泽众 CodeAnalyzer 代码审查软件

技术白皮书
Version2.0

上海泽众软件科技有限公司

2018 年 1 月

目录

1.系统概述

1.1.系统定位

Code Analyzer 2.0 是上海泽众软件科技有限公司开发的，拥有自主产权的，脱离任何编译器的代码静态分析工具，缩写 CA。

CA 能够用来对 C 源代码和 JAVA 源代码进行扫描并分析分析，根据预先定义好的代码规范对代码进行规范化检查，找出代码中不合理、不符合规范定义的部分并生成分析报告。开发工程师可以通过报告总结分析问题，使代码合理化、规范化，从而提高程序质量。

另外，可以利用 CA 在代码审计系统中充当代码合规检查的角色。

1.2.适用范围

CA 适用于软件开发过程中的编码和单元测试阶段，可以单独使用，也可以与版本管理工具、集成开发环境等工具来集成，实现对提交 C 代码的自动扫描和分析，预先发现程序中的不合规代码、软件漏洞、后门。

CA 使在编码阶段就能够检查程序员的代码，提早避免缺陷的产生。CA 能够大幅度的提升软件的代码的规范性，减少缺陷的产生。

另外，在测试体系中，可以使用 CA 评估开发工程师的工作量与工作质量，即代码审计。

代码审计的内容有：

1）代码修改行数（与上一个版本进行比较）；

2）代码修改的有效性（防止反复修改）；

3）记录分析结果到数据库。

开发工程师工作量评估的依据为：

——修改代码的行数（包括增加、修改、删除）

开发工程师工作质量评估的依据为：

——代码质量

1）根据总行制定的《代码编写规范》来设定代码规则；

2）给代码进行评分。

代码管理

1）定义代码质量的标准；

2）在代码（check in）到版本库时进行自动分析：

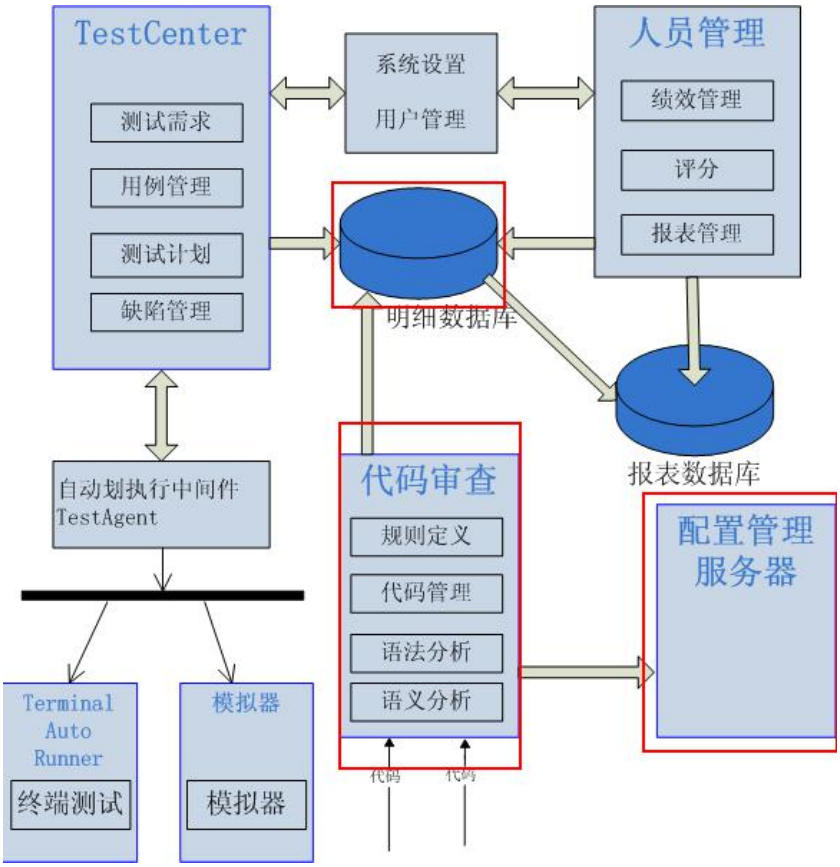
➢分析代码的编码规范适合程度

➢对代码进行评估

➢给出修改建议或者打回（不允许 check in）

➢记录到数据库

如下图所示，是 CA 软件运用到代码审计系统的示意图：



系统中红框的部分提供了代码审计的功能，这部分又 3 个组件组成：

1)代码审查模块，由规则定义、代码管理、语法分析、语义分析等子模块组成，用于接收用户源代码文件作为输入，经过语法分析得到分析日志，并将分析结果转存到明细数据库。审计人员可到明细数据库查询审计结果。代码审查模块可以基于 CA 软件产品实现。

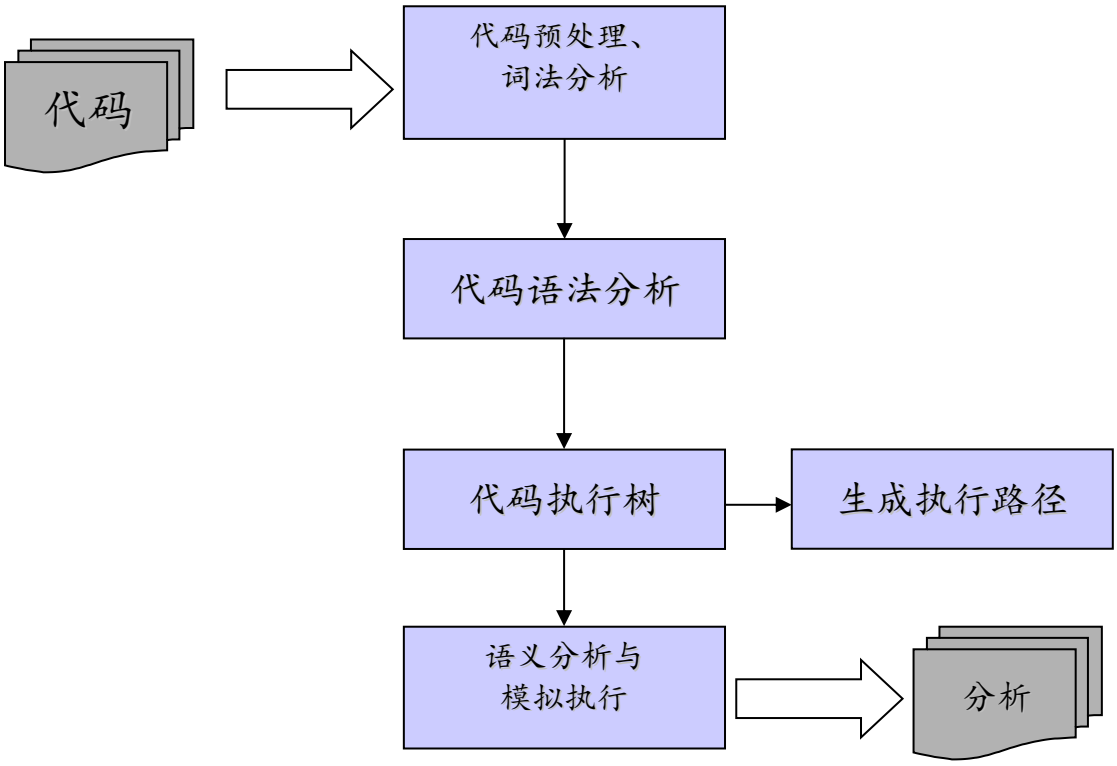
2)明细数据库。用于存放代码审计结果，允许用户通过客户端对代码审计结果进行查询。可以开发辅助模块将 CA 返回的审计结果转存到明细数据库中。

3)配置管理服务器。用于存放通过代码审计的源代码。

用户可以通过不同的方式访问和分析代码审计的结果。例如，通过报表模块汇总明细数据库中的代码审计结果，形成审计报告。人员管理模块可以读取报表模块数据库对人员绩效进行评分和评估。

2.系统构架

CA 的工作原理和流程如下图：



CA 是采用编译的方法来处理程序代码，它分成：词法、语法、语义三个层面。

CA 对代码进行分析，生成语法树，然后对语法树进行遍历，生成代码执行树和执行路径。然后对每个路径进行规则检查。

所有的功能体现在规则上。

规则通过配置的 XML 文件来配置和扩展。

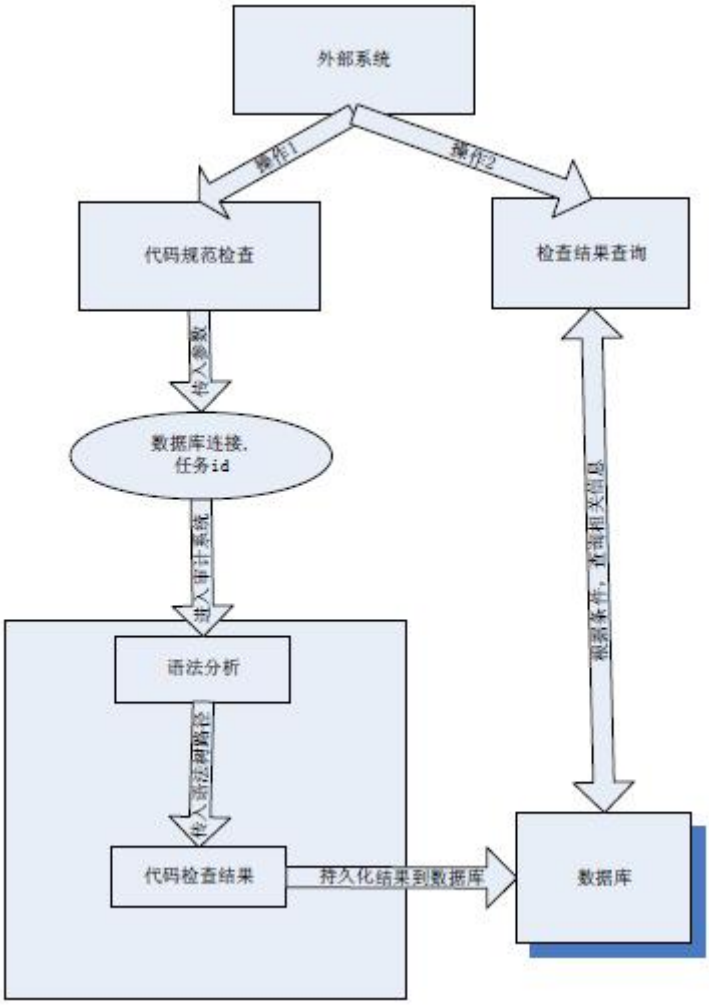
CA 不依赖于任何编译器，而是在 CA 内部实现了一个编译程序，通过对代码的编译来实现。规则分成不同的级别：词法级别、语法级别和语义级别。不同的级别在不同的处理阶段被处理。

3.对外编程接口 API

3.1.对外编程 API 结构

为了向外系统提供业务功能，将代码审计模块集成到其它代码管理系统或项目管理系统中去，CA 提供了对外编程接口 API。

如下图所示为对外 API 的结构示意图：



3.2.API 定义

1)JAR 文件

API 库	codeChecker.jar	用于分析文法树，返回分析结果，打印分析结果信息等
依赖库	autoCodeScanner.jar	用于解析类文件，创建文法树
	dom4j-1.6.1.jar	用于分析文法树时，对 xml 文件的解析
	jdom.jar	用于创建文法树时，输出 xml 文件
	log4j.jar	用于打印文法树分析的结果信息日志
	parser.jar	解析类文件的核心JAR 包
	en.mdb	英文词库文件
动态可库	ccb_dll.dll	C 解析核心动态库
	iconv.dll	C 解析核心动态库
	libxml2.dll	C 解析核心动态库
	zlib1.dll	C 解析核心动态库
	Replace.dll	关键性替换核心动态库

注：该JAR 文件使用JAVA5 编译，使用时需将所有的JAR 文件全部放在CLASSPATH 当中，其中

codeChecker.jar 需依赖于autoCodeScanner.jar ， dom4j-1.6.1.jar ，log4j.jar;autoCodeScanner.jar 需依赖于jdom.jar 和parser.jar。

、并将en.mdb 文件放入项目中任意目录，调用的时候需要配置该路径。所有动态库放在WEB-INF 下的dll 文件夹中。

注2:

C 代码解析接口不变,但数据库已有更改(详情见数据库文档说明)

C 代码文件的编码格式必须为UTF-8 格式

2)Class1 CodeCheckConsole

包：com.spasvo.gui
类：public CodeCheckConsole
注： 此类在JAR 文件codeChecker.jar 中， 是开放给客户代码调用的 codeChecher_report_analyser 的接口。此类是一个工具类，提供得到单例的 *getInstance()*方法得到实例。

获取操作对象实例
static CodeCheckConsole getInstance();
param:
    <no params>
returns:
    CodeCheckConsole : 操作对象实例

初始化检查器配置的对外方法
Void configure(File dictionnary ,File grammerPath);
Param:
    Dictionnary：单词库的路径
    grammerPath：生成的语法树放置路径
注：1.检查单词前必须调用此方法法对检查器进行初始化。
    2.若词库文件不存在或配置错误，检查器认为所有单词都正确。

得到检查结果的对外方法
List<Rules> getResult(String srcFilePath);
param:
    srcFilePath: 要检查的类文件的绝对路径
returns:
    List<Rules>: 检查结果的集合

List<Rules> getResult(File srcFile);
param:
    srcFile:要检查的类文件实例
returns:
    List<Rules>: 规则检查结果信息

设置标志的对外方法(获取结果后,是否删除语法树的标志位true = 删除,默认 false)
void setDeleteGrammerTempFile(boolean deleteGrammerTempFile);
param:
    deleteGrammerTempFile:是否删除true = 删除,false= 不删除;
return:
    No returns

得到标志的对外方法(获取是否删除语法树的标志位)
boolean getDeleteGrammerTempFile();
prama:
    No params;
return:
    boolean : 获取是否删除语法树的标志位

根据任务编号解析文件并保存到数据库中的对位方法:
void analyserFileAndSaveResultToDB(Connection conn,String taskId);
param:
    conn : 保存结果的目标数据库的连接
    taskId:要检查的任务的编号Id
return:
    No returns
注:考虑到用户可能会继续对数据库做其他操作,在该方法中,并未将数据库连接关闭.所以,调用方法后,连接仍然可用;数据库连接必须要用户手动编码关闭.

3)Class2 Rules

包：com.spasvo.bean
类：public Rules
注：此类在 JAR 文件 codeChecker.jar 中，用于保存检查规则信息和规则对应的检查结果信息

属性
String xmlPath:文法树的路径;
String srcFilePath:被解析的类文件的路径;
String name:规则名称;
String formula:规则详细内容;
String description:规则详细描述;
List<ResultBean> resultBeanList:规则验证结果的信息集合;
Integer grade: 规则的严重程度; value: 1.为必须遵守的规则，2.为建议遵守的规则

构造方法
public Rules(String xmlPath,String srcFilePath);
param:
xmlPath:文法树的路径
srcFilePath:被解析的类文件的绝对路径
returns：<no returns>

方法
提供所有属性的 Getter 和 Setter 方法

4)Class3 ResultBean

包：com.spasvo.bean 类：public ResultBean 注：此类在 JAR 文件 codeChecker.jar 中，用于保存具体规则的验证的结果信息

返回结果类型常量 Public static final Integer TYPE_RIGHT: 正确 Public static final Integer TYPE_ERROR：错误

属性 String row ： 被检查的代码行 String col ： 被检查的代码的开始字母的列 Integer resultType ： 返回结果类型（对应常量值） String detail：要检查的内容

构造方法 默认构造方法

方法 提供所有属性的 Getter 和 Setter 方法

4.系统基本功能

4.1.定义规则

标准化的规则定义在 CA 的引擎中。

CA 支持众多的 C/C++静态测试规范，支持 GJB5369 测试规范。

词法规则：CA 支持英语的单词表，变量命名的定义来自于词表检查。

语法规则：CA 通过标准化的语法模版来处理语义规则。

语义规则：CA 通过调用标准化的处理程序来分析定义的规则。

下表罗列出 CA 软件支持的语法规则，其中 1.0 版本实现的规格在“是否实现”一列中标识为“是”。

章节名称	规则定义	规则	是否实现
命名规则			
	命名原则	标识符应当直观且可以拼读	
		标识符的长度应当符合“min-length && max-information”原则	
		命名规则尽量与所采用的操作系统或开发工具的风格保持一致	
		程序中不要出现仅靠大小写区分的相似的标识符	
		程序中不要出现标识符完全相同的局部变量和全局变量	是
		变量的名字应当使用“名词”或者“形容词＋名词”	
		全局函数的名字应当使用“动词”或者“动词＋名词”（动宾词组）。	
		用正确的反义词组命名具有互斥意义的变量或相反动作的函数等	
		尽量避免名字中出现数字编号	是
		禁止使用汉语拼音来命名	
应用程序的命名		“系统简称”+模块名称	
子模块的命名		每个子模块的名字应该由描述模块功能的 1-3 以单词组成。每个单词的首字母应大写	
变量的命名		可以用多个英文单词拼写而成,每个英文单词的首字母要大写，其中英文单词有缩写的可用缩写；	

		变量的前缀表示该变量的类型	
		对于作用域跨越 10 行以上的变量名称不能少于 4 个字符	
		除循环变量，累加变量外不得使用 I、j、k 等名称的变量。	
		对于全局变量以加前缀“g_”来区分	是
常量的命名		常量所有的字母均为大写。并且单词之间使用下划线”_”隔开	是
函数/过程的命名		函数/过程名称应该尽量使用能够表达函数功能的英文名称	
		全局函数/过程名称以“g_”前缀开始	是
接口命名		接口名称要以大写字母开头。如果接口包含多个单词，每个单词的首字母大写，其他字母小写	是
类的命名		类名称要以大写字母开头	是
		类名称如果包含多个单词，每个单词的首字母要大写，其他字母小写	
		类名称不能出现下划线	是
方法的命名		方法名称以小写字母开头	是
		方法名称如果包含多个单词，除了第一个单词外，每个单词的首字母大写，其它字母小写	
程序版式			
头文件的结构		头文件由三部分内容组成:头文件开头处的版权和版本声明；预处理块；函数和类结构声明等	
		1． 为了防止头文件被重复引用，应当用 ifndef/define/endif 结构	
		2. 用#include < filename.h> 格式来引用标准库的头文件	
		3. 用#include “filename.h” 格式来引用非标准库的头文件	

		4. 头文件中只存放“声明”而不存放“定义”	是
		5. 不提倡使用全局变量，尽量不要在头文件中出现象 extern int value	是
定义文件的结构		定义文件有三部分内容:定义文件开头处的版权和版本声明；对一些头文件的引用；程序的实现体	
空行		1. 在每个类声明之后、每个函数定义结束之后都要加空行	
		对独立的程序块之间、变量说明之后必须加空行	
		2. 在一个函数体内，逻辑上密切相关的语句之间不加空行	
代码行		1. 一行代码只做一件事情	是
		2. if、for、while、do 等语句自占一行，执行语句不得紧跟其后。	是
		3. 尽可能在定义变量的同时初始化该变量	
代码行内的空格		1. 关键字之后要留空格	
		2. 函数名之后不要留空格，紧跟左括号	
		3. ‘（’向后紧跟，‘）’、‘，’、‘;’向前紧跟	是
		，’之后要留空格	是
		5. 赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=” “>=”、“<=”、“+”、“*”、“%”、“&&”、“ ||”、“<<”，“ˆ”等二元操作符的前后应当加空格	是
		6. 一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格	是
		7. 象“［］”、“.”、“->”这类操作符前后不加空格	是
		8. 对于表达式比较长的 for 语句和 if 语句，为了	

		紧凑起见可以适当地去掉一些空格	
对齐			
		1. 程序应采用缩进风格编写，每层缩进使用一个制表位（TAB），类定义、方法都应顶格书写	是
		2. 程序的分界符‘{’和‘}’应独占一行并且位于同一列，同时与引用它们的语句左对齐	是
		3. {}之内的代码块在‘{’右边数格处左对齐，不能跟在上一行的行末。	是
		4. 一个变量定义占一行，一个语句占一行	是
长行拆分			
		1. 代码行最大长度宜控制在 80 字符以内。对于较长的语句（>80 字符）要分成多行书写	是
		2. 长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读	
修饰符的位置			
		1. 将修饰符 * 和 &紧靠变量名	
注释			
		1. C 语言的注释符为“/* … */”。C++语言中，程序块的注释常采用“/*… */”，行注释一般采用“//… ”。注释通常用于：版本、版权声明；函数接口说明；重要的代码行或段落提	
		2. 注释是对代码的“提示”，而不是文档。程序中的注释不可喧宾夺主，注释太多了会让人眼花缭乱	
		3. 如果代码本来就是清楚的，则不必加注释	
		4. 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除	
		5. 注释应当准确、易懂，防止注释有二义性。错误的注释不但无益反而有害。	

		6. 尽量避免在注释中使用缩写,特别是不常用缩写。	
		7. 注释的位置应与被描述的代码相邻，可以放在代码的上方或右方，不可放在下方。	
		8. 当代码比较长，特别是有多重嵌套时，应当在一些段落的结束处加注释，便于阅读	
源程序头的注释和规范	源程序头部说明	FileName: Copy Right: System: Module: Function: See also: Author: Create Date: 本程序的外部名字(如 *.prg, *.cpp) xxx 公司 版权所有 版本信息 本文件所在的系统或工程的名字 本文件所在的功能模块名称 简要说明本程序的功能 相关详细设计文档号 编码人员 创建日期	
	源程序版本说明	Editor： Version： Edit Date： 修改人员 版本号 修改日期	
函数头的注释和规范		Name: Function:	

		Input: Output: Return: Syntax: Env: Calling: 函数名称 简述函数或过程的功能 [参数 1] — [说明…] [参数 2] — [说明…] [参数 1] — [说明…] [参数 2] — [说明…] [返回码 1] — [说明…] [返回码 2] — [说明…] 调用语法(可选) 环境要求和影响(可选的) 被调用的函数(可选的)	
函数体及其他代码的注释		1. 注释要清晰，与代码保持一致，避免没有价值的注释；	
		2. 保持注释与代码完全一致	
		3. 对于全局标识符（函数、全局变量、常量定义等）必须要加注释	
		4. UNIX 下 C 程序或存储过程注释统一用/* 和*/，不允许用//；	
		5. 循环语句要有注释。	
		6. 分支语句要有注释。	
		7. 数据库语句要有注释。	

		8. 主要变量（结构、联合、类或对象）定义或引用时，注释能反映其含义。	
		9. 处理过程的每个阶段都有相关注释说明。	
		10.在典型或特殊算法前要有注释	
		11. 在代码的关键部分要有注释	
		12. 在逻辑性较强的代码前要有注释	
		13. 注释可以与语句在同一行，也可以在上行	
		14. 注释行数（不包括程序头和函数头说明部份）应占总行的 1/5 到 1/3	
		15. 对函数中某处细节进行修改必须留下注释，其格式如下： 〈程序源代码〉 /*［修改原因］［修改者］［日期］*/	
变量注释		直接在变量后面注明变量的用途和取值约定	
类型定义注释		指类和记录等等定义的注释。在注释中标明定义的用途	
区的注释		同一个类的成员方法要求排列在一起，共同协作而实现同一个功能的函数和过程要求排列在一起	
类		1. 将 private 类型的数据写在前面，而将 public 类型的函数写在后面	
		2. 将 public 类型的函数写在前面，而将 private 类型的数据写在后面	
		3. 建议采用“以行为为中心”的书写方式，即首先考虑类应该提供什么样的函数	
编码过程中的约定			
表达式和基本语句			
运算符的优先		1. 如果代码行中的运算符比较多，用括号确定表达	

级		式的操作顺序，避免使用默认的优先级。为了防止产生歧义并提高可读性，应当用括号确定表达式的操作顺序。	
复合表达式		1. 不要编写太复杂的复合表达式。	
		2. 不要有多用途的复合表达式	
		3. 不要把程序中的复合表达式与“真正的数学表达式”混淆	
if 语句		1. 布尔变量与零值比较	
		2. 整型变量与零值比较	
		3. 浮点变量与零值比较	
		4. 指针变量与零值比较	
循环语句的效率		1. 在多重循环中，如果有可能，应当将最长的循环放在最内层，最短的循环放在最外层，以减少 CPU 跨切循环层的次数。	
		2. 如果循环体内存在逻辑判断，并且循环次数很大，宜将逻辑判断移到循环体的外面	
语句的循环控制变量			
		1. 不可在 for 循环体内修改循环变量，防止 for 循环失去控制	
		2. 建议 for 语句的循环控制变量的取值采用“半开半闭区间”写法	
switch 语句		1. 每个 case 语句的结尾不要忘了加 break，否则将导致多个分支重叠（除非有意使多个分支重叠）	
		2. 不要忘记最后那个 default 分支。即使程序真的不需要 default 处理，也应该保留语句 default：break	是
goto 语句		1. 禁止 GOTO 语句	是
strcpy 语句		1. 对应用传递的输入输出参数，禁止使用 strcpy,	

		strlen, strcat, strcmp 等相应的函数操作输入输出参数，尽量使用 strncpy, memcpy 等含有处理长度参数的函数进行处理	
常量		1. 需要对外公开的常量放在头文件中，不需要对外公开的常量放在定义文件的头部。为便于管理，可以把不同模块的常量集中存放在一个公共的头文件中	
		2. 如果某一常量与其它常量密切相关，应在定义中包含这种关系，而不应给出一些孤立的值	
函数			
参数			
		1. 参数的书写要完整，不要贪图省事只写参数的类型而省略参数名字。如果函数没有参数，则用 void 填充	是
		2. 参数命名要恰当，顺序要合理。	
		3. 如果参数是指针，且仅作输入用，则应在类型前加 const，以防止该指针在函数体内被意外修改	
		4. 如果输入参数以值传递的方式传递对象，则宜改用“const & ”方式来传递，这样可以省去临时对象的构造和析构过程，从而提高效率	
		5. 在使用应用传递的输入输出参数之前，必须对参数进行合法性检查，保证代码执行使用参数的安全性。比如，应用的一个输出参数地址为 NULL，如果处理之前不检查参数的合法性，那么将导致一个内存错误；如果检查合法性，就不会造成代码执行时出现问题	
		6. 参数缺省值只能出现在函数的声明中，而不能出现在定义体中	
		7. 如果函数有多个参数，参数只能从后向前挨个儿缺省，否则将导致函数调用语句怪模怪样	

		8. 如果若函数中的参数较长，则要进行适当的划分	
		9. 避免函数有太多的参数，参数个数尽量控制在 5 个以内。如果参数太多，在使用时容易将参数类型或顺序搞错	是
		10. 尽量不要使用类型和数目不确定的参数	是
		11. 形参的排序风格	
返回值			
		1. 不要省略返回值的类型，如果函数没有返回值，那么应声明为 void 类型	
		2. 函数名字与返回值类型在语义上不可冲突	
		3. 不要将正常值和错误标志混在一起返回。正常值用输出参数获得，而错误标志用 return 语句返回	
		4. 给以“指针传递”方式的函数返回值加 const 修饰，那么函数返回值（即指针）的内容不能被修改，该返回值只能被赋给加 const 修饰的同类型指针	
		5. 函数返回值采用“值传递方式”，对于有不同状态的返回值，建议用 long 型的返回值，0 为成功。由于函数会把返回值复制到外部临时的存储单元中，加 const 修饰没有任何价值	
		6. 函数返回值采用“引用传递”的场合并不多，这种方式一般只出现在类的赋值函数中,目的是为了实现链式表达	
函数内部实现		1. 在函数体的“入口处”，对参数的有效性进行检查	
		2. 在函数体的“出口处”，对 return 语句的正确性和效率进行检查。	
		3. return 语句不可返回指向“栈内存”的“指针”或者“引用”，因为该内存在函数体结束时被自动销毁	

		4. 要搞清楚返回的究竟是“值”、“指针”还是“引用”	
		5. 如果函数返回值是一个对象，要考虑 return 语句的效率	
其它			
		1. 函数的功能要单一，不要设计多用途的函数	
		2. 函数体的规模要小，尽量控制在 50 行代码之内	是
		3. 尽量避免函数带有“记忆”功能。相同的输入应当产生相同的输出。带有“记忆”功能的函数，其行为可能是不可预测的,因为它的行为可能取决于某种“记忆状态”	
		4. 不仅要检查输入参数的有效性，还要检查通过其它途径进入函数体内的变量的有效性，例如全局变量、文件句柄等	
		5. 用于出错处理的返回值一定要清楚，让使用者不容易忽视或误解错误情况	
重载和内联			
重载		1. 重载函数中的参数不同（包括类型、顺序不同），才是重载函数，而仅仅返回值不同则不行	
		2. 当心隐式类型转换导致重载函数产生二义性，数字本身没有类型,将数字当作参数时将自动进行类型转换（称为隐式类型转换）	
		3. 成员函数的重载、覆盖（override）与隐藏很容易混淆，注意区分	
		4. 注意如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无 virtual 关键字，基类的函数将被隐藏（注意别与重载混淆）	
		5. 注意如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有 virtual 关键字。	

		此时，基类的函数被隐藏（注意别与覆盖混淆）	
内联		1．尽量用内联取代宏代码,提高函数的执行效率(速度）。	
		2． 关键字 inline 必须与函数定义体放在一起才能使函数成为内联，仅将 inline 放在函数声明前面不起任何作用	
		3． 如果函数体内的代码比较长或函数体内出现循环，则不宜使用内联	
内存管理		1． 用 malloc 或 new 申请内存之后，应该立即检查指针值是否为 NULL。防止使用指针值为 NULL 的内存	
		2． 不要忘记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。	
		3．避免数组或指针的下标越界,特别要当心发生“多 1”或者“少 1”操作。	
		4．动态内存的申请与释放必须配对,防止内存泄漏。	
		5． 用 free 或 delete 释放了内存之后，立即将指针设置为 NULL，防止产生“野指针”。	
		6． 内存分配原则是必须先分配系统资源，才能分配函数内部需要的资源；相反，必须首先释放函数内部分配的资源，再释放系统资源	
类的构造函数、析构函数、成员函数与赋值函数		1． “缺省的拷贝构造函数”和“缺省的赋值函数”均采用“位拷贝”而非“值拷贝”的方式来实现，若类中含有指针变量，不能采用缺省的方式	
		2． 如果类存在继承关系，派生类必须在其初始化表里调用基类的构造函数。	
		3． 类的 const 常量只能在初始化表里被初始化，因为它不能在函数体内用赋值的方式来初始化	
		4．非内部数据类型的成员对象采用初始化表的方式	

		初始化较好	
		5. 拷贝构造函数和赋值函数非常容易混淆，常导致错写、错用。拷贝构造函数是在对象被创建时调用的，而赋值函数只能被已经存在了的对象调用。	
		6. 任何不会修改数据成员的函数都应该声明为 const 类型	
类的继承和组合		1. 如果类 A 和类 B 毫不相关，不可以为了使 B 的功能更多些而让 B 继承 A 的功能和属性	
		2. 若在逻辑上 B 是 A 的“一种情况”，则允许 B 继承 A 的功能和属性	
		3. 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	
一些有益的建议			
关于效率			
其他		1. 当心那些视觉上不易分辨的操作符发生书写错误。我们经常会把“==”误写成“=”，象“||”、“&&”、“<=”、“>=”这类符号也很容易发生“丢失”失误。然而编译器却不一定能自动指出这类错误	
		2. 变量（指针、数组）被创建之后应当及时把它们初始化，以防止把未被初始化的变量当成右值使用。	是
		3. 当心变量的初值、缺省值错误，或者精度不够。	
		4. 当心数据类型转换发生错误。尽量使用显式的数据类型转换，避免让编译器轻悄悄地进行隐式的数据类型转换	
		5. 当心变量发生上溢或下溢，数组的下标越界	
		6. 当心忘记编写错误处理程序，当心错误处理程序本身有误	

		7. 当心文件 I/O 有错误	
		8. 避免编写技巧性很高代码。	
		9. 不要设计面面俱到、非常灵活的数据结构	
		10. 如果原有的代码质量比较好，尽量复用它。但是不要修补很差劲的代码，应当重新编写	
		11. 尽量使用标准库函数，不要“发明”已经存在的库函数	
		12. 尽量不要使用与具体硬件或软件环境关系密切的变量	
		13. 把编译器的选择项设置为最严格状态	
错误和异常处理规范		错误代码长度固定为四个字符	
		提示信息长度不定，但最长不超过 60 个字符	
出错类型定义约定		出错类型分为错误、警告、提示等三类信息，分别用 E、W、I 开头；错误代码统一用宏描述，并且放在一个头文件中	
异常的捕获			
异常和错误的处理			
VC 的规则			

4.2.静态分析

支持 GUI 模式的使用方法和命令行方式的使用方法。

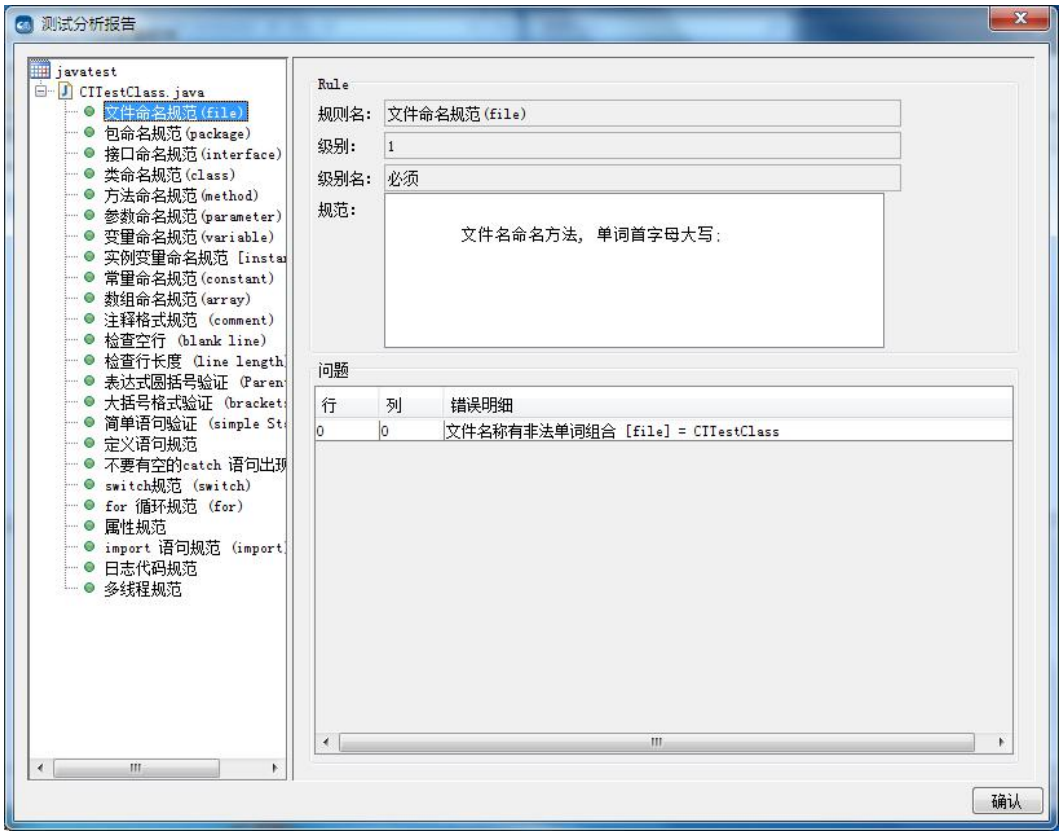
命令行方式：可以在批处理文件中来调用 CA 的命令，通过命令来指定需要扫描的程序文件。

GUI 方式：用户通过打开 GUI 界面，通过 GUI 交互的方式来扫描程序文件或者目录。

4.3.查看报告

分析报告可以存放在指定的日志文件中。

使用 GUI 的用户，可以直接在 GUI 中查看日志。



提供图形化的测试报告，并且可以输出成多种格式，包括 PDF 和 WORD

4.4.规则扩展

CA 允许用户定义自己的规则。

例如，用户可以要求文件注释中包括特殊的名字，如公司名。

用户通过配置扩展 XML 文件来扩展规则。对于词法、语法规则，用户可以通过配置的方法来实现；对于复杂的语义规则，用户可以通过定义指定的 java class 来处理。

5.厂商支持能力

泽众 CodeAnalyzer 代码审查软件，我们通过在线 QQ、微信、电话、电子邮件为您提供支持与服务，您也可访问我们的网站 http://www.spasvo.com/寻求帮助；为保证服务质量，确保有效地解决用户的问题，保障用户的项目实施进度，技术支持仅向授权用户和授权试用用户提供。请您在联系泽众技术支持时，告知您的单位名称和服务代码。

技术支持

电话：021-60725088-8007

传真：021-60725088-8017

电子邮件：support@spasvo.com

QQ：1404189128

泽众微信公众号

产品服务

有关培训、产品购买及试用授权方法的问题，请与销售代表联系，或联系泽众咨询热线.

电话：021-60725088-8006

传真：021-60725088-8017

电子邮件: sales@spasvo.com

提供完备的用户手册，管理员使用手册，系统技术手册并再系统升级后及时修改更新服务。

厂商能够根据在实际应用中的问题，迅速给予解答（2 小时内），并给出解决方案(48 小时内)。