ICS 25.040.20
J09
备案号：

JB

中华人民共和国机械行业标准

JB/T 13775—2020

工业机械电气设备及系统 数控系统软件白盒测试规范

Electrical equipment and system of industrial machines — Specification for numerical control system software white-box testing

（报批稿）

2018-06-28

XXXX－XX－XX 发布　　XXXX－XX－XX 实施

中华人民共和国工业和信息化部　发布

目　次

前　言

本标准按GB/T 1.1—2009《标准化工作导则 第1部分：标准的结构和编写》给出的规则起草。

本标准由中国机械工业联合会提出。

本标准由全国工业机械电气系统标准化技术委员会（SAC/TC 231）归口。

本标准主要起草单位：国家机床质量监督检验中心、沈阳高精数控智能技术股份有限公司、山东建筑大学、广州数控设备有限公司、武汉华中数控股份有限公司、浙江省机电设计研究院有限公司、北京凯恩帝数控技术有限责任公司、上海开通数控有限公司、大连光洋科技集团有限公司、北京计算机技术及应用研究所等。

本标准主要起草人：陈渌萍、黄祖广、于东、韩方旭、姬帅、张文博、王勇、蒋峥、刘涛、薛瑞娟、巩潇、杜瑞芳、陈建明、杨洪丽、武南、韩文业、孙文貌。

本标准为首次发布。

引 言

白盒测试用于保障被测对象代码的质量，从逻辑结构上考察代码的内部质量，主要测试内容包括静态结构分析、代码质量度量、代码检查、逻辑测试、函数（方法）执行性能测试、动态内存分析以及代码安全测试，以保障被测对象的质量。

工业机械电气设备及系统
数控系统软件白盒测试规范

1 范围

本标准规定数控系统软件白盒测试的要求及对应的测试方法。

本标准适用于金属加工机械、木工机械、锻压机械用数控系统软件白盒测试与评价，其他工业机械用数控系统软件亦可参照此标准。

2 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅所注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 11457-2006 信息技术 软件工程术语

GB/T 26220-2010 工业自动化系统与集成 机床数值控制 数控系统通用技术条件

GB/T 18759.5-2016 机械电气设备 开放式数控系统 第5部分：软件平台

JB/T XXXXX-XXXX 工业机械电气设备及系统 数控系统软件可靠性测试规范

3 术语、定义、符号及缩略语

GB/T 11457-2006，GB/T 26220-2010所界定的以及下列术语和定义适用于本文件。为了便于使用，以下重复列出GB/T 26220-2010中的一些术语和定义。

3.1

数控系统 numerical control system

使用数值数据的控制系统,在运行过程中不断地引入数值数据,从而实现机床加工过程的自动控制。

[GB/T 26220-2010,定义3.1]

3.2

数控系统软件 numerical control system software

包括数控系统的实时操作系统、中间件、应用软件及应用编程接口。

[GB/T 18759.5-2016,定义3.1.16-3.1.19]

3.3

白盒测试 white-box testing

侧重于系统或部件内部机制的测试。

注：改写GB 11457-2006,定义2.1849

3.4 **软件单元 software unit**

一段可分开编译的代码。

[GB/T 18759.5-2016,定义2.1537]

4 测试对象

数控系统软件架构应参照 JB/T XXXXX-XXXX4.1 的要求，白盒测试对象的范围包括数控系统软件架构中开发人员可编程的模块。

进行数控系统软件白盒测试时，需提交的文档包括但不限于：

a) 软件需求规格说明书；

b) 软件设计说明书；

c) 软件用户手册。

5 测试内容及测试方法

5.1 静态测试

5.1.1 静态结构分析

5.1.1.1 测试内容

静态结构分析是对代码的机械性的、程式化的特性进行分析，测试项目包括控制流分析、数据流分析、表达式分析。

5.1.1.2 测试方法

使用测试工具获取软件的内部结构数据，自动或人工检查是否符合要求,测试项见表 1。

表 1 静态结构分析测试项目表

测试项目	测试内容	技术要求
控制流分析	是否存在任何条件下都不能运行到的代码	不存在
	是否存在不影响任何输入/输出的代码	不存在
	是否存在不合理的循环结构（例如终止条件不正确、错误地修改循环变量等）	不存在
	是否存在失败的递归过程（例如无穷的递归调用，递归次数过多造成堆栈数据溢出）	不存在
	是否存在无效的函数参数	不存在
	是否存在多个函数出口	不存在
	是否存在浮点数相等比较	不存在
	是否使用 goto 语句	不使用
	是否使用赋值测试语句	不使用

表 1 静态结构分析测试项目表（续）

测试项目	测试内容	技术要求
数据流分析	变量和常量是否被引用	被引用
	变量使用是否初始化	初始化
	传递参数值是否正确	正确
表达式分析	表达式中的括号是否使用正确	使用正确
	是否存在数组下标越界	不存在
	是否存在表达式中的除数为 0	不存在
	是否存在输入/输出数据超出定义域值域范围	不存在

5.1.2 代码质量度量

5.1.2.1 测试内容

质量度量是软件质量特性的量化表示。代码质量度量是通过分析软件源代码，建立软件质量模型，对软件质量特性进行量化评估。

注：软件质量模型的建立可参照 GB/T 30961-2014，本标准只列出质量模型中比较重要的测试指标。

5.1.2.2 测试方法

使用测试工具获取软件的内部结构数据，自动或人工检查是否符合要求，测试项见表 2。

表 2 代码质量度量测试项目表

测试项目	测试内容	技术要求
静态质量度量	软件单元的语句数	≤200
	软件单元的有效注释率	≥20%
	函数调用的下层函数个数	<7
	函数参数个数	≤8
	圈复杂度（不适用于 switch、case 结构）	≤10
	基本复杂度	≤4

5.1.3 代码检查

5.1.3.1 测试内容

主要检查代码和设计的一致性，代码对标准的遵循程度与可读性，代码的逻辑表达的正确性，代码结构的合理性等方面。

5.1.3.2 测试方法

通过代码测试工具或人工方式检查代码，若使用测试工具，在测试工具中选择需遵守的编码规则，使用测试工具进行分析后，对分析结果进行人工处理，测试项见表 3。

表 3　代码检查测试项目表

测试项目	测试内容	技术要求
编程风格检查	按照代码编写规范，该缩进的地方（如配对出现的语句、嵌套的 IF 语句、类声明定义等）是否已正确地缩进	正确缩进
	程序代码布局结构是否清楚	清楚
	注释是否准确并有意义，例如在每一个模块之前，是否有注释说明，描述该模块的输入/输出、参数、功能处理和其调用的外部模块以及该模块是否有使用限制等	准确有意义
	是否有多余的资源定义和宏定义	没有
	头文件是否使用了 ifndef/define/endif 预处理块	使用
	程序结构和模块功能定义是否清楚	清楚
	是否遵循该语言的指令编写格式	遵循
	注释说明和代码功能是否一致	一致
	错误处理分支信息表达是否清楚	清楚
	模块内是否做到了高内聚、模块之间是否达到了低耦合	达到
	模块的扇出是否不超出 7-9 之间	不超出
	是否屏蔽了没有明确含义的输入和按键	屏蔽
	常量、变量、类、数据结构等命名是否有意义	有意义
函数接口检查	实参和形参的个数、属性和次序是否一致	一致
	对另一个模块的每一次调用：全部所需的参数是否已传送给每一个被调用的模块，被传送的参数值是否正确设置	准确传送
	函数功能是否齐全	齐全
	函数返回值类型是否正确	正确
	return 语句是否有返回指向“栈内存”的“指针”或者“引用”	没有
	函数的返回值是否全面反应了各种状态和结果	全面
程序语言检查	动态链接库和外部设备接口驱动程序使用是否正确	正确
	动态分配的指针是否在不使用之后删除，并释放内存	删除并释放

表 3　代码检查测试项目表（续）

测试项目	测试内容	技术要求
程序语言检查	调用类成员函数或 API 函数时，是否检查了返回值	已检查
	文件、数据库和注册表等打开后，在对其进行操作之后是否进行了关闭	进行关闭
	对于使用附带例外的函数是否增加了例外处理程序，如对数据库或文件操作	增加
	变量的数据类型定义是否合理	合理
	程序中是否出现相同的局部变量和全部变量	不出现
	数据类型转换是否使用了正确的转换函数并转换正确	使用正确
	是否使用了只用于调试版本的函数、宏等	不使用
	有多个线程的程序中，资源分配是否合理，不会造成死锁	合理
	在使用 GDI 对象后是否进行删除	进行删除
	变量的作用域和生命期是否满足设计的目的	满足
	表达式中运算优先级是否正确	正确
	是否缺少 switch 的 default 分支	不缺少
	使用 goto 语句时是否留下隐患， 例如跳过了某些对象的构造、变量的初始化、重要的计算等	无隐患
	Case 语句的结尾是否缺少 break	不缺少
	如果有运算符重载，则检查运算符重载是否正确	正确
类检查	类封装是否合理，检查成员函数和成员变量的访问属性是否满足操作要求	满足
	外部是否可以修改类的行为	不可以
	多重继承中，虚拟函数定义是否明确	明确
	继承类和自定义类所封装的函数和过程是否合理，类的功能是否详细，全面	合理
	是否违背编程规范而让 C++ 编译器自动为类产生四个缺省的函数，包括： 1）缺省的无参数构造函数 2）缺省的拷贝构造函数 3）缺省的析构函数 4）缺省的赋值函数	不违背
	构造函数中是否遗漏了某些初始化工作	不遗漏

表 3 代码检查测试项目表（续）

测试项目	测试内容	技术要求
类检查	是否正确地使用构造函数的初始化表	正确
	析构函数中是否遗漏了某些清除工作	不遗漏
	是否错用了拷贝构造函数和赋值函数	未错用
	赋值函数是否遗漏了重要步骤，包括： 1）检查自赋值 2）释放原有内存资源 3）分配新的内存资源，并复制内容 4）返回*this	不遗漏
	是否违背了继承和组合的规则？ 1）若在逻辑上 B 是 A 的“一种”，并且 A 的所有功能和属性对 B 而言都有意义，则允许 B 继承 A 的功能和属性。 2）若在逻辑上 A 是 B 的“一部分”（a part of），则不允许 B 从 A 派生，而是要用 A 和其它东西组合出 B。	不违背
内存检查	每一个域在每一次使用前是否正确地初始化	正确
	是否缺少为数组和动态内存赋初值，防止将未被初始化的内存作为右值使用	不缺少
	动态内存的申请与释放是否配对，以防止内存泄漏	配对
	是否有效地处理了“内存耗尽”问题	有效处理
	是否出现修改“指向常量的指针”的内容	不出现
	每个域是否已由正确的变量类型声明	正确声明
	存储区是否存在重复使用，以防止出现冲突	不重复
	用 malloc 或 new 申请内存之后，是否立即检查指针值是否为 NULL，以防止使用指针值为 NULL 的内存	检查
	是否出现野指针，包括： 1）指针变量没有被初始化 2）用 free 或 delete 释放了内存之后，忘记将指针设置为 NULL	不出现
	未使用的内存中的内容是否影响系统安全	不影响

表 3 代码检查测试项目表（续）

测试项目	测试内容	技术要求
测试和转移检查	条件逻辑组合是否正确	正确
	逻辑“或”中一个条件满足就执行对其它逻辑表达式是否有影响	无影响
	是否使用正确的变量用于测试	正确
	每个转移目标是否正确并至少执行一次	正确
	三种情况（大于 0，小于 0，等于 0）是否测试完整，边界值是否进行了测试	测试完整
	循环语句是否测试完整，如是否有正常跳出循环的条件，检查是否会出现死循环，break 和 continue 语句是否使用正确	测试完整
可维护性检查	标号和子程序名是否符合代码的意义	符合
	是否使用了非通用的函数库	未使用
	对于非标准的库是否提供了源程序	提供
	对于重复出现的常量是否定义了宏	定义
	对于重复出现并完成同样单一功能的一段代码，是否用函数对其进行了封装	进行封装
	如使用技巧性编程，是否有详细解释说明	有
	错误或异常信息提示是否正确	正确
逻辑检查	代码是否正确地实现了设计功能	正确
	编码是否涉及了设计所规定以外的内容	未涉及
	每个循环是否执行正确的次数	正确
	输入参数的所有异常值是否全面测试	全面测试
	逻辑判断表达式是否符合程序设计	符合
软件多余物	是否存在可能执行不到的代码	不存在
	是否存在即使不执行也不影响程序功能的指令	不存在
	是否存在未引用的变量、标号和常量	不存在
	是否存在多余的程序单元	不存在

5.2 动态测试

5.2.1 逻辑测试

5.2.1.1 测试内容

主要分析代码的测试覆盖率并显示未覆盖的代码路径，由此发现未测试源代码中隐藏的缺陷。测试内容包括语句覆盖 SC（Statement Coverage）、判定覆盖 DC（Decision Coverage）、条件覆盖 CC（Condition Coverage）、条件判定组合覆盖 CDC（Condition/ Decision Coverage）、多条件判定覆盖 MCC（Multiple Condition Coverage）、修正条件判定覆盖 MC/DC（Modified Condition / Decision Coverage）。

5.2.1.2 测试方法

5.2.1.2.1 语句覆盖

设计足够的测试用例，使被测程序中每条语句至少执行一次。以下面一段程序为例：

```
int function1(bool a, bool b, bool c)
{
   int x;
   x = 0;
   if ( a && ( b || c ) )
       x = 1;
   return x;
}
```

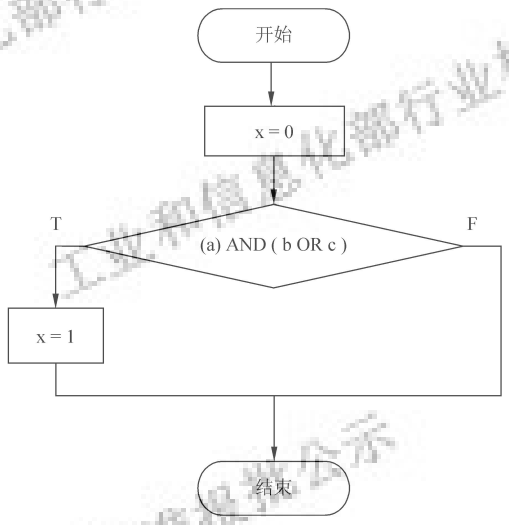

图 1 参考示例程序流程图

为了使上述程序中的每条语句都能够至少执行一次，可以构造以下测试用例：

a = T，b = T，c = T

从程序中的每条语句都得到执行这一点看，语句覆盖的方法能够比较全面地检验每一条语句，但是语句覆盖对程序执行逻辑的覆盖很低，这是其最严重的缺陷。

5.2.1.2.2 判定覆盖

设计足够的测试用例，使得程序中的每个判定至少都获得一次“真值”或“假值”。除了双值的判定语句外，还有多值判定语句，如 case 语句，因此判定覆盖更一般的含义是：使得每一个判定获得每一种可能的结果至少一次。

以图 1 所示代码为例，构造以下测试用例即可实现判定覆盖：

- a = T，b = T，c = T
- a = F，b = F，c = F

5.2.1.2.3 条件覆盖

设计足够的测试用例，使得每一判定语句中每个逻辑条件的可能值至少出现一次。

以图 1 所示代码为例，构造以下测试用例即可实现条件覆盖：

- a = F，b = T，c = F
- a = T，b = F，c = T

5.2.1.2.4 条件判定组合覆盖

设计足够的测试用例，使得判定中每个条件的所有可能（真/假）至少出现一次，并且每个判定本身的判定结果（真/假）也至少出现一次。

以图 1 所示代码为例，构造以下测试用例即可实现条件判定组合覆盖：

- a = T，b = T，c = T
- a = F，b = F，c = F

5.2.1.2.5 多条件判定覆盖

设计足够的测试用例，使得每个判定中条件的各种可能组合都至少出现一次。所以满足多条件覆盖的测试用例是一定满足判定覆盖、条件覆盖和条件判定组合覆盖的。

以图 1 所示代码为例，构造如表 4 的测试用例即可实现多条件判定覆盖。

表 4 示例程序多条件判定测试用例表

序号	a	b	c	a && (b ‖ c)
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

5.2.1.2.6 修正条件判定覆盖

设计足够多的测试用例，使之满足以下条件：首先，每一个程序模块的入口和出口点都要考虑至少要被调用一次，每个程序的判定到所有可能的结果值要至少转换一次；其次，程序的判定被分解为通过逻辑操作符（and、or）连接的 bool 条件，每个条件对于判定的结果值是独立的。

以图 1 所示代码为例，可以构造如表 5 的 8 个用例，在此基础上，按照 MC/DC 的要求选择需要的用例。

表 5 修正条件判定覆盖

序号	a	b	c	a && (b ‖ c)	a	b	c
1	T	T	T	T	5		
2	T	T	F	T	6	4	
3	T	F	T	T	7		4
4	T	F	F	F		2	3
5	F	T	T	F	1		
6	F	T	F	F	2		
7	F	F	T	F	3		
8	F	F	F	F			

从表 5 中可以看出，布尔变量 a 可以通过用例 1 和 5 达到 MCDC 的要求（用例 2 和 6 或用例 3 和 7 也可以满足相应要求），变量 b 可以通过用例 2 和 4 达到 MCDC 的要求，变量 c 可以通过用例 3 和 4 达到 MCDC 的要求，因此使用用例集{1，2，3，4，5}即可满足 MCDC 的要求。

5.2.2 函数（方法）执行性能测试

5.2.2.1 测试内容

函数（方法）执行性能测试主要测试函数（方法）的调用次数和执行时间。

5.2.2.2 测试方法

函数（方法）调用次数测试：通过在被测函数（方法）中插入执行计数的语句，以记录并输出被测函数（方法）的实际执行次数。

函数（方法）执行时间测试：通过在被测函数（方法）的入口与出口分别插入获取当前系统时间的函数，计算函数（方法）执行到出口的时间与入口时间的差值，以获得函数（方法）执行时间。

5.2.3 动态内存分析

5.2.3.1 测试内容

对于Linux操作系统与Windows操作系统的数控系统，动态内存分析测试内容见表6与表7。

表6 Linux操作系统动态内存分析测试项目

测试内容	描　　述
Page-in rate	每秒钟读入到物理内存中的页数
Page-out rate	每秒钟写入页面文件和从物理内存中删除的页数
Paging rate	每秒钟读入物理内存或写入页面文件中的页数

表7 Windows操作系统动态内存分析测试项目

对　　象	度　　量	描　　述
内存	Page Faults/sec	此值为处理器中的页面错误的计数。当进程引用特定的虚拟内存页，该页不在其在主内存的工作集当中时，将出现页面错误。如果某页位于待机列表中（因此它已经位于主内存中），或者它正在被共享该页的其他进程所使用，则页面错误不会导致该页从磁盘中提取出
	Pool Nonpaged Bytes	非分页池中的字节数，指可供操作系统组件完成指定任务后从其中获得空间的系统内存区域。非分页池页面不可以退出到分页文件中，它们自分配以来就始终位于主内存中
	Pages/sec	为解析内存对页面（引用时不在内存中）的引用而从磁盘读取的页数或写入磁盘的页数。这是“Pages Input/sec”和“Pages Output/sec”的和。此计数器中包括的页面流量代表着用于访问应用程序的文件数据的系统缓存。此值还包括传递到/来自非缓存映射内存文件的页数，如果关心内存压力过大问题（即系统失效）和可能产生的过多分页，则这是值得考虑的主要计数器

5.2.3.2 测试方法

通过内存监测工具或调用操作系统资源监视界面，查看内存在软件运行情况下的使用情况。

5.3 代码安全测试

5.3.1 测试内容

5.3.1.1 概述

代码安全测试的范围可以是以C、C++等开发语言编写的程序的全部代码，也可以是某个独立的模块或关键的流程模块的代码。测试内容包括但不限于本节内容。

5.3.1.2 API使用不当

软件使用API的方式与它的预期用途不符，使用陈旧的API，功能及安全将存在隐患；对API的适用场景理解不深刻，也会导致不良影响。

5.3.1.3 代码质量漏洞

代码不直接引入弱点和漏洞，但是软件并没有被认真开发或维护。如果一个软件表现出来的情况是复杂的、难以维护的、稳定性差等等，最大可能是代码质量差，代码中存在潜在的风险。

5.3.1.4 封装不充分

软件不能充分封装关键数据或功能漏洞。在此“封装”主要是指各系统/组件间的信任边界的建立，如果封装不充分，则会造成内部数据的泄露，把未经验证的数据误作为可信数据，以及他人保密信息的泄露等问题。

5.3.1.5 安全配置漏洞

对软件运行环境的合理配置是实现安全性的重要因素，程序运行环境通常都有损害安全性的不合理配置。良好的安全性需要软件和所在平台定义和部署了一套好的安全配置，并且所有的软件（包括应用程序使用的代码库）都应该实时更新。

5.3.1.6 异常处理漏洞

程序没有妥善处理在运行过程中发生的错误。异常处理不可用或没有正确实现，比如在软件抛出异常的时候没有捕获，很可能会泄露调试信息以及敏感数据，给攻击者披露程序的内部运作，提供细节信息。

5.3.1.7 数据处理漏洞

在数据处理功能方面的漏洞。软件对输入信息验证不正确，会接收到意想不到的输入，很可能导致改变控制流，任意控制资源，或执行任意代码。如果组件之间有交互，那么不适当的编码或转义可以让攻击者改变命令发送到另一个组件，插入恶意命令，组件的接收输出将执行错误的操作，或者解释数据不正确。

5.3.1.8 安全特性

软件安全漏洞，如身份验证、访问控制、保密、加密、权限管理。包括但不限于以下情况：

a) 使用一个空字符串作为密码，容易造成非授权访问；
b) 密码在配置文件中明文存储，容易造成密码泄露；
c) 密码长度、复杂度如果设置简单，也会被攻击者易于猜测，造成非授权访问；
d) 对保护资源的授权检查不当，会造成非授权访问。

5.3.1.9 时序状态

并发进程或线程环境中不当的时间和状态管理。在多进程系统中，任务之间切换非常快，可能多个进程都需要访问、控制某一资源，同步机制控制不好，则会产生错误的结果。

6 测试方法

6.1 代码选择

选择全部源代码进行测试时，需首先经代码编译，生成应用程序，确认应用程序功能内容无误。代码量较大时，一般选择部分源代码进行测试。对选定代码的测试结果仅对被测代码有效，不能作为评价全部源代码的依据。

6.2 测试工具准备

选择支持C、C++、JAVA、.NET等开发语言、内置安全代码规范、能够对代码自动地进行数据流、语义、结构、控制流、配置五个方面分析的测试工具。按照所选用的测试工具手册及技术文档的要求选择工具安装所依赖的硬件和软件，安装测试工具，启动测试工具自检，确认测试工具安装成功，运行正常。选择安全代码规范，按照测试需求配置工具。

6.3 代码安全扫描

将源代码按功能模块分类，以文件夹形式存放。采用源代码分析工具的静态扫描分析模式选取源代码目录，设置扫描深度、扫描类型等参数，进行自动扫描。

当代码中存在相互引用关系过于复杂、重复代码过多、程序质量低等问题时，工具扫描可能会出现中断，此时需与开发代码的分析人员一起分析代码，调整扫描参数，重新启动扫描或继续扫描。

扫描结束后，工具自动生成扫描报告，为人工审计提供参考。

6.4 人工审计

在扫描结束后，对扫描结果进行分析。分析工具扫描得到的漏洞是否为误报。之后对不是误报的漏洞进行进一步分析，分析其被利用的可能性，进而确定漏洞的风险级别。

7 判定标准

7.1 安全问题风险级别定义

安全问题风险级别定义见表8

表 8 风险级别及定义

序号	风险级别	定义
1	高风险	被利用可造成软件一定范围受到影响，如攻击者可以远程控制信息系统的重要资源，获取应用系统管理权限等
2	中风险	被利用可造成软件小范围受到影响，如攻击者可以获取浏览软件数据，可以获得用户名、口令等敏感信息，潜在可能导致高风险的漏洞等
3	低风险	被利用可获取软件一些服务信息，如版本号、操作系统类型等

7.2 源代码测评结果判定

测评结果中存在高风险，测评结果为“不通过” 。

测评结果中存在中风险，其占源代码总行数低于每千行代码5个，测评结果为“通过”，源代码安全性较好；否则为“不通过”。

测评结果中只存在低风险，测评结果为“通过”，源代码安全性较好。

参考文献

[1] GJB 5369-2005 航天型号软件C语言安全子集
[2] GB/T 28169-2011 嵌入式软件 C语言编码规范
[3] GB/T 20984-2007 信息安全技术 信息安全风险评估规范
[4] GB/T 22239-2008 信息安全技术 信息系统安全等级保护基本要求