道哥的第 023 篇原创

一、前言

我们在撸代码的时候，经常需要对代码的安全性进行检查，例如：

1. 指针是否为空？
2. 被除数是否为 0？
3. 函数调用的返回结果是否有效？
4. 打开一个文件是否成功？

对这一类的边界条件进行检查的手段，一般都是使用 if 或者 assert 断言，无论使用哪一个，都可以达到检查的目的。那么是否就意味着：这两者可以随便使用，想起来哪个就用哪个？

这篇小短文我们就来掰扯掰扯：在不同的场景下，到底是应该用 if，还是应该使用 assert 断言？

写这篇文章的时候，我想起了孔乙己老先生的那个问题：茴香豆的“茴”字有几种写法？

似乎我们没有必要来纠结应该怎么选择，因为都能够实现想要的功能。以前我也是这么想的，但是，现在我不这么认为。

成为技术大牛、拿到更好的offer，也许就在这些细微之间就分出了胜负。

二、assert 断言

刚才，我问了下旁边的一位工作 5 年多的嵌入式开发者：if 和 assert 如何选择？他说：assert 是干什么的？！

看来，有必要先简单说一下 assert 断言。

assert() 的原型是：

```
void assert(int expression);
```

1. 如果宏的参数求值结果为非零值，则不做任何操作（no action）；
2. 如果宏的参数是零值，就打印诊断消息，然后调用abort()。

例如下面的代码：

```
#include <assert.h>
int my_div(int a, int b)
{
    assert(0 != b);
    return a / b;
}
```

1. 当 b 不为 0 时，assert 断言什么都不做，程序往下执行；
2. 当 b 为 0 时，assert 断言就打印错误信息，然后终止程序；

从功能上来说，`assert(0 != b);` 与下面的代码等价：

```
if (0 == b)
{
    fprintf(stderr, "b is zero...");
    abort();
}
```

assert 是一个宏，不是一个函数

在 assert.h 头文件中，有如下定义：

```
#ifdef NDEBUG
    #define assert(condition) ((void)0)
#else
    #define assert(condition) /*implementation defined*/
#endif
```

既然是宏定义，说明是在预处理的时候进行宏替换。(关于宏的更多内容，可以看一下这篇文章:提高代码逼格的利器：宏定义-从入门到放弃)。

从上面的定义中可以看到：

1. 如果定义了宏 NDEBUG，那么 assert() 宏将不做什么动作，也就是相当于一条空语句：`(void)0;`，当在 release 阶段编译代码的时候，都会在编译选项中(Makefile)定义这个宏。
2. 如果没有定义宏 NDEBUG，那么 assert() 宏将会把一些检查代码进行替换，我们在开发阶段执行 debug 模式编译时，一般都会屏蔽掉这 NDEBUG 这个宏。

三、if VS assert

还是以一个代码片段来描述问题，以场景化来讨论比较容易理解。

```
// brief: 把两个短字符串拼接成一个字符串
char *my_concat(char *str1, char *str2)
{
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int len3 = len1 + len2;
    char *new_str = (char *)malloc(len3 + 1);
    memset(new_str, 0 len3 + 1);
    sprintf(new_str, "%s%s", str1, str2);
    return new_str;
}
```

如果一个开发人员写出上面的代码，一定会被领导约谈的！它存在下面这些问题：

1. 没有对输入参数进行有效性检查；
2. 没有对 malloc 的结果进行检查；
3. sprintf 的效率很低；
4. ...

1. 使用 if 语句来检查

```
char *my_concat(char *str1, char *str2)
{
    if (!str1 || !str2)  // 参数错误
```

```
        return NULL;

    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int len3 = len1 + len2;
    char *new_str = (char *)malloc(len3 + 1);

    if (!new_str)    // 申请堆空间失败
        return NULL;

    memset(new_str, 0 len3 + 1);
    sprintf(new_str, "%s%s", str1, str2);
    return new_str;
}
```

2. 使用 assert 断言来检查

```
char *my_concat(char *str1, char *str2)
{
    // 确保参数正确
    assert(NULL != str1);
    assert(NULL != str2);

    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int len3 = len1 + len2;
    char *new_str = (char *)malloc(len3 + 1);

    // 确保申请堆空间成功
    assert(NULL != new_str);

    memset(new_str, 0 len3 + 1);
    sprintf(new_str, "%s%s", str1, str2);
    return new_str;
}
```

3. 你喜欢哪一个?

首先声明一点：以上这 2 种检查方式，在实际的代码中都很常见，从功能上来说似乎也没有什么影响。因此，没有严格的错与对之分，很多都是依赖于每个人的偏好习惯不同而已。

（1） assert 支持者

我作为 `my_concat()` 函数的实现者，目的是拼接字符串，那么传入的参数必须是合法有效的，调用者需要负责这件事。如果传入的参数无效，我会表示十分的惊讶！怎么办：崩溃给你看！

（2）if 支持者

我写的 `my_concat()` 函数十分的健壮，我就预料到调用者会乱搞，故意的传入一些无效参数，来测试我的编码水平。没事，来吧，我可以处理任何情况！

这两个派别的理由似乎都很充足！那究竟该如何选择？难道真的的跟着感觉走吗？

假设我们严格按照常规的流程去开发一个项目：

1. 在开发阶段，编译选项中不定义 NDEBUG 这个宏，那么 assert 就发挥作用；
2. 项目发布时，编译选项中定义了 NDEBUG 换个宏，那么 assert 就相当于空语句；

也就是说，只有在 debug 开发阶段，用 assert 断言才能够正确的检查到参数无效。而到了 release 阶段，assert 不起作用，如果调用者传递了无效参数，那么程序只有崩溃的命运了。

这说明什么问题？是代码中存在 bug？还是代码写的不够健壮？

从我个人的理解上看，这压根就是单元测试没有写好，没有测出来参数无效的这个 case！

4. assert 的本质

assert 就是为了验证有效性，它最大作用就是：在开发阶段，让我们的程序尽可能地 crash。每一次的 crash，都意味着代码中存在着 bug，需要我们去修正。

当我们写下一个 assert 断言的时候，就说明：断言失败的这种情况是不可以的，是不被允许的。必须保证断言成功，程序才能继续往下执行。

5. if-else 的本质

if-else 语句用于逻辑处理，它是为了处理各种可能出现的情况。就是说：每一个分支都是合理的，是允许出现的，我们都要对这些分支进行处理。

6. 我喜欢的版本

```
char *my_concat(char *str1, char *str2)
{
    // 参数必须有效
    assert(NULL != str1);
    assert(NULL != str2);

    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int len3 = len1 + len2;
    char *new_str = (char *)malloc(len3 + 1);

    // 申请堆空间失败的情况，是可能的，是允许出现的情况。
    if (!new_str)
        return NULL;

    memset(new_str, 0 len3 + 1);
    sprintf(new_str, "%s%s", str1, str2);
    return new_str;
}
```

对于参数而言：我认为传入的参数必须是有效的，如果出现了无效参数，说明代码中存在 bug，不允许出现这样的情况，必须解决掉。

对于资源分配结果(malloc 函数)而言：我认为资源分配失败是合理的，是有可能的，是允许出现的，而且我也对这个情况进行了处理。

当然了，并不是说对参数检查就要使用 assert，主要是根据不同的场景、语义来判断。例如下面的这个例子：

```
int g_state;
void get_error_str(bool flag)
{
    if (TRUE == flag)
    {
        g_state = 1;
        assert(1 == g_state); // 确保赋值成功
    }
    else
```

```
    {
        g_state = 0;
        assert(0 == g_state); // 确保赋值成功
    }
}
```

flag 参数代表不同的分支情况，而赋值给 g_state 之后，必须保证赋值结果的正确性，因此使用 assert 断言。

五、总结

这篇文章分析了 C 语言中比较晦涩、模糊的一个概念，似乎有点虚无缥缈，但是的确又需要我们停下来仔细考虑一下。

如果有些场景，实在拿捏不好，我就会问自己一个问题：

这种情况是否被允许出现？

不允许：就用 assert 断言，在开发阶段就尽量找出所有的错误情况；

允许：就用 if-else，说明这是一个合理的逻辑，需要进行下一步处理。

不吹嘘，不炒作，不浮夸，认真写好每一篇文章！

欢迎转发、分享给身边的技术朋友，道哥在此表示衷心的感谢！ 转发的推荐语已经帮您想好了：

道哥总结的这篇总结文章，写得很用心，对我的技术提升很有帮助。好东西，要分享！

推荐阅读

C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
一步步分析-如何用C实现面向对象编程
我最喜欢的进程之间通信方式-消息总线
物联网网关开发：基于MQTT消息总线的设计过程(上) 提高代码逼格的利器：宏定义-从入门到放弃
原来gdb的底层调试原理这么简单
利用C语言中的setjmp和longjmp，来实现异常捕获和协程
关于加密、证书的那些事
深入LUA脚本语言，让你彻底明白调试原理