第3章　程序理解工具

程序理解是人们将程序及其环境对应到面向人的概念知识的过程，它是软件开发过程中的一项重要活动，无论是软件的维护还是测试，抑或是逆向工程，都离不开对源代码的理解。

尽管程序理解可以手工进行，但要达到好的效果、高的效率，就必须要运用程序理解技术并在工具的支持下进行。随着软件规模及复杂度的不断增大，程序理解也变得越来越困难，需要耗费理解人员大量的时间和精力，却往往还不能得到理想的效果。因此，对通过计算机来完成软件系统分析和理解的程序理解辅助工具的需求变得越来越迫切。

3.1　程序理解概述

对于软件系统特别是大型复杂软件系统来说，由于分析和理解的困难性，导致其系统维护或系统演化任务异常艰巨，且成本开销巨大。根据 Boehm 等人的研究统计，软件维护开销占系统总成本的 50%～80%，而维护开销的 47%～62%则用于对软件系统的分析与理解。

3.1.1　程序理解的概念

程序理解是从计算机程序中获取知识信息的过程。这些知识信息可用于程序排错、程序增强、程序重用以及文档整理等方面的工作。程序理解的目标是从不同抽象层次，多视角、多方面地综合表达并展示程序理解结果，以加速对软件系统的理解。

程序理解是软件工程领域的一个重要部分。软件工程最为关心的是如何提高软件开发效率和软件产品质量，但是，就目前实际情况来看不容乐观。软件开发精力大多都花费在维护老系统上，而不是开发新系统上，用于维护、逆向工程或再工程方面的资源和时间占了 50%~70%。而软件维护过程的绝大多数时间被用于理解目标系统，国内外最新研究结果表明，维护和逆向工程工作的 70%的时间花在对系统的理解上。因此，程序理解不但是软件维护中的一个重要部分，而且还在整个软件维护、逆向工程或再工程过程中起到了举足轻重的作用。另外，程序理解对静态测试中的代码检查(如代码走查、代码评审等)能够起到有效的支撑作用。

程序理解旨在理解一段现存的程序，通过不断从源程序中抽取所需信息，检查部分代码，来逐步构建所需的理解。理解过程中，需要不断地从中获取知识并提炼。程序理解是一个复杂的过程，它需要分析目标系统，标识目标系统组件及其相互关系，创建不同形式或更高抽象层次的系统表示。程序理解的目标是理解软件系统以促使性能提高、纠错、建档、再设计或使用另外一种语言重新编程。不论是对目标系统进行概念建模、数据抽取还是系统抽象，都主要依赖于通过分析程序源代码来抽取程序结构和控制流信息。因此，程序理解是软件逆向工程主要的实现手段和行为活动，它贯穿于整个软件逆向工程，并且是决定软件逆向工程成败的关键。

3.1.2　程序理解的任务与内容

程序理解的任务是在不同的抽象级别上建立基本程序的思维模型，实际上就是建立从问题(应用)领域到程序设计(实现)领域的映射集。其范围包括从代码本身的模型到基本应用领域的模型，其目的是为了便于软件的维护、进展和再工程。从概念上讲，可以从抽象程度的几个不同层次来理解一个程序。归纳起来，有 4 个抽象程度不同的层次：实现层、结构层、功能层和领域层。

实现层是从程序设计语言的角度去对程序进行理解。检查单个的程序设计结构，程序按某种方式表示成抽象语法树、符号表或普通源文本。实现层包括程序扫描、语法提取、语义检查、静态分析、动态模拟运行等几个过程。

结构层是在程序语言的基础上，检查程序构造过程中的结构，对程序语言中所出现的各种实体(如全局变量、数据结构、过程等)以及它们之间的关系进行分析，如数据调用和控制流程图、程序调用图等，明确表示程序各组成部分之间的依赖关系。结构层包括逆向工程、信息提取、信息抽象、结构模型匹配、识别构造规范和结构、聚集分级结构等过程。

功能层是从程序中不同模块的功能来推断它们之间的逻辑联系。检查两个程序结构和行为(功能)之间的关系，同时研究程序构造的合理性。功能层包括设计恢复、语义和行为模式匹配等过程。

领域层是检查特定于应用领域的概念，进一步从功能上来推断此软件在其领域中的作用。领域层包括智能软件理解、格局识别、概念赋值和推理等过程。

以上 4 个层次是从一个抽象程度较低的层次(一般源代码)到一个抽象程度较高的层次的转换。这里主要介绍结构层的理解技术。

经过分析，全面、准确、迅速地理解程序，对于软件的开发、维护、质量保证及逆向工程和再工程有着重要意义。不过，程序理解的内容比较多，要讲述程序理解的内容，就必须结合具体的程序设计语言。尽管如此，程序理解一般都包含如下内容。

(1) 程序理解的功能和目标。

(2) 掌握程序的结构信息，即从程序中细分出若干结构成分，如程序系统结构、控制结构、数据结构、输入输出结构等。

(3) 了解数据流信息，即涉及的数据源于何处，在哪里被使用。

(4) 了解控制流信息，即了解每条路径的结果。

(5) 程序理解的操作(使用)要求。

(6) 面向对象的理解(对象、类、继承、通信等)。

3.1.3　程序理解的相关技术

结构层的理解技术有语句分析、表示程序单元之间关系的调用和被调用图、与程序内部结构相关的程序控制流图和数据流图等。

1. 语句分析

与自然语言类似，程序文件是由语句构成的，包括标识符、操作符、关键字、字符串、数字、标点符号等词法单元。语句分析分别构造程序的词法模型与语法模型，它们对应于词法分析与语法分析。由于这两种分析不生成任何语义信息，所以它们生成的模型仅适用于源代码的模式匹配。

2. 程序流分析

程序流分析技术，即在程序运行之前，通过静态分析去发现程序在运行行为方面的某些特性。程序流分析包括控制流分析和数据流分析两种，其中控制流分析侧重于对程序结构的分析，而数据流分析则侧重于对变量控制结构中数据的赋值、使用及传递情况的分析。

程序理解的首要任务是发现它的控制结构，即语句的可能执行路径，通过控制流分析建立过程内部的控制层次。控制流分析可以分为两大类：必经点分析和区间分析，这两种分析方法都是先对程序文本进行分析，将其转化成某种中间表示，然后在中间表示的基础上进行分析，具体选用哪种方法需根据程序理解的任务而定。

数据流分析是为了计算被分析程序在生成数据方面的行为，通常用于程序优化，即为程序优化建立环境。目前，使用最为广泛的方法是对控制流图进行循环分析，称为迭代数据流分析。

由于非结构化程序会给测试、排错和程序维护带来较大的困难，因此按照结构化程序设计的要求，理想的程序设计是尽量避免使用 goto 语句，程序的控制结构尽量做到单入口、单出口。基于这些原因，在对被测软件进行分析时，系统地检查程序的控制结构成了十分有意义的工作。

3. 软件结构图

软件结构图分为程序调用关系图(或函数调用关系图)和系统结构图。函数调用关系图或者说是程序调用关系图，都是对源程序中函数关系的一种静态描述，在函数调用关系图中，节点表示函数，边表示函数之间的调用关系。

系统结构图反映的是系统中模块的调用关系和层次关系，即谁调用谁，这里有一个时序关系。在系统结构图中，有向线段表示调用时程序的控制权将从调用模块转移到被调用

模块，并隐含了当调用结束时控制权将从被调用模块交回给调用模块。若一个模块有多个下属模块，那么这些下属模块的左右位置可能与它们的调用次序有关。

3.1.4　程序理解工具

阅读源代码是程序理解的一项重要活动，但是阅读别人的代码是枯燥乏味而且比较困难的工作，所以开发辅助工具成了程序理解的一项重要研究内容，并且在这一领域已经有了很多成果。这些工具能以更清晰、更可读、更可理解的方式组织和表示源代码，把人们从烦躁的代码阅读中解放出来。常见的辅助工具有以下几种：程序切分器、静态分析器、动态分析器等。程序切分器能够帮助程序员选择并只观察所提议更改影响的程序部件，不受无关部件的干扰，显示数据链和相关特征，使程序员能够跟踪更改影响。静态分析器能够帮助程序员快速提取模块、过程、变量、数据元素、对象与类、类层次结构等信息。在理解过程中，理解人员应该使用这些工具，以提高理解效率。

目前，除了针对 C++、Java 以及 Ada 等语言而专门开发的程序理解工具 Understand 外，专门用于程序理解的工具还不是很多，并且其中大多是作为辅助功能用于支持开发、测试或其他任务，如 Logiscope、Panorama++、McCabe IQ、Klocwork 以及有关的 IDE。国内北大青鸟在“九五”期间专门将 C++的程序理解工具作为科技攻关项目，并取得了较好的成绩，但遗憾的是并未看到他们广泛应用的商业化产品或开源产品。

令人感到欣慰的是，可以在网上找到几款不错的程序理解工具，尽管它们还存在不足或功能不全，但作为学习和实践之用还是足够的。

3.2　Oink 程序理解工具

Oink 是一个开源的、能够对 C 和 C++程序进行静态分析的工具，但它的基础或核心部分是程序的理解功能。

Oink 主要由 Scott McPeak、Karl Chen、Daniel S. Wilkerson 等人开发和维护。Oink 最基本的工具是 Cqual++，Cqual++的开发借鉴了开源工具 Cqual 的许多设计思想，并在其基础上进行了扩展和创新(基本上重写了代码)。Cqual 是由 Jeffrey S. Foster 等人开发的，可以对 C 程序进行基于类型(type-based)的数据流分析，而 Cqual++则可以对 C 和 C++程序进行基于类型的数据流分析。

Oink 的源码包(Oink-Stack)中有 smbase、ast、elkhound、elsa、libregion、libqual 和 platform-model 这几个部分。smbase、ast、elkhound 和 elsa 主要是由 Scott McPeak 开发的。

(1) smbase 包是由 Scott McPeak 开发的一个 C++的包和字符串库，用来代替 C++标准库的相关库，也可将此库用在别的项目开发中。

(2) ast 包是用于生成 AST(抽象语法树)的工具，它是 Oink 的可扩展的前端。

(3) elkhound 是用于管理 GLR 语法的分析器，它和 bison(一个用于自动生成语法分析器的程序)的功能相似，但 elkhound 还可以用来分析任何上下文无关的语法，而 bison 需要先生成 LALR(1)的上下文无关语法，然后再将其描述转换为可作语法分析的 C++程序(在新版的 bison 中已经加入了对 GLR 语法的分析)。

(4) elsa 包是由 Scott McPeak 开发的软件前端，它是在 elkhound 语法分析的基础上，将 C 和 C++程序生成为相应的 AST；elsa 也可以对程序进行部分类型检查，主要是与生成程序相关的部分，但不能期待它会做所有的类型检查工作。

(5) libregion 包是由 David Gay 开发的基于区域(region-based)的 C 语言内存管理库，这个库沿用了 Cqual 的部分。

(6) Libqual 包是由 Rob Johnson 开发的可序列化的、多态的类型修饰符推理接口，它在 Cqual 的基础上进行了部分的修改。

(7) platform-model 包是由 Karl Chen 开发的针对 C 库和 C++标准库程序的静态模型。

Oink 可以对 C 和 C++程序进行许多静态分析，主要是对数据流进行分析。在程序理解中，包括对源程序进行表达式级(expression-level)和类型级(type-level)的数据流以及语句级的(statement-level)控制流进行分析处理(这是通过 elsa 的相关功能实现的)，并可以实现对程序数据流图、控制流图、类继承关系图(C++)等的输出显示。

开发 Oink 主要是基于以下两个方面：一个是具有很快地查找程序错误和缺陷的能力，希望能和许多商业级的程序理解工具相媲美(在某些方面也的确做到了)；另一个是具有良好的可扩展性，可以比较方便地实现对工具前端和后端的裁剪和扩展。

Oink 在程序理解方面的应用主要是在结构层，即在程序语言的基础上，检查程序构造过程中的结构关系，对程序语言中出现的各种实体(包括全局变量、数据结构、过程等)以及它们之间的关系进行分析，如数据流图和控制流图等，给出程序各组成部分之间的依赖关系。

关于 Oink 的更多资料，可以去网站 https://github.com/dsw/oink-stack 查看。

3.2.1　Oink 环境建立

Oink 可以成功安装到一些版本的 Linux 系统、Apple OSX 系统以及 Linux 的虚拟机上。如果在 Windows 系统下安装了 cygwin，那么也可以在 cygwin 上安装并使用 Oink。

下面介绍 Linux 系统下 Oink 环境的建立(这里以在 VMWare 上安装 Fedora 8 系统为例来说明安装过程)。

(1) 下载 Oink 源代码。

免费下载网址：https://github.com/dsw/oink-stack。

(2) 安装 Oink。

在 Linux 系统终端上，进入解压缩后的安装文件目录，输入：

```
./configure && make clean all check
```

经过一段较长时间的等待后，可能会出现如图 3-1 所示的报错信息。

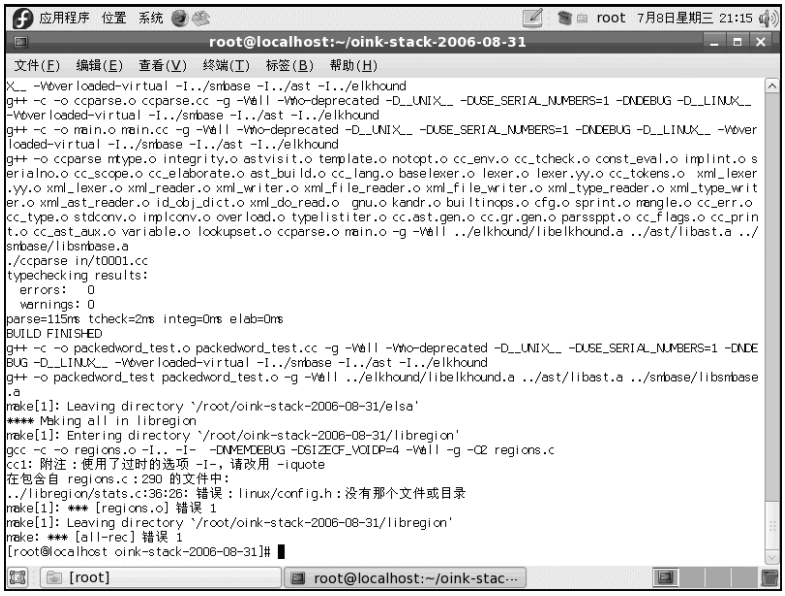

图 3-1　最初安装时可能出现的报错信息

这是由于 Oink 软件与该操作系统的内核版本存在兼容性问题。根据提示，将 libregion 文件夹里 stats.c 文件的第 36 行的#include<linux/config.h>去掉或者注释掉，问题即可解决。

再重新运行：./configure && make clean all check。

再经过较长时间的等待，如果出现如图 3-2 所示的结果，则说明安装成功。

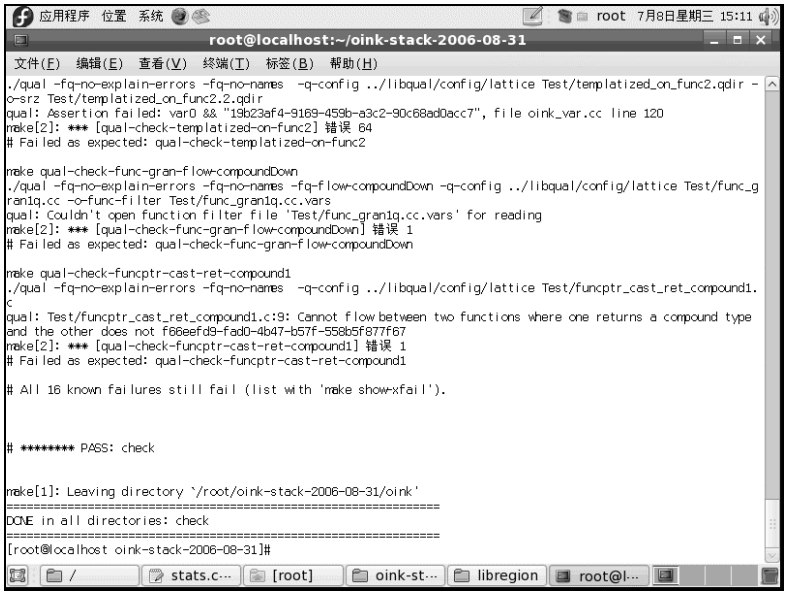

图 3-2　Oink 安装成功

Windows 系统下 Oink 环境的建立步骤如下。

(1) 安装 cygwin 软件。可以在网站 http://www.cygwin.com/上找到 cygwin 的下载和安装信息，这里不再详述。

(2) 打开 cygwin，将下载的 Oink 压缩包在 cygwin 下解压缩。

(3) 在 cygwin 下进入刚才解压缩后的文件夹，运行“./configure && make clean all check”。当出现“DONE in all directories: check”提示时，说明安装成功。

安装时需注意以下几点。

(1) 在安装之前最好查看系统的 GCC 版本，以及安装 Oink 所依赖的 Flex、Bison、Python、Perl 等软件是否安装，并且版本是否合适。GCC 版本可以是 gcc 3.2、gcc 3.4、gcc 4.0、gcc 4.1。有些最新的 Linux 系统默认安装的 gcc 版本比较高(如 gcc 4.3)，安装时会提示有许多 C++源程序语法错误，如找不到一些标准库文件、命名空间定义不对等。要安装成功，就需要手动将这些问题在源代码中按标准 C++规则修改过来。Flex 版本可以是 flex 2.5.4、flex 2.5.31、flex 2.5.33。Bison 版本可以是 bison 1.35、bison 2.1、bison 2.3。Python 版本可以是 python 2.4。即使 GCC 版本适合，Perl 也最好是 perl 5.8.8 以上的版本。即使都满足以上要求，也有可能无法直接安装成功，这主要是由于 Oink 与系统内核存在兼容性问题，这些问题可以通过手动修改代码来解决。

(2) 可以根据具体的应用，安装可选的依赖软件 dot、libzipios++和 zlib。例如，为了将 Oink 的数据流、控制流等分析结果以图形化形式显示，就需要用到 dot 工具，dot 工具可以将分析生成的 dot 格式文件转换为*.ps 格式的图形文件。dot 工具可以从网站 http://www.graphviz.org/ 下载，上面会有详细的安装说明。libzipios++和 zlib 允许 Oink 读取和生成许多*.qz 和*.qdir 格式的存档文件。

3.2.2　Oink 工具及使用流程

Oink 工具主要包括以下几个。

1. oink

该工具不做任何程序分析工作，它与其他的工具如 staticprint、cfgprint、dfgprint 等共享命令行标志，这几个工具则在此基础上增加一些各自不同的命令行标志。当在终端输入“./oink –help”后，将输出以下结果，这些信息说明了各个命令参数的作用：

```
All arguments not starting with a '-' are considered to be input files.
oink flags that take an argument:
  -o-lang LANG                  : specify the input language; one of:
        KandR_C， ANSI_C89， ANSI_C99， GNU_C， GNU_KandR_C， GNU2_KandR_C，
        ANSI_Cplusplus， GNU_Cplusplus， SUFFIX.
  -o-program-files FILE         : add *contents* of FILE to list of input files
  -o-control FILE               : give a file for controlling the behavior of oink
  -o-func-filter FILE           : give a file listing Variables to be filtered out
  -o-srz FILE                   : serialize to FILE
```

oink boolean flags; precede by '-fo-no-' for the negative sense.

-fo-help，-help，--help　　: print this message and exit
-fo-verbose　　: print the setting of each flag
-fo-print-stages　　: announce each processing stage
-fo-exit-after-parse　　: exit after parsing
-fo-exit-after-typecheck　　: exit after typechecking
-fo-exit-after-elaborate　　: exit after elaborating
-fo-print-startstop　　: delimit transformed output with cut lines
-fo-func-gran　　: compute and print function granularity CFG only
　　(use -o-srz to write to file)
-fo-func-gran-dot　　: print function granularity CFG in dot format
-fo-all-pass-filter　　: assert that all variables pass the filter
-fo-print-ast　　: print the ast
-fo-print-typed-ast　　: print the ast after typechecking
-fo-print-elaborated-ast　　: print the ast after elaboration
-fo-print-ML-types　　: print types in ML-style; AST print，not pretty-pr
-fo-print-buckets　　: print buckets
-fo-print-stats　　: print analysis stats
-fo-print-sizes　　: print internal data structure sizes and exit
-fo-print-proc-stats　　: print process stats
-fo-pretty-print　　: print the ast as source
-fo-trace-link　　: trace linking
-fo-report-link-errors　　: print un-satisfied/over-satisfied function symbols in linker
-fo-report-unused-controls　　: print controls that go unused
-fo-print-controls-and-exit　　: print the controls and stop
-fo-do-overload　　: do overload res.，overriding language default
-fo-do-op-overload　　: do op overload res.，overriding language default
-fo-do-elaboration　　: do elaboration (for C++)
-fo-check-AST-integrity　　: check the AST is a tree
-fo-exclude-extra-star-amp　　: exclude consecutive '&*' or '&*' exprs
-fo-merge-E_variable-and-var-values :
　　optimization: merge Values for E_variable expressions and Variables
-fo-instance-sensitive　　: C mode only.
　　1) fields get unique values per struct instance
　　2) 'void's get unique values by type
-fo-array-index-flows　　: the array index flows through the array deref

2. staticprint

这是一个静态输出工具，可以输出程序的许多静态分析结果。比如，可以输出类继承关系树图和 AST 节点(histogram of AST nodes)。

在 oink 目录下，输入“./staticprint –help”可查看该工具运行时的详细命令参数信息。

为了方便初学者学习，oink 包里面已经写好了一些运行 staticprint、dfgprint、cfgprint 和 Cqual++工具的 makefile 文件。

在 oink 目录下输入“make staticprint-check-print”，将会运行 staticprint 工具，生成 Test/staticprint1.cc 文件的类继承图。程序代码如图 3-3 所示(生成的图与代码中元素的位置有关，所以这里用代码文件截图来说明它们之间的关系，后面的 dfgprint、dfgprint、cfgprint 和 Cqual++工具也是出于同样的原因，对代码文件进行了截图)。

图 3-3　类继承信息

运行后生成的类继承图如图 3-4 所示(参见彩图)。

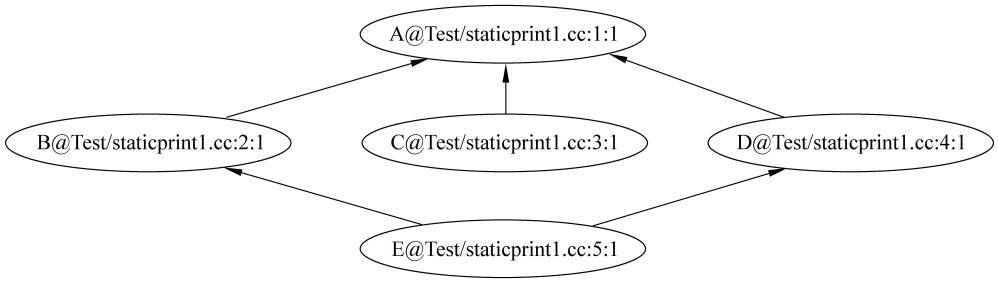

图 3-4　类继承图

从图 3-4 中可以看出 A、B、C、D、E 这 5 个类之间的继承关系，蓝箭头表示的是虚继承，黑箭头表示的是一般继承。

如果要分析其他程序，则可以修改 staticprint_test.incl.mk 文件中的文件名，然后运行即可，流程与上面的相同。

3. dfgprint

用来输出 C++程序的数据流图。

在 oink 目录下输入“make dfgprint-check-print”，将会运行 dfgprint 工具，生成 Test/dfgprint1.c 文件的数据流图。程序代码如图 3-5 所示。

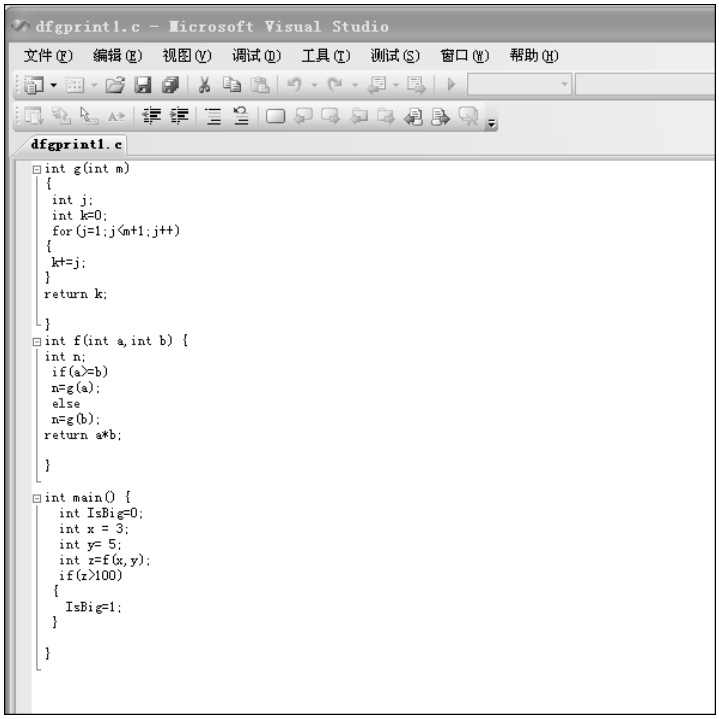

图 3-5　程序示例 1

图 3-6 中(参见彩图)显示了数据流向，红色箭头表示在调用时函数，实参传给形参，数据流向了被调用函数，而蓝色箭头则表示被调用函数将数据返回到调用处的数据流向，至于黑色箭头，则表示一般的数据流向。而对于每个节点来说，@符号之前部分表示数据状态的名称，@符号之后部分表示数据在代码中的位置。如果要分析其他程序，则可以通过修改或添加 dfgprint_test.incl.mk 文件中的 CHK_DFGPRINT 信息来实现。例如：

```
CHK_DFGPRINT+=Test/dfgprint1.c
```

运行后生成的数据流图如图 3-6 所示。

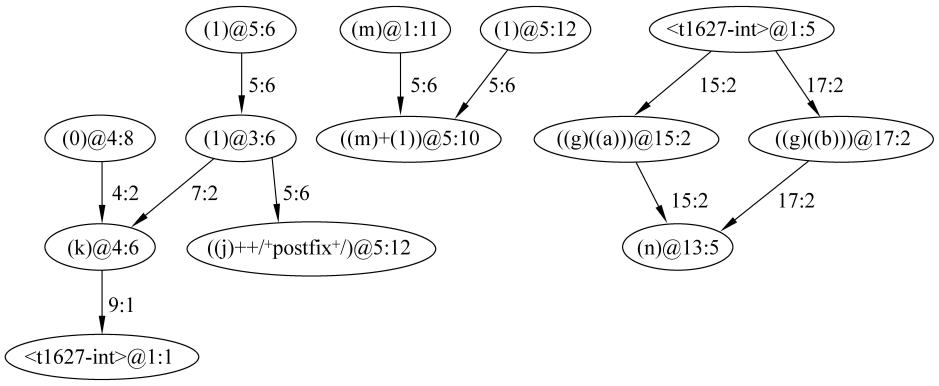

图 3-6　程序示例 1 的数据流图

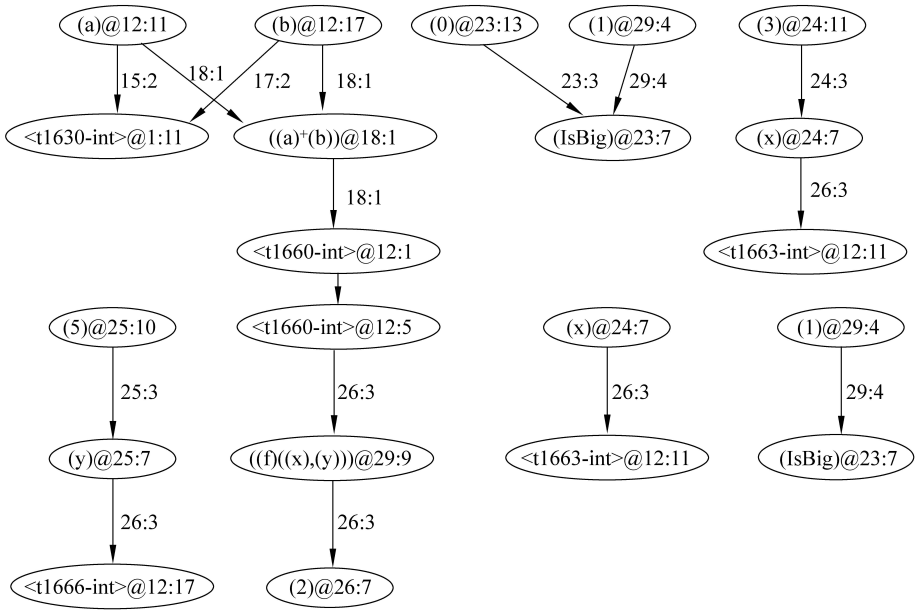

图 3-6　(续)

4. cfgprint

用来输出 C++程序的控制流图。

在 oink 目录下输入“make cfgprint-check-print”，将会运行 cfgprint 工具，生成 Test/cfgprint1.c 文件的数据流图。程序代码如图 3-7 所示。

cfgprint1.c

```
int checkGrade(int g)
{
    int result;
    switch(g)
    {case 1:
            result=1;
            break;
        case 2:
            result=1;
            break;
        case 3:
            result=2;
            break;
        case 4 :
            result=2;
            break;
        default:
            result=3;
            break;
    }
    return result;

}

int main() {

    int m=4;
    int n=0;
       int i;
    for(i=0;i<=m;i++)
    {
         n+=i;
    }
    return  (checkGrade(n/m));

 }
```

图 3-7　程序示例 2

运行后生成的控制流图如图 3-8 所示。

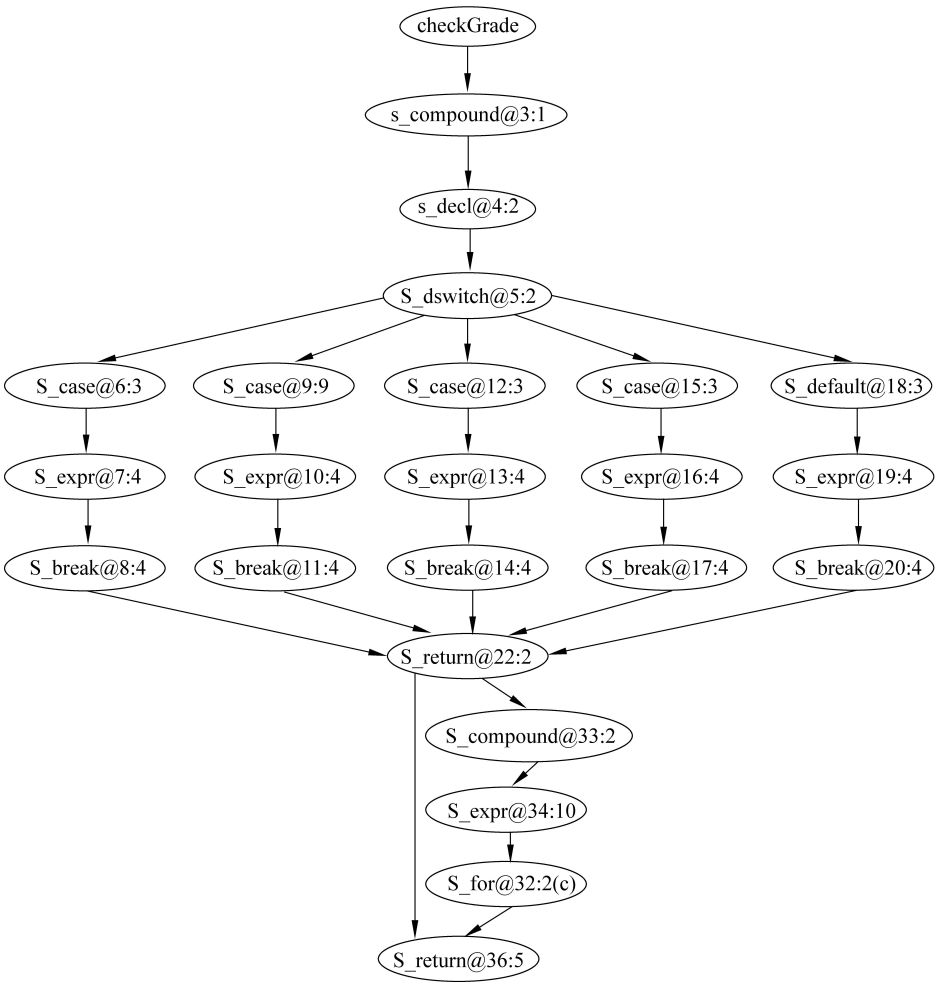

图 3-8　程序示例 2 的控制流图

图 3-8 中显示了程序的控制结构，如果要分析其他程序，则可以修改 cfgprint_test.incl.mk 文件中的文件名，然后运行，具体流程与上面的相同。

5. Cqual++

Cqual++是 Oink 的主要工具，它有许多功能，可以通过检验 C++程序的静态断言来对程序进行分析和理解。在 oink 目录下输入“./qual –help”，可以查看 Cqual++工具的详细命令行参数。

3.2.3　Oink 应用举例

目前，Oink 的主要用途是对代码进行数据流和控制流的分析以及基于断言的类型修饰符分析。数据流和控制流分析的使用方法在 3.2.2 节已经说明，而基于断言的类型修饰符分析可以对代码进行格式字符串缺陷(format string bug)分析。

下面是一个 printf()函数的格式字符串缺陷的例子，如图 3-9 所示为 Test/taint1.c 程序的代码。

图 3-9　程序示例 3

图 3-9 中多了$tainted 和$untainted 这两个在 C 语言语法中没有的修饰符，它们是 Oink 添加的类型修饰符(如果去掉它们，用 GCC 编译器编译运行，将会发现并没有提示错误，这说明 GCC 没有发现程序有错误)。分析程序可知，这个程序要通过函数 getenv()读取环境变量 LD_LIBRARY_PATH，并将这个变量以格式字符串形式传递给 printf()函数。如果一个不能信任的用户在程序运行时随便更改了这个环境变量，那么程序将可能出现格式字符串漏洞。比如，如果用户将环境变量 LD_LIBRARY_PATH 设置为一个很长的字符串，那么可能超出内存空间，产生因越界而使程序终止的段错误。为了查找这样的错误，Oink 在程序中添加了两个类型修饰符——$tainted 和$untainted。$tainted 表示不可相信的数据，而$untainted 则表示可以相信的数据。在第一行定义函数 getenv()的返回值是不可以相信的($tainted)，通过该函数可以得到用户设置的任何环境变量数据；第二行定义的函数 printf()的形参表中的参数是可以相信的($untainted)。

图 3-10 是运行“make qual-check-demo-taint1”后的结果。

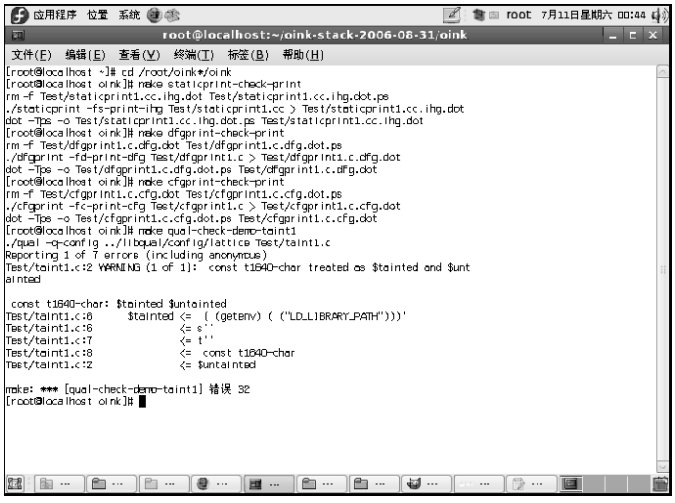

图 3-10　程序示例 3 的分析结果

由图 3-10 可以看出，程序运行结果的第 1 行，即“const t1640-char: $tainted $untainted”

表示的意思是 t1640-char，也就是示例 3Test/taint1.c 中的变量 t 既是$tainted 类型，又是$untainted 类型。前面讲过，$tainted 和 $untainted 是 Oink 增添的两个数据类型，分别表示不安全数据和安全数据。那么在程序运行结果中得到了 t1640-char 既是安全的又是不安全的，这明显是矛盾的，我们认为是一个错误。从程序运行结果的第 2 行到最后一行给出了产生错误的根本原因：在程序中定义 getenv(const char * name)的返回类型是$tainted，然后赋给变量 s，又赋给变量 t，最后输出打印。而对于输出打印的数据，则要求是$untainted 类型，这样就得出了$tainted<=$untainted。那么在数据流中从不安全的数据传向安全的数据，就使得安全的数据也变得不安全了，这是极其危险的操作。

Oink 除了可以进行上面的简单分析外，还可以对多态类型提供支持。例如如图 3-11 所示的程序。

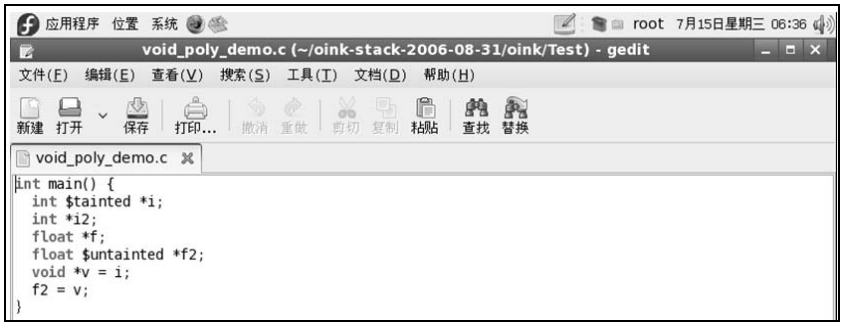

图 3-11　程序示例 4

图 3-11 的代码先将 int 型指针 i 赋给 void 型指针 v，再将 void 型指针 v 赋给 float 型指针 f2。运行“make qual-check-demo-void-poly”之后将会出现如图 3-12 所示的运行结果。

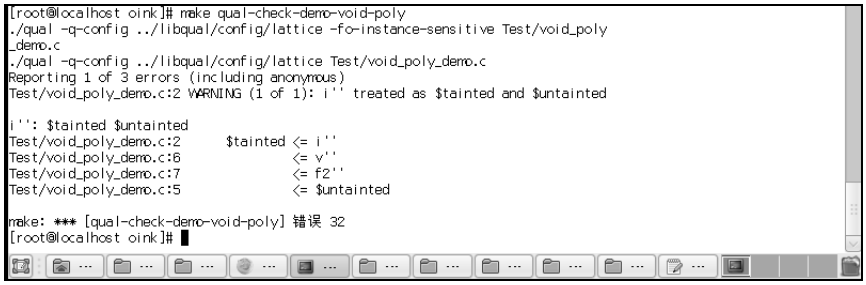

图 3-12　程序示例 4 的分析结果

由图 3-12 可以看出，程序运行结果的第 1 行，即“i'':$tainted $untainted”发生错误。这个错误和示例 3 中的错误是一样的，都得出了一个变量既是安全的又是不安全的结果。我们知道在示例 4 中，定义 i 为$tainted 类型、f2 为$untainted 类型。然后把 i 赋给 v，又把 v 赋给 f2，那么数据流向就从不安全数据流向了安全数据，从而导致安全数据变得不安全了，这同样是数据流中极其危险的操作。

Oink 除了可以进行格式字符串缺陷分析之外，还可以找出代码中的 user-kernel 缺陷。内核必须保证用户代码中的指针所指的内存是合法的，但不允许用户代码操纵内核数据，否则代码也将是不安全的。Oink 的开发者们使用 Oink 对 Linux 内核源码进行了 user-kernel

分析，从 2.4.20 版的内核中找到 7 个 bug 和 275 个 false pos，从 2.4.23 版的内核中找到 6 个 bug 和 264 个 false pos。

3.3　Eclipse PTP/CDT 程序理解工具

Eclipse CDT 是 Eclipse 插件，它把 Eclipse 转换为功能强大的 C/C++ IDE。它被设计为将 Java 开发人员喜爱的许多 Eclipse 优秀功能提供给 C/C++开发人员，例如项目管理、集成调试、类向导、自动构建、语法着色和代码完成等。当 Eclipse 被用作 Java IDE 时，它将利用 JDK 并与之集成。同样地，CDT 将利用标准的 C/C++工具并与之集成，例如 g++、make 和 GDB，这使得 CDT 在 Linux 中变得非常流行。这些工具都可在 Linux 中使用并用于大多数 C++开发。可以在 CDT 的基础上安装 PTP，以便通过 PTP 进行更好的静态分析。

3.3.1　PTP/CDT 介绍

1. CDT 介绍

CDT 是完全用 Java 实现的开源项目(根据 Common Public License 特许的)，它作为 Eclipse SDK 平台的一组插件。这些插件将 C/C++透视图添加到 Eclipse 工作台(Workbench)中，现在后者可以用许多视图和向导以及高级编辑和调试支持，来支持 C/C++开发。

由于 CDT 的复杂性，CDT 被分成几个组件，它们均采用独立插件的形式。每个组件都作为一个独立自主的项目进行运作，有它自己的一组提交者、错误类别和邮件列表。但是，所有插件都是 CDT 正常工作所必需的。下面是 CDT 插件/组件的完整列表。

(1) 主 CDT 插件(Primary CDT plug-in)：是“框架”CDT 插件。

(2) CDT 功能 Eclipse(CDT Feature Eclipse)：是 CDT 功能组件(Feature Component)。

(3) CDT 核心(CDT Core)：提供了核心模型(Core Model)、CDOM 和核心组件(Core Component)。

(4) CDT UI：是核心 UI、视图、编辑器和向导。

(5) CDT 启动(CDT Launch)：为诸如编译器和调试器之类的外部工具提供了启动机制。

(6) CDT 调试核心(CDT Debug Core)：提供了调试功能。

(7) CDT 调试 UI(CDT Debug UI)：为 CDT 调试编辑器、视图和向导提供了用户界面。

(8) CDT 调试 MI(CDT Debug MI)：是用于与 MI 兼容的调试器的应用程序连接器。

CDT 的开发有两个主要目标：一是在 Eclipse 上提供一个用来进行 C/C++开发的平台；二是提供支持 GNU 工具链的 API，例如支持 GNU make-based build systems、GCC/G++ compilers、GDB debugger 等。

其中用来进行源代码静态分析的信息存储在 CDT 核心之一——CDOM 中。CDOM 中有一个文件对象模块，在这个模块中会产生 DOM AST(Abstract Syntax Tree)，所有进行静态分析的信息均是通过 AST 来产生的，如图 3-13 所示。

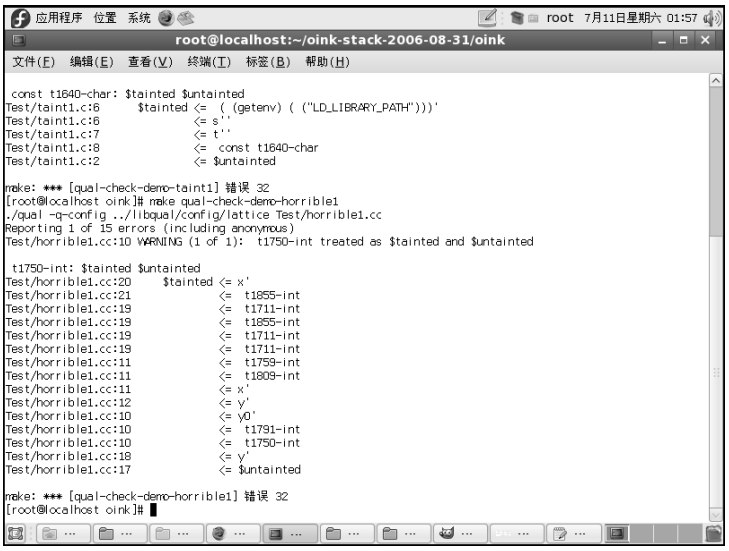

图 3-13　AST 信息输出

2. PTP 介绍

PTP(Parallel Tools Platform)作为 Eclipse 的插件，其开发目的是为了生产出一个开源的工业化的平台，特别是为并行应用开发提供一个高度整合的环境。PTP 插件有以下 4 个主要领域。

(1) 运行时的工具，使开发人员能够检测和控制执行并行应用程序。

(2) 调试工具，用于查找运行并行应用程序中的错误。

(3) 分析工具，提供先进的编辑、错误检查，帮助程序员开发并行应用程序。

(4) 性能工具，对并行应用程序进行性能分析和优化。

其中作为分析工具，PTP 在 CDT 的基础上利用抽象语法树进行静态分析，提供了一系列的高级编辑和错误检查功能。这也是用 PTP 来进行静态分析的原因。

3.3.2　PTP 环境建立

目前，PTP 的最新版本是 PTP 7.0.7；官方网址是 http://www.eclipse.org/ptp。在这里，仍用早期的 PTP 2.1 版，读者也可尝试用新版本。

在安装 PTP 之前需要安装以下软件。

(1) Eclipse：这里将使用 Eclipse 的插件 CDT。

(2) Java Runtime Environment：这里要构建 C++应用程序，而且要使用 Eclipse。而 Eclipse 本身是 Java 应用程序，因此也需要安装 Java Runtime Environment(JRE)。在这里使用 Eclipse V3.2，它要求使用 JRE V1.4 或更高版本。如果还需要使用 Eclipse 进行 Java 开发，则需要安装 Java Development Kit(JDK)。

(3) Eclipse C/C++ Development Toolkit(CDT)：PTP 是在 CDT 的基础上进行安装的，因此当然需要安装 CDT。

(4) Cygwin/MinGW：如果要使用 Microsoft Windows，则 Cygwin /MinGW 十分有用，Cygwin/MinGW 在 Windows 中提供了类似 Linux 的环境。

(5) GNU C/C++ Development Tools：CDT 将使用标准的 GNU C/C++工具来编译代码、构建项目和调试应用程序。这些工具包括 GNU Compiler Collection(GCC)for C++ (g++)、make 和 GNU Project Debugger(GDB)。如果读者是使用 Linux 或 Mac OS X 的程序员，则可以将这些工具安装到计算机上。本节包含针对 Windows 设置这些工具的说明。

PTP 既可以安装在 Windows 上，也可以安装在 Linux 上，下面分两种方式来介绍 PTP 环境的建立。

表 3-1 是安装 PTP 的系统配置要求列表(PTP 版本：2.1)。

表 3-1　PTP 安装要求列表

Component	OS(Eclipse)	OS(Server)	Java	Eclipse	CDT	RSE	MPI	Others
PTP	Linux	Linux	1.5 或更高版本	3.4.1	5.0.1 或更高版本	3.0	Open MPI 1.2.x 或 1.3.x	GDB 6.3～6.8
	Mac OS X	Mac OS X					MPI CH2.1.0.6p1	
	Windows	UNIX					IBM 并行环境	

1. Linux 下 PTP 环境的建立

在 Linux 下安装 PTP 有以下几个基本步骤。

1) Install Java

首先下载 jre-1_5_0_09-linux-i586-rpm.bin。

(1) 新建一个 Java 的目录：

```
[root@localhost~]#mkdir /usr/local/java
```

(2) 将下载的 jre-1_5_0_09-linux-i586-rpm.bin 放到/usr/local/java 目录下。

(3) 进入超级用户模式：

```
[root@localhost~]#su
```

(4) 进入 Java 目录：

```
[root@localhost ~]#cd /usr/java
```

(5) 更改 jre-1_5_0_09-linux-i586-rpm.bin 的权限为可执行：

```
[root@localhost java]#chmod a+x jre-1_5_0_09-linux-i586-rpm.bin
```

(6) 启动安装过程：

```
[root@localhost java]#./jre-1_5_0_09-linux-i586-rpm.bin
```

(此时将显示二进制许可协议，按空格键显示下一页，阅读完许可协议后，输入“yes”继续安装，如图 3-14 所示。此时会将 JRE 解压缩，产生 jre-1_5_0_9-linux-i586.rpm，如图 3-15 所示。)

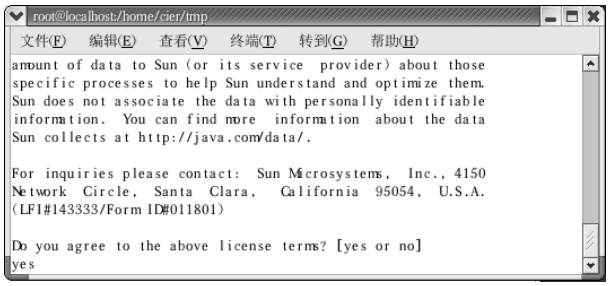

图 3-14　阅读 Sun 有关协议

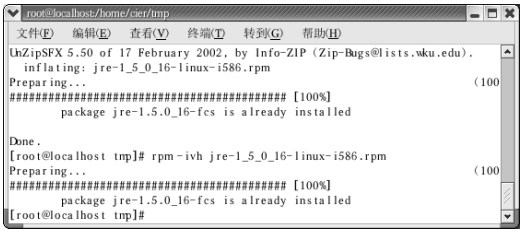

图 3-15　解压缩 JRE

(7) 安装 jre-1_5_0_09-linux-i586-rpm：

[root@localhost java]#rpm –ivh jre-1_5_0_9-linux-i586.rpm

(此时会将 JRE 安装在/usr/java/jre1_1_5_0_09 目录下。)

(8) 设定环境变量，让 Linux 能找到 JRE：

[root@localhost java]#vi /etc/profile

将以下内容添加到文件后面：

PATH=$PATH:/usr/java/jre1.5.0_09/bin
export JAVA_HOME=/usr/java/jre1.5.0_09
export CLASSPATH=$JAVA_HOME/lib:.

(存盘后，重新启动 Linux。)

(9) 测试 Java 是否安装成功，如图 3-16 所示。

[root@localhost ~]#java –version

2) Install Eclipse

(1) 首先下载 eclipse-java-ganymede-SR1-linux-gtk.tar，可以将其放到桌面上。

(2) 进入要存放 Eclipse 的目录：

[root@localhost~]#cd /usr/loca

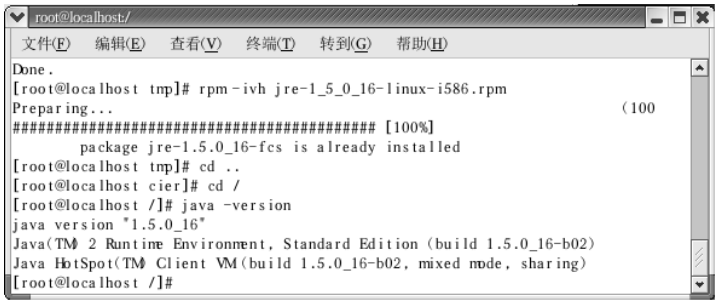

图 3-16　查看 Java 版本信息

(3) 复制 eclipse-java-ganymede-SR1-linux-gtk.tar 到当前目录：

[root@localhost~]#cp ~Desktop/eclipse-java-ganymede-SR1-linux-gtk.tar

(4) 解压缩：

[root@localhost~]#tar –zxvf /eclipse-java-ganymede-SR1-linux-gtk.tar

(5) 进入 Eclipse 目录：

[root@localhost~]#cd eclipse

(6) 执行 Eclipse：

[root@localhost~]#./eclipse

(7) 选择 Java 程序的存放目录，如图 3-17 所示。

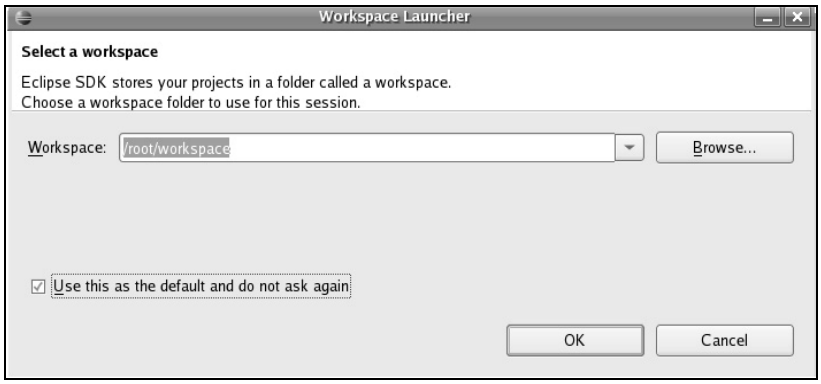

图 3-17　选择 Java 程序的存放目录

(若希望以后不出现此对话框，可选中 Use this as the default and do not ask again 复选框。)

(8) 进入 Eclipse 主界面，如图 3-18 所示。

3) Install CDT

(1) 下载 cdt-master-4.0.1，可将其放到桌面上。

(2) 进入存放 Eclipse 的目录，如图 3-19 所示。

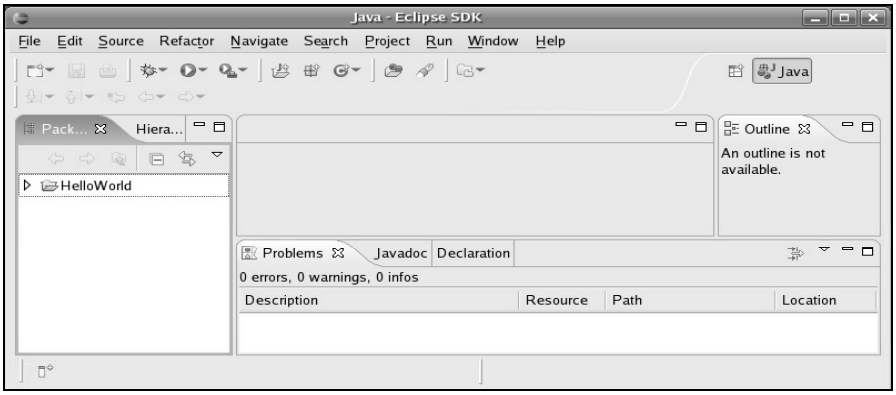

图 3-18　Eclipse 主界面

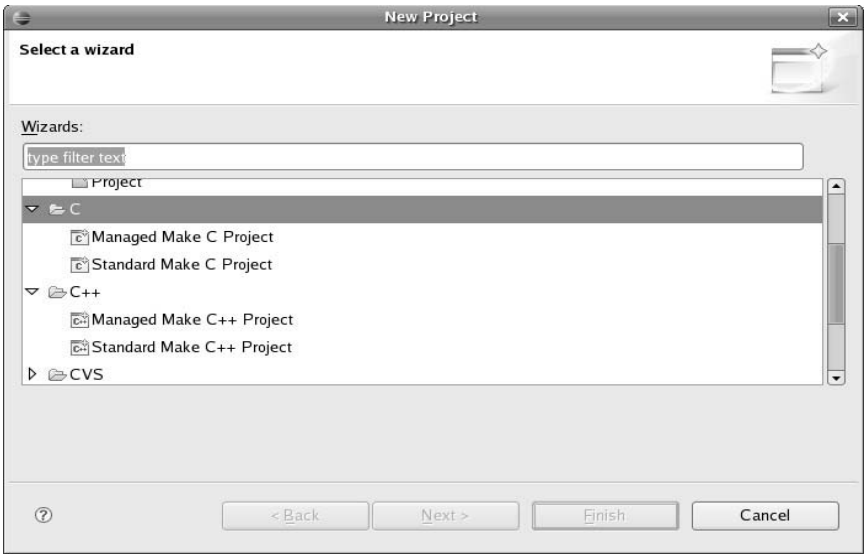

图 3-19　Eclipse 的存放目录

[root@localhost~]#cd /usr/local

(3) 复制 cdt-master-4.0.1 到当前目录：

[root@localhost~]#cp ~Desktop/CDT-master-4.0.1

(4) 解压缩：

[root@localhost~]#unzip　/CDT-master-4.0.1r

(5) 安装 CDT 插件。

将 plugins 和 festures 目录下的文件复制到 Eclipse 中相应的目录下：

[root@localhost~]#cp –r eclipse/plugins/* /usr/local/eclipse/plugins
[root@localhost~]# cp –r eclipse/features/* /usr/local/eclipse/teatures

(6) 重新启动 Eclipse，多了 C 和 C++项目支持。