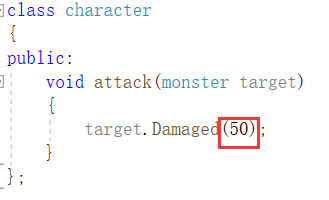
硬编码相关文档

1. 什么是硬编码

硬编码就是将一些常量值直接以值的形式写到用到这些值的地方。

例如以下代码



这个脚本表示角色对怪物进行攻击造成50伤害

这个50就是硬编码,而且这个值显然是可调的,但直接写在函数里面可能就会令后面调整的时候难以找到,而且可能多次引用

不恰当的硬编码通常出现在UI界面的元素,表格数据提取的行列编号,界面风格属性设置,一些默认参数等

1. 硬编码的危害
2. 硬编码的位置通常在实现文件(cpp),实现文件通常复杂(访问函数一般在头文件链接访问),所以要更改的时候会导致很难找到
3. 硬编码的值可能在多个地方被引用,如果都使用硬编码,会导致要修改的时候很容易漏掉,然后引发bug,而且修改的时候需要不断寻找所有位置也极大提高了工作量
4. 硬编码的使用不灵活,难以适应迭代开发
5. 硬编码的使用

虽然硬编码带来很大的危害,但并不是完全禁止,它在某些时候还是能发挥作用的

除去某些安全功能,硬编码通常会在下列情况使用

1. 一些简单提示通知,例如输入不合法提示等,直接使用字符串常量发送提示信息方便代码编写,但大型项目中一般会引入错误代码体系来将提示信息规范化
2. 一些不得不使用硬编码的区域

例如文件存档时有系统数据块需要对每个属性逐一读取和写入,此时使用硬编码标定数据名称,但为了避免硬编码带来的迭代困难,通常要求文件存取的这部分代码需要单独写在一个文件内,并且位置相近,只能在该文件的两个方法内引用,以方便查找和同步

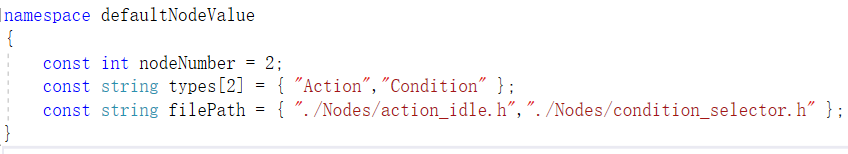
例:



1. 确定不会被多次引用或不被修改的值,以硬编码形式写在代码中可以确保安全,通常是完全固定的数学常量,或者默认值
2. 硬编码的修复与替代

对于一些默认数值会被多次引用,那么就不能以硬编码的形式写在实现方法中,而是应该以常量的形式定义在头文件中(或全局常量文件)

在C++中,常量应该以命名空间的方式定义,如:



使用命名空间有几个好处

1. 相比宏定义和全局常量,避免重名错误,常量命名方式自由,不必要全大写
2. 相比类内定义静态成员,更为灵活且可以很简单地设置值
3. 在某个区域内如果确定不会发生重名错误还能使用using namespace来引用命名空间,省去了引用常量时要加xxx::constName这样的前缀导致语句很长很难读
4. 相比静态成员,常量的引用通常不需要访存,提高运行效率
5. 项目开发核心思想

在编辑项目代码的时候,要记住一个核心思想:以后会不会改,如何方便以后的修改(以后的自己会不会想回来打死现在的自己),以及别人能不能看懂!

有这个迭代意识就能避免很多不规范的代码

通常代码阅读的基础有两个:命名和注释,如果没能想到很好的命名,请在对应的属性/方法定义的前一行加上详细的注释,通常一个方法定义命名和注释都要好读能顾名思义