Code Analyzer

技术白皮书
Version1.0

上海泽众软件科技有限公司

2014 年 3 月

1. 概述

1.1. 背景

静态分析软件，通过对源代码的扫描分析来实现：编码规则检查、潜在问题检查、安全规范检查。通过检查，能够发现源代码的问题，在开发阶段提升代码的质量，减少程序缺陷。

在源代码检查过程中，采用已有标准规则和用户自定义规则来实现源代码的扫描分析。

源代码检查，最好采用“自动化”的方式进行，减少程序员的开发工作，或者是“安静”的执行，不需要增加程序员额外的工作量，但是可以起到告警等目标。

静态分析，也是白盒测试的一个种类，最佳的方式是能够跟测试管理平台集成，并且跟踪管理静态分析工具产生的问题和缺陷。

1.2. 工具说明

Code Analyzer 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软件产品，可缩写为 CA。

CA 运行在 windows 平台上，支持 windows XP、windows 2003 等系统。

CA 的工作方式是编译处理之后的规则验证，而不是直接面向源代码的规则验证。这样的优势是漏报率大幅度降低：编译程序已经把基本的代码转换为语法树，能够更精确的进行规则匹配。

2. Code Analyzer 产品介绍

2.1. 产品概述

Code Analyzer 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软件产品，可缩写为 CA。

CA 目前包括 C 和 JAVA 两个版本分别用来对 C 源代码和 JAVA 源代码进行静态分析。用户通过使用 CA，根据预先定义好的代码规范对代码进行规范化检查，找出代码中不符合规范定义和不合理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。

CA 通过词法分析和语法分析实现了代码规范性检查，然后通过语义分析实现了代码潜在错误的检查，用户只需要将待检查的源文件导入到 CA，就可以实现一键分析。首先，CA 通过大量内建规则对源代码进行检查；其次，用户可以通过自定义代码规则轻松地实现客户化，使 CA 变成自己专有的规则检查器。CA 分析潜在错误的特性可以从实质上帮助客户提高代码质量，使客户代码更加健壮，它是比编译器更加严格的检查器，可以检测出没有释放

的自由内存，没有初始化并被使用的变量以及没有出口的循环等代码安全漏洞。

CA 可以对源代码进行逆向工程，通过对源代码进行解析，得到代码的控制流程图，用户可以对流程图进行代码走查，实现代码级的测试覆盖。在此基础之上，可以通过 CA 轻松实现 XUnit 单元测试模块的构建。

CA 可以生成多样化的分析报告。首先，可以控制报告的输出形式，将分析结果返回到日志文件中；或者，将分析结果返回到数据库表中，为客户提供进一步处理的资料。其次，CA 可以生成错误报告，错误统计分析报告，代码覆盖流程图等多种有效地测试报告，满足客户对测试分析数据的各种需求。

CA 提供了多样化的用户接口：图形用户界面（GUI），命令行，外部接口（DLL）。用户可以根据自己的需要，通过图形界面将源代码逐个导入到 CA 里进行分析，也可以通过 SHELL 或者批处理命令 BAT 来调用命令行，另外，如果用户可以利用 CA 提供的对外接口 API 实现在代码审计系统中进行代码规范性检查，通过代码符合规范的程度对程序员编写的代码评分。

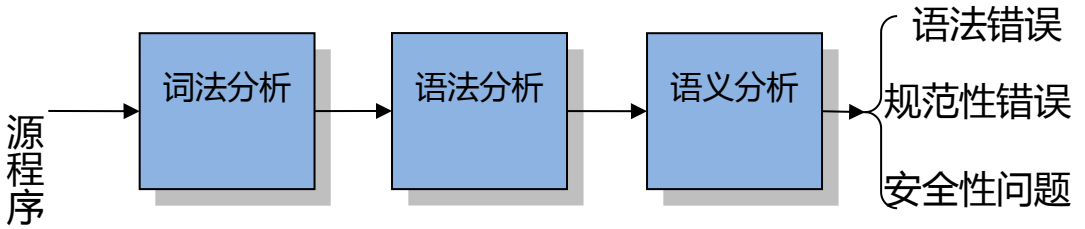
通过 CA 对外接口 API，可以轻松实现 CA 和配置管理工具 SVN 的集成。客户在编写好代码，并将代码 SUBMIT 到 SVN 服务器端前，调用 CA 对即将 SUBMIT 的源代码进行检查，从而实现代码质量控制。

2.2. 代码支持

目前，CA 可以支持 C 语言和 JAVA 语言两种编程语言创建的源程序的静态分析。它们分别是通过 CA 的两个版本实现的：Code Analyzer for C 和 Code Analyzer for JAVA。

2.3. 词法分析、语法分析和语义分析

首先，CA 作为静态分析软件测试工具，可以读取并分析源程序，然后判断读取的源程序是否符合内建的规则，或者用户自定义规则。这个过程是通过：词法分析、语法分析和语义分析三个连续的步骤完成的。分析的原理可以由下图说明：



接下来将对三个阶段 CA 完成的工作进行简要说明：

1. 词法分析阶段和语法分析阶段是连续的两个处理过程，CA 通过词法分析识别到源代码中的语法元素，通过语法分析验证源代码语法的正确性（每一个语法元素必须满足一定的文法规则，例如：

   JAVA 中常见的循环语句： for (init_part; condition; increment) statement
2. 语义分析是相对独立的过程，这个过程中将对语法分析得到的合法元素的语义进行合理化分析，例如：

   int i;

   printf(“%d”, i+1);

上面的代码片段存在一个错误，未对声明的整型变量 i 付初值便开始使用变量进行数值计算。这个代码从语法上讲是正确的，但是却存在不合法的语义。除此之外，语义分析还可以用于检查形如：malloc、realloc 等方法获得的自由内存在生命周期结束后有没有对应的 free 动作。

2.4. 支持规则（内建、自定义）

标准化的规则定义在 CA 的引擎中，这些规则成为 CA 的内建规则。CA 支持众多的 C 语言编码规范。

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
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
		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
		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
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
函数/过程的命		函数/过程名称应该尽量使用能够表达函数功能的英文名

名		称
		全局函数/过程名称以“g_”前缀开始
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
类的命名		类名称要以大写字母开头
		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
方法的命名		方法名称以小写字母开头
		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”
		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
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
		2．　if、for、while、do 等语句自占一行，执行语句不得紧跟其后。
		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
		，’之后要留空格
		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“=”、“+=” “>=”、“<=”、“+”、“*”、“%”、“&&”、“ ||”、“<<”，“^”等二元操作符的前后应当加空格

		6. 一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格
		7. 象“［］”、“.”、“->”这类操作符前后不加空格
		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
		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
		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。
		4. 一个变量定义占一行，一个语句占一行
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
		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		
运算符的优先级		1. 如果代码行中的运算符比较多，用括号确定表达式的操作顺序，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。

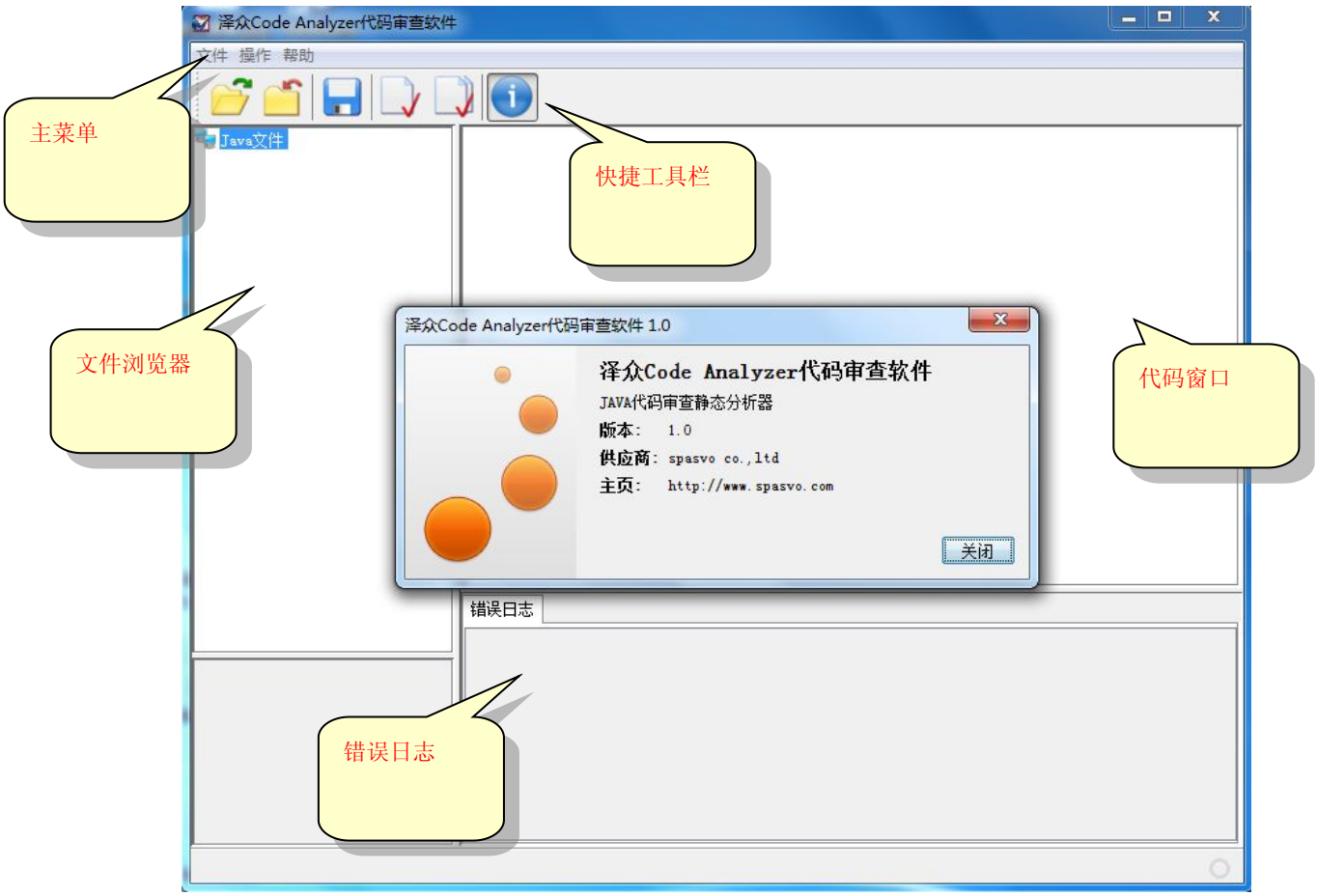
		2. 如果循环体内存在逻辑判断，并且循环次数很大，宜将逻辑判断移到循环体的外面
语句的循环控制变量		
		1. 不可在 for 循环体内修改循环变量，防止 for 循环失去控制
		2. 建议 for 语句的循环控制变量的取值采用“半开半闭区间”写法
switch 语句		1. 每个 case 语句的结尾不要忘了加 break，否则将导致多个分支重叠（除非有意使多个分支重叠）
		2. 不要忘记最后那个 default 分支。即使程序真的不需要 default 处理，也应该保留语句 default : break
goto 语句		1. 禁止 GOTO 语句
strcpy 语句		1. 对应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
		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

		10. 尽量不要使用类型和数目不确定的参数
		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
		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
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
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。
		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

2.5.用户界面（GUI 、COMMAND）

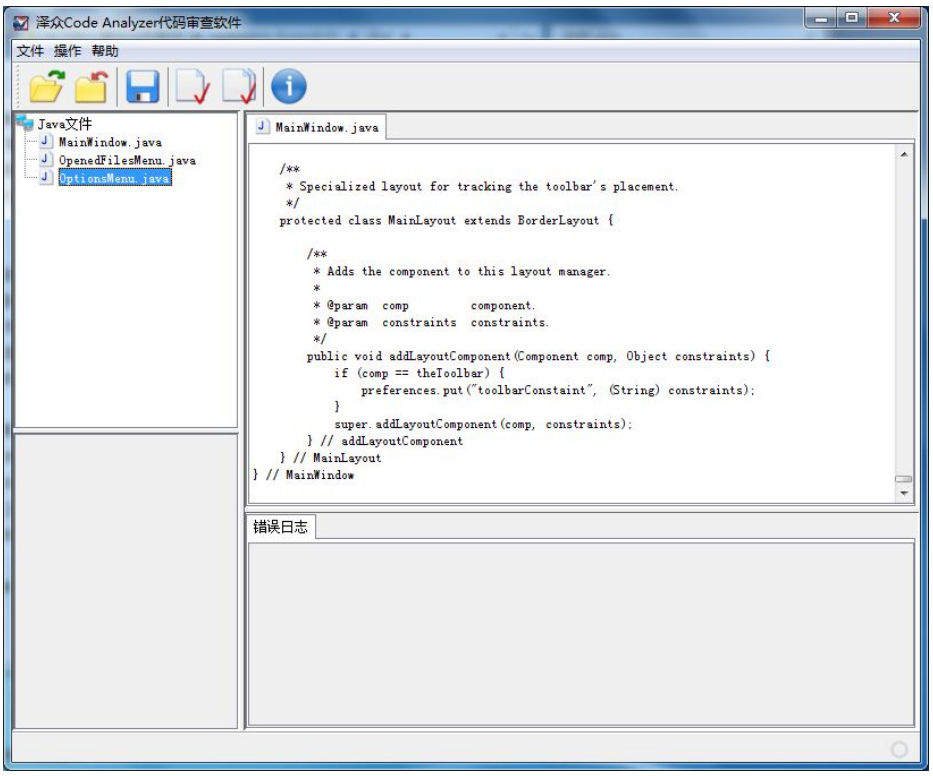
CA 提供了 GUI 和 COMMAND 两种不同的用户界面以适应不同的用户需求。用户可以通过 GUI 界面单独使用 CA，也可以通过 COMMAND 将静态分析与其他过程集成起来。



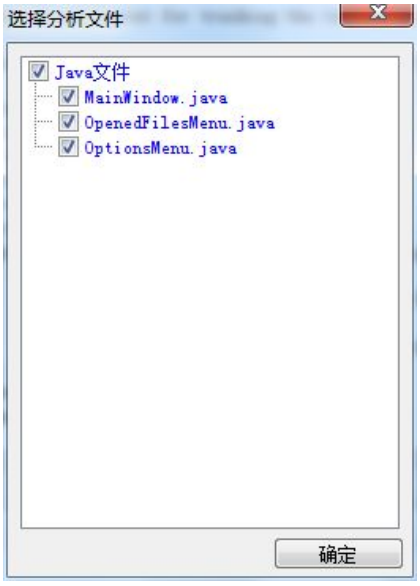
如上图所示，为 CA 图形界面的主窗口，主窗口由若该子窗格组成：

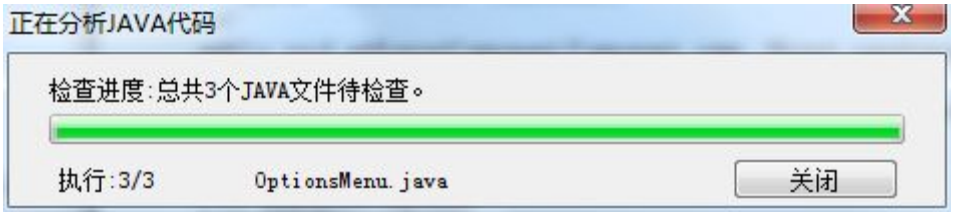
1. 主菜单——提供所有的功能
2. 快捷工具栏——提供快捷访问的功能
3. 文件浏览器——允许用户查看并访问本地源代码
4. 代码窗口——允许用户查看并编辑本地源代码
5. 错误日志——显示分析结果

如下图所示，通过文件选择器导入了三个 JAVA 源程序，在文件浏览器中选中的文件的内容将显示在代码窗口中。

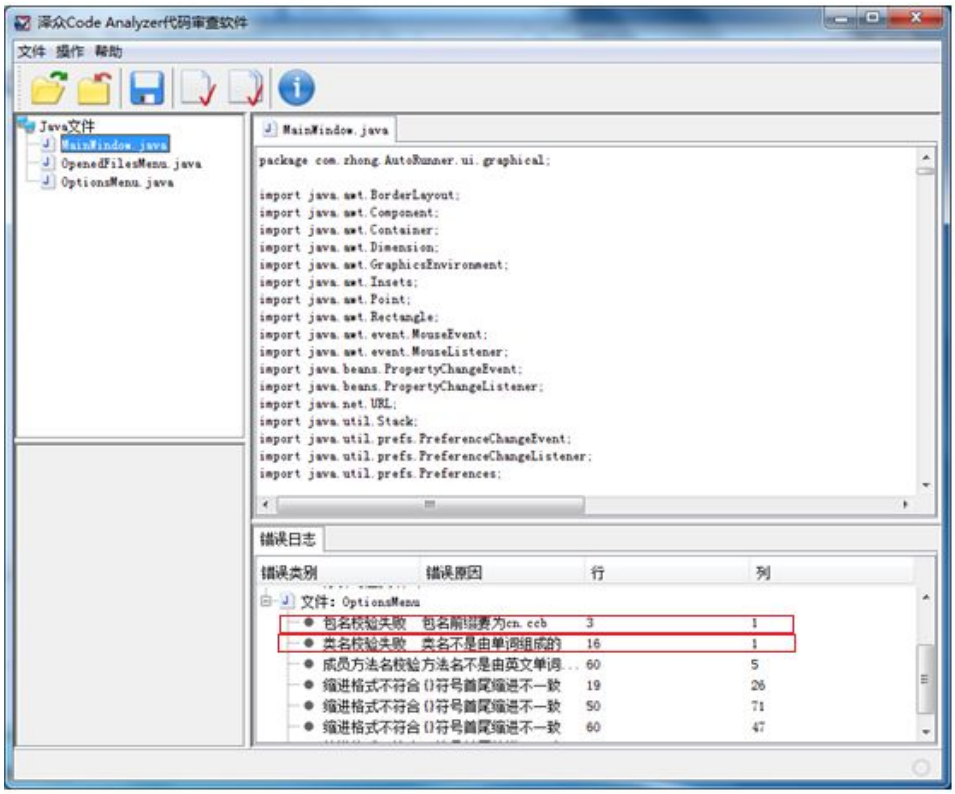


选择菜单 操作/批量分析，将弹出批量分析文件选择框，用户可以勾选要分析的代码文件，点击确定开始分析，CA 将弹出进度条来显示分析的进程。





分析结束后，分析的结果将显示在错误日志中，每一个错误将关联一个错误的说明以及错误发生的所在行和列。

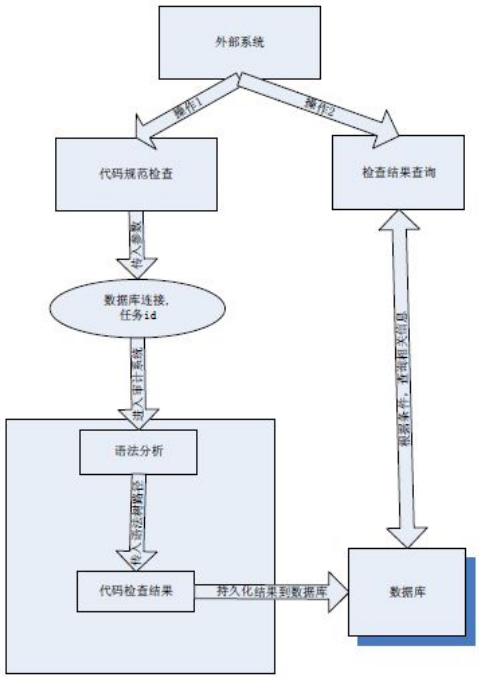


2.6. 用户接口

CA 为高级用户提供用户接口 API，通过调用用户接口 API，用户可以方便地将静态分析工具集成到其它测试工具与平台中：

1. 与代码审计系统集成
2. 与配置管理平台集成
3. 与测试平台集成

如下图所示为对外 API 的结构示意图：



CA 的 API 是通过标准 JAVA JAR 文件发布的，具体的 API 定义如下表所示：

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

3. 功能描述

3.1. 分析扫描能力

CA 支持业界标准的代码安全漏洞定义规则，也支持代码规范检查等规则。

主流的编程语言，目前 CA 支持：c 和 java 编程语言。

支持主流被测试系统平台。CA 内部通过编译技术来实现，通过 LL1 文法分析建立语法树，再通过扫描文法树来进行规则检查。因此，CA 是根据标准的 C 和 java 的语法来进行编译和扫描的，实现了平台无关性。

CA 无需编译程序。CA 本身包括了一个编译器，不再需要额外的编译程序和第三方程序来实现代码扫描，并且与开发环境无关。

对主流框架的支持。CA 对主流框架不进行特别关注，凡是基于 java 语法和 c 语法的框架均可以运行。如果需要对框架的行为来定义规则，则可以额外增加对配置文件等进行规则定义来实现针对性的规则检查。

支持主流数据库的 PL/SQL 语言。

CA 的结果查看支持 B/S 结构：把扫描的结果存放在 CA 服务端的数据库中，通过浏览器来访问扫描结果。支持 IE、Firefox、Safari 等浏览器。

3.2. 扫描规则及自定义能力

CA 采用先编译再扫描的方式工作。因此，用户可以非常方便的通过潜入标准检查规则代码的方式来扩展规则。

对 xml 语法树的访问也非常简单容易，因此编写负责的语义规则是非常简单和容易的。

对词法规则，CA 提供了标准化的英语词语字典，可以支持复杂的词法规则。

对于语义规则，CA 可以支持用户采用 pattern 的方式来扫描 xml 语法树实现。

3.3. 持续集成能力

- **支持主流版本管理（配置管理）工具。**

CA 提供了丰富的接口，可以通过版本管理工具的脚本语言来调用（配置管理工具，cvs、svn、clearcase、firefly 等均提供脚本）来实现与配置管理工具的集成。

- **缺陷追溯、支持缺陷管理工具**

CA 通过客户化，把扫描发现的问题归入到缺陷管理系统。通过调用各个不同的缺陷管理系统的 API 实现集成。

把 CA 扫描程序作为一个提交缺陷的用户即可，也可以分配到一个具体的用户上来实现。

- **自动监控版本服务器，并触发代码扫描及检测分析**

CA 可以通过以下方式来实现：

修改配置管理的提交脚本，潜入扫描触发程序。当用户 checkin 程序的时候，脚本触发了 CA，来进行代码扫描，并且提交扫描结果；

- **支持 SMTP 邮件服务功能**

CA 支持 SMTP 的接口，可以根据需要向指定的 SMTP 发送请求，提交发送的邮件。

CA 需要配置固定的用户名、密码，作为邮件发件人。

3.4. 开发环境集成能力

- **支持主流 IDE 环境，开发人员桌面上即可进行扫描。**

CA 支持通过命令行方式嵌入 IDE 的方式；也支持通过提供客户端的方式来工作。

对于命令行的方式，可以通过配置 IDE 环境来调用命令行工作；

对于客户端方式，用户可以通过操作客户端来扫描指定的代码，甚至整个项目。

3.5. 报表功能

- **支持测试结果进行统计分析，并产生统计分析报表。**

CA 把扫描结果输出到测试管理平台的测试日志和缺陷。之后，就可以在测试管理平台的缺陷管理模块来分析缺陷了。

测试管理平台本身提供自定义报表和分析，能够很好的实现缺陷分析和查看。

- **提供多种方式的检测报告，包括 PDF、word、excel。**
  CA 支持把扫描分析结果输出到测试管理平台。
  测试管理平台支持 PDF、WORD 格式的检测报告，也可以省略中间环节，提升系统的易用性。

3.6. 系统管理功能

- **LDAP 集成，支持组织级的用户、角色以及权限设置。**
  CA 可以实现通过与 LDAP 集成，支持组织级的用户、角色及权限设置。
  CA 同时可以实现与现有的测试管理平台和项目管理系统集成，实现单点登录和单一用户 ID 登录管理。
- **支持跨项目及项目群管理设置。**
  CA 通过集成到现有的测试管理平台和项目管理平台，来实现跨项目和项目群的管理和设置。
  用户采用当前项目管理平台和测试平台的用户，扫描的结果体现在这些项目上，便于统一管理和查看。查看和管理的权限来自项目管理平台和测试管理平台。
- **支持云服务实现，支持跨 internet 实现源代码安全扫描“云服务”。**
  CA 可以改造成云服务的版本：通过本地化扫描程序，生成 xml，上传到云服务，再进行扫描处理，在云端保存扫描结果，并且提供浏览器访问服务。

3.7. 快速部署与集中扫描能力

- **与版本管里服务器集成，自动扫描。**

  CA 提供命令行模式和接口，与版本服务器集成。

  当版本服务器发起 checkin 或者 new 等命令的时候，脚本被启动，同时调用 CA 来执行扫描。

  扫描的结果被发送给测试管理平台，作为执行日志和缺陷存放。
- **集中快速部署，不改变或影响被测试系统的原有编译过程。**
  Checkin/new 等版本管理的操作，本身对程序无任何影响。
  CA 自带编译器，对程序进行编译处理，不需要借助其他的编译器，也不需要额外的编译处理。

3.8. 低误报率

- 符合业界 OWASP 质量缺陷及安全规则定义的误报率低于 15%

OWASP 包含了以典型下风险，以及处理方式：

安全风险	应对策略
第一位：注入式风险	数据库注入测试

第二位：跨站点脚本 （简称 XSS）	处理问题脚本
第三位：无效的认证及会话管理功能	客户端脚本分析，session 安全管理
第四位：对不安全对象的直接引用	扫描后台处理代码
第五位：伪造的跨站点请求(简称 CSRF)	N/A
第六位：安全配置错误	分析配置参数
第七位：加密存储方面的不安全因素	N/A
第八位：不限制访问者的 URL	对 URL 的限制规则
第九位：传输层面的保护力度不足	对敏感数据的保护扫描
第十位：未经验证的重新指向及转发	扫描转发的 url 和转发内容

CA 实现了对 session 安全管理的规则，以及可以通过订制实现转发、加密扫描、对象引用方面的规则，可以很好的支持 OWASP。

误报率，基于编译处理的误报率，能够排除由于 pattern 的模式不够精确造成的误报，只要定义的语法结构是合理的，可以使得基本没有误报。

3.9. 低漏报率

- **符合业界 OWASP 质量缺陷及安全规则定义的漏报率低于 15%。**

关于 OWASP，上文已经说明。

漏报的原因在于：很多的静态分析程序是基于模版的，而当代码编写的方式与模版存在一定的出入，就会导致误报。

CA 的原理是构建了语法树，如果模版定义的语法树是正确的，就可以排除由于特殊代码编写模式而导致的漏报。

4. 非功能描述

4.1. 软件技术要求

- **稳定性：系统稳定，能连续运行 3 个月不宕机**

CA 分成两个部分：扫描分析部分和报表分析部分。

对于扫描分析部分，它一般在很短的时间内启动运行，在生成报告之后进程就结束了，不会由于持续运行造成问题的累积。

对于报表部分，CA 的报表服务器基于 java 服务，运行稳定。另外，本方案建议把扫描

的结果输出到测试管理平台，以进一步提高集成度和稳定性。

综上，CA 运行稳定，能够实现连续运行 3 个月不宕机。

- 运行性能：
  - **出具第三方性能测试报告**

    由于 CA 的最消耗 CPU 的部分是在各个客户端来执行，因此对并发性能执行的要求不高。

    目前没有第三方性能测试报告。
  - **能够支持 30 个并发和 300 个在线用户的支持，并能保障操作员及时的系统反应时间，平均操作反应时间不大于 2 秒**

    支持 30 个并发和 300 个在线用户。

    平均操作反应时间小于 2 秒。
  - **万行代码扫描分析时间不超过 1 分钟，10 万行代码扫描分析时间约 10 分钟**

    根据测试 CA 为：万行代码扫描分析时间 1 分钟内，10 万行代码扫描时间为 10 分钟内。

- **安全性：存储目录不共享，对客户端不透明，客户端不可直接访问存储目录**

  CA 的方案是，客户端把分析日志插入到数据库中，不访问文件系统。

- **易用性：本地化中文的安装、操作界面及缺陷报告，安装、配置、使用较为简单，包括安装、配置、培训在内的整个实施周期不会超过一个月。**

  客户端安装在几分钟内完成。

  安装、配置、培训能够保证整个实施周期不超过一个月。

- **兼容性：**
  - **能与一般的开发工具 IDE 很好的兼容，支持第三方的插件并能针对客户现场项目情况定制支持**
  - **底层数据库支持 DB2、SQL-SERVER、ORACLE 等多种数据库**

提供脚本和 API，能够与开发工具 IDE 集成和兼容。

支持第三方插件，支持根据客户需求订制。

支持 DB2、SQL-SERVER、ORACLE 等多种数据库。

- **可移植性：平台移植性好，支持 WINDOWS XP，Windows7 等常用 32 位或 64 位操作系统**

  目前已经支持 windowsXP、win7、windows server2003 等多种系统。

4.2. 软硬件环境需求

- **系统软件：对操作系统要求不高，能够安装各种主流系统和平台。占用系统资源少。与常见开发工具 IDE 无缝集成**

能够支持 windows XP、windows7、windows2000、windows server2003 等系统。

占用系统资源少，主要是空间在：200M 之内；

能够与常见开发工具集成。

硬件需求：

（1） 应用系统对硬件设备(包括主机和桌面设备)必须具备的最低技术指标要求（供性能测评用，不作为评估依据）

（2） 建议各种硬件的选型和配置（供性能测评用，不作为评估依据）

（3） 在上述指标和配置下系统可提供的处理能力估算，并提供一个投标人的实际案例做出的比对参照

硬件要求：

服务器：

CPU：Intel Xeon 系列，4-6 核

内存：DDR3 8G

硬盘：SATA 500G

网卡：100M

建议的硬件：BL680c G7(643782-B21)

- 产品类别：刀片式
- CPU 型号：Xeon E7-4830 2.13GHz
- 标配 CPU 数量：2 颗
- 内存容量：64GB DDR3
- 网络控制器：6 个 10Gb NC553i FlexFabric，4 端...
- RAID 模式：RAID 0，1，5

- 扩展槽：7×PCI-E Gen2
- 光驱：标配不提供
- 最大 CPU 数量：2 颗
- 最大内存容量：2TB
- 内存插槽数量：64
- CPU 类型：Intel 至强 E7-4800

系统组件要求：

操作系统：windows XP 或者 windows2003。

数据库：SQL SERVER、oracle、DB2。

中间件/容器：tomcat

客户端：程序客户端、IE 浏览器

配置管理接口：支持与 CVS/SVN/Firefly 连接;

缺陷管理接口：支持与测试管理平台对接。

4.3. 其他需求

- 中文支持

  支持中文系统、中文操作界面、中文文档、中文帮助和用户手册。
- 系统资源

  普通 PC 机即可满足需要。

- 支持不同平台

  支持 windows 绝大多数操作系统。

  支持 linux、aix 上的 c 语言，和标准 java 语言。

4.4. 测试报告生成

CA 可以生成不同输出形式的报告：

1. 通过 GUI 显示报告，并将通过报表对静态分析的结果进行统计分析；
2. 通过 LOG 文件保存报告。
3. 加载到测试管理平台的测试报告。

在本方案中，我们建议通过把静态分析结果发送到测试管理平台来进行统一管理，包括缺陷管理。

4.5. 与其它系统、平台集成

在测试体系中，通常使用代码审计的方法来保证代码的质量。最常见的方法是将代码静态分系工具集成到审计系统中，在编码用户提交代码之前分析代码的质量并将分析结果记录到审计数据库。

利用 CA 提供的用户接口 API 可以很方便地将 CA 集成到审计平台中，通过输出配置，将 CA 分析的结果数据保存到数据库表中，最后

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

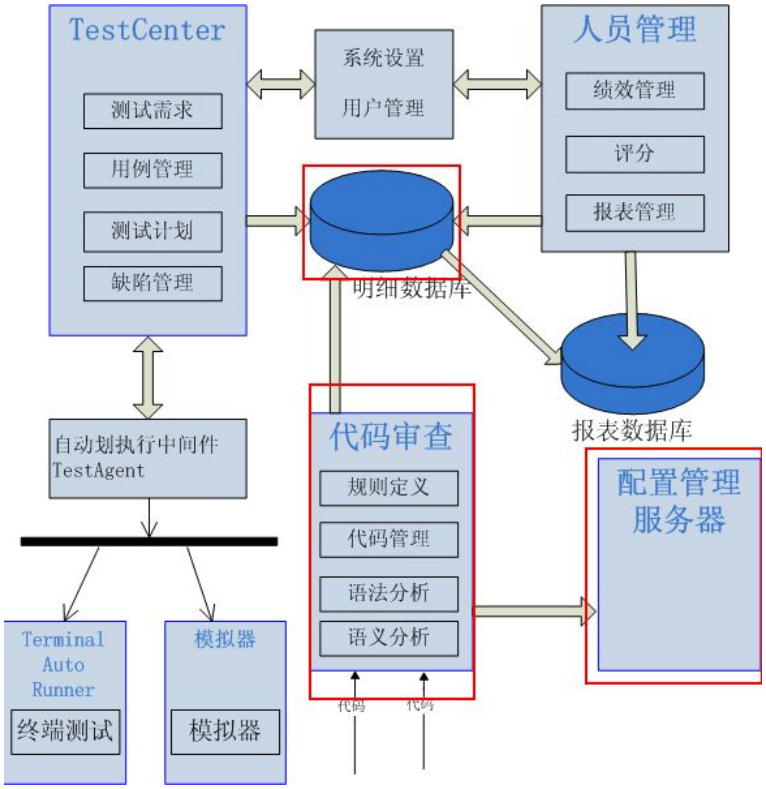
根据用户制定的《代码编写规范》来设定代码规则；给代码进行评分。

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。

5. Code Analyzer 的特点

评估静态分析的关键在于：第一，规则的完整性与可用性；第二，二次开发与自定义规则；第三，便于维护使用；第四，便于与其他工具集成；第五，扫描速度的可用性。

下面，我们就 CA 在这几个方面的特点简要介绍：

CA 无需测试用例的测试。静态分析工具与其他测试工具的差异在于：无需测试用例。CA 是根据预订的规则来对代码进行扫描分析，无需编写测试用例就可以实现自动化测试。通过对代码的分析，检查代码是否符合编码规范和各种规则，来检查程序代码可能的错误，这将为您节省大量的人力。

基于编译的代码分析。静态分析工具分成基于编译的和基于模式的。基于模式的方法，在于通过扫描

某个段落的上下文是否符合一个规则的模式，来判断是否违反了规则。而基于编译的方法是对真个代码进行扫描分析，变成语法树来识别，能够获得更精准和完备的分析，也便于用户自定义规则的实现。。

跨平台。CA 本身全部基于 JAVA 来开发，支持 windows 平台和 linux 平台，包括命令行环境和 IDE 环境都可以显示。基于 JAVA Swing 的模式便于系统跨不同的平台运行。

集成与扩展性。CA 的使用时基于用户的软件生命周期环境来进行的，因此可以集成在用户的 ALM 管理过程中。通过提供开放的接口，支持与测试管理软件、项目管理软件等集成。

6. 支持服务

6.1. 服务支持

CA 产品通过在线 MSN、电话、电子邮件为您提供支持与服务，也可访问我们的网站 http://www.spasvo.com/。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。

6.2. 技术支持

- 售后服务电话：400 035 7887
- 电子邮件：support@spasvo.com
- 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。
- 电子邮件： sales@spasvo.com