时用于表示执行错误及正常执行结果的值，如 strtol；
- 不承诺设置 errno，但可能会返回表示执行错误的值，如 setlocale；
- 在不同标准（比如 ISO C 和 POSIX）下有不同行为，如 fopen。

可以看到，在执行发生错误时，并非所有 C 标准库函数都会对 errno 的值进行合理的设置。因此，仅通过该值来判断函数执行是否正常可能并不明智。

更加合适的做法是，当你明确知道所调用库函数会返回唯一的、不具有歧义，且不会与其正常返回值混用的错误值时，应直接使用该值来进行判断。而当不满足这个条件时，再使用 errno 来判断错误是否发生，以及错误的具体类型。同时，建议你养成一个好习惯：**每一次调用相应的库函数前，都应首先将 errno 置零，并在函数调用后，及时对它的值进行检测。**

自定义数据对齐

最后，我们再来看看有关对齐的内容。关于对齐的一些理论性知识，我曾在 07 讲 中为你介绍过。如果觉得记忆有些模糊了，可以先点击链接去那一讲温习下。

这里，我们来看看如何在 C 语言中为数据指定自定义的对齐方式。默认情况下，编译器会采用自然对齐，来约束数据在内存中的起始位置。但实际上，我们也可以使用 C11 提供的关键字 `_Alignas`，来根据自身需求为数据指定特殊的对齐要求。并且，头文件 stdalign.h 还为我们提供了与其对应的宏 alignas，可以简化关键字的使用过程。来看下面这段代码：

复制代码

```
#include <stdio.h>
#include <stdalign.h>
int main(void) {
#if __alignas_is_defined == 1 && __alignof_is_defined == 1
  alignas(1024) int n = 1;
  printf("The alignment of n is %zu\n", alignof(n));  // "The alignment of n is
  printf("The address of n is: %p\n", &n);  // "The address of n is: 0x7ffe8065
#endif
  return 0;
}
```

这里，在代码的第 4 行，我们首先通过验证宏常量 __alignas_is_defined 与 __alignof_is_defined 的值是否为 1，来判断当前编译环境是否支持 alignas 与 alignof 这两个宏。

紧接着，在代码第 5 行，通过在变量 n 的定义中添加 `alignas(1024)` 标识符，我们可以限定，变量 n 的值被存放在内存中时，其起始地址必须为 1024 的倍数。而在接下来代码的第 6~7 行，我们分别通过使用 alignof 宏函数和直接查看地址这两种方式，来验证我们对变量 n 指定的对齐方式是否生效。

同 alignas 类似的是，宏函数 alignof 在展开后也会直接对应于 C11 新引入的运算符关键字 `_Alignof`，而该关键字可用于查看指定变量需要满足的对齐方式。并且，通过打印变量 n 的地址，你会发现，这个例子中结尾处的三位 16 进制数字 “c00”，也表示该地址已经在 1024 的边界上对齐。

表面上看，alignas 只是用来修改数据在内存中的对齐方式的。但实际上，合理地运用这个功能，我们还可以优化程序在某些情况下的运行时性能。在下一讲中，我会为你详细介绍这些内容。

总结

好了，讲到这里，今天的内容也就基本结束了。最后我来给你总结一下。

在这一讲中，我们主要讨论了如何借助 C 标准库提供的相关接口，在 C 程序中使用各种断言和检查库函数调用错误，以及使用自定义数据对齐方式。

通过 assert.h 头文件提供的 static_assert 与 assert 宏函数，我们可以在 C 代码中使用静态断言与运行时断言。其中，前者主要用于约束程序在编译时需要满足的环境要求，而后者则可被应用于防御式编程与契约式编程等程序设计方法中。

通过访问 errno.h 头文件提供的预处理器宏 errno，我们能够得知程序在调用某个标准库函数后，该函数的执行是否发生了错误。而通过定义在 stdalign.h 头文件中的宏函数 alignas 与 alignof，我们可以为数据指定除自然对齐外的其他对齐方式。

思考题

这一讲的最后，给你留个小作业：请你查阅相关文档，并在评论区告诉我你对契约式编程的理解。

今天的课程到这里就结束了，希望可以帮助到你，也希望你在下方的留言区和我一起讨论。同时，欢迎你把这节课分享给你的朋友或同事，我们一起交流。

上一篇 16 | 标准库：日期、时间与实用函数

下一篇 18 | 极致优化（上）：如何实现高性能的 C 程序？

精选留言 (1)

写留言

zxk
2022-03-14

契约式编程是一种编程风格，类比商业中的服务供应商与客户的，将编程中划分为了服务提供方与服务调用方，两者之间的关系如下：
1. 服务提供方期望服务调用方能够遵循一定的规范进行调用，这是服务调用方应满足服务提供方定下的先验条件
2. 服务提供方退出时需要保证能够返回特定的结果，这是服务提供方承诺服务调用方定下的后验条件
3. 先验条件与后验条件之间的交集，在进入与退出之后都应保持不变

具体到编程来说，就是客户端遵循一定规范调用方法接口，方法内部会对函数做一定程度的检查（通常是断言 + 异常信息，但 C 中是 errno 的间接方式），客户端会期望方法接口具有特定的行为，并返回一个可预测的返回值。
个人浅显理解，不知道对不对。

作者回复: 讲解的很赞！