



中标软件

中标麒麟高级服务器操作系统 V6.0

开发者指南

中标软件有限公司

2014年4月20日

目录

1 Eclipse 入门.....	1
1.1 认识 Eclipse 项目.....	1
1.2 Eclipse 中的帮助.....	4
1.3 开发工具包.....	6
2 Eclipse 集成开发环境(IDE).....	8
2.1 用户界面.....	8
2.2 有用提示.....	12
2.2.1 快速访问菜单.....	12
2.2.2 libhover 插件.....	18
2.2.2.1 设置与用法.....	19
3 库与运行时支持.....	21
3.1 版本信息.....	21
3.2 兼容性.....	21
3.2.1 API 兼容性.....	22
3.2.2 ABI 兼容性.....	22
3.2.3 策略.....	23
3.2.4 静态链接.....	23
3.3 库与运行时详情.....	23
3.3.1 GNU C 库.....	23
3.3.1.1 GNU C 库更新.....	23
3.3.1.2 GNU C 库文档.....	25
3.3.2 GNU C++ 标准库.....	25
3.3.2.1 GNU C++ 标准库更新.....	26
3.3.2.2 GNU C++ 标准库文档.....	27
3.3.3 Boost.....	28
3.3.3.1 Boost 更新.....	29
3.3.3.2 Boost 文档.....	30
3.3.4 Qt.....	30
3.3.4.1 Qt 更新.....	31
3.3.4.2 Qt Creator.....	31
3.3.4.3 Qt 库文档.....	32
3.3.5 KDE 开发框架.....	32
3.3.5.1 KDE4 架构.....	32
3.3.5.2 kdelibs 文档.....	34

3.3.6 NSS 共享数据库	34
3.3.6.1 反向兼容	34
3.3.6.2 NSS 共享数据库文档	35
3.3.7 Python	35
3.3.7.1 Python 更新	35
3.3.7.2 Python 文档	35
3.3.8 Java	36
3.3.8.1 Java 文档	36
3.3.9 Ruby	37
3.3.9.1 Ruby 文档	37
3.3.10 Perl	37
3.3.10.1 Perl 更新	38
3.3.10.2 安装	38
3.3.10.3 Perl 文档	38
4 编译与组建	41
4.1 GNU 编译器集合(GCC)	41
4.1.1 GCC 状态与特性	41
4.1.2 语言兼容性	42
4.1.3 对象兼容性与互操作性	43
4.1.4 反向兼容性包	44
4.1.5 预览 LINUX5 中的 LINUX6 编译器功能	45
4.1.6 运行 GCC	45
4.1.6.1 简单的 C 用法	45
4.1.6.2 简单 C++ 用法	46
4.1.6.3 简单多文件用法	46
4.1.6.4 推荐的优化选项	47
4.1.6.5 使用配置文件反馈来调整启发式优化算法	49
4.1.6.6 在 64 位机器上使用 32 位编译器	50
4.1.7 GCC 文档	51
4.2 分布编译	52
4.3 Autotools	52
4.3.1 Eclipse 的 Autotools 插件	53
4.3.2 配置脚本	53
4.3.3 Autotools 文档	54
4.4 Eclipse 内置 Specfile 编辑器	54
5 调试	56

5.1 安装 Debuginfo 包	56
5.2 GDB	56
5.2.1 简单 GDB	57
5.2.2 运行 GDB	58
5.2.3 条件断点	60
5.2.4 Forked 执行	61
5.2.5 调试单个线程	63
5.2.6 可选的 GDB 用户界面	68
5.2.7 GDB 文档	68
5.3 作业的变量跟踪	68
5.4 Python Pretty-Printers	69
6 性能分析 Profiling	73
6.1 Eclipse 中的 Profiling	73
6.2 Valgrind	74
6.2.1 Valgrind 工具	75
6.2.2 使用 Valgrind	75
6.2.3 Eclipse Valgrind 插件	76
6.2.4 Valgrind 文档	76
6.3 OProfile	76
6.3.1 OProfile 工具	77
6.3.2 使用 OProfile	77
6.3.3 Eclipse OProfile 插件	78
6.3.4 OProfile 文档	79
6.4 SystemTap	79
6.4.1 SystemTap 编译服务器	80
6.4.2 对非特权用户的 SystemTap 支持	80
6.4.3 SSL 与证书管理	81
6.4.3.1 用于连接的编译服务器授权	81
6.4.3.2 用于模块签名（对非特权用户）的编译服务器授权	81
6.4.3.3 自动授权	81
6.4.4 SystemTap 文档	82
6.5 Eclipse-Callgraph	82
6.5.1 启动带有 Eclipse-Callgraph 的 Profile	83
6.5.2 Callgraph 视图	84
6.6 Linux (PCL) 工具性能计数器和 perf	87
6.6.1 Perf 工具命令	88

6.6.2 使用 Perf.....	88
6.7 ftrace	90
6.7.1 使用 ftrace	90
6.7.2 ftrace 文档	91

1 Eclipse 入门

Eclipse 是一个强大的开发环境，它提供了开发过程的每个阶段所需的工具。为了方便使用，它集成了单一、可全面配置的用户界面，允许以多种方式扩展的可插拔架构。

Eclipse 将多种不同工具集成到统一环境中，提供非常丰富良好的开发体验。例如，Valgrind 插件允许程序员通过 Eclipse 用户界面执行内存监控(通常 Valgrind 操作由命令行完成)。该功能不是 Eclipse 自带的。

作为图形应用，Eclipse 是非常受开发人员欢迎的选择，这些开发人员发现命令行界面不方便使用。另外，Eclipse 内置的帮助系统为每个集成的特性和工具提供大量文档。这极大缩短了初级开发人员从入门到精通的时间。

传统的(即多数基于命令行的)Linux 工具包(gcc、gdb 等)的编程方法比起 Eclipse 提供的编程方法是截然不同的。多数传统的 Linux 工具都很灵活精致，而且(总体上)比它们集成在 Eclipse 的部分都更加强大。但是另一方面，这些传统的 Linux 工具也更难于掌握，它们给多数程序员或项目提供更多所需的强大功能。所以，Eclipse 为了支持集成环境牺牲了一些优点，这很适合那些喜欢在单个图形界面中使用工具的用户。

1.1 认识 Eclipse 项目

Eclipse 在指定的工作区中存储了所有项目和用户文件。您可拥有多个工作区，并可随意切换。然而，Eclipse 只能从当前活动工作区加载项目。若要在活动工作区之间切换，可使用【文件】=>【切换工作空间】进行工作空间切换。还可通过【工作空间启动程序】向导增加新的工作区；若要打开该向导，可选择：【文件】=>【切换工作区间】=>【其他】。

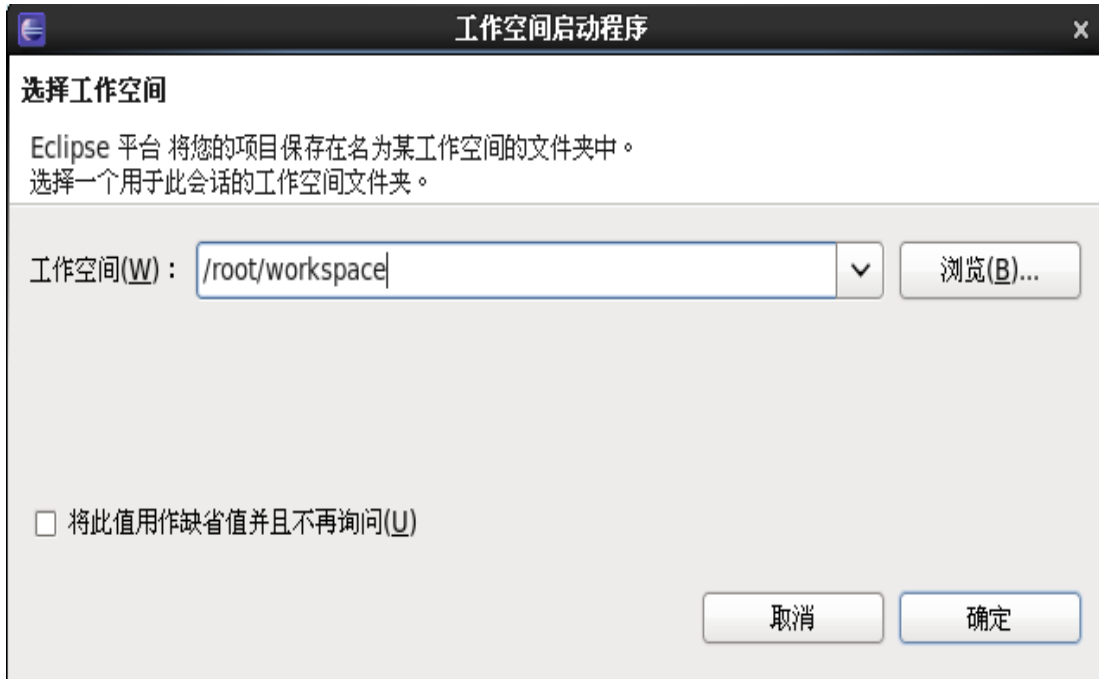


图 1-1 工作空间启动程序

若项目包含必要的 Eclipse 元文件，则它可被直接导入 Eclipse。Eclipse 使用这些文件来决定透视图、工具和其他用户界面配置的实施类型。

同样，当试图导入从未在 Eclipse 上运行过的项目时，可能有必要通过【新建项目】（【文件】→【新项目】）向导，而非【导入】向导来完成。这样做将为项目创建必要的 Eclipse 元文件，当您提交该项目时，也可以包含这些元文件。

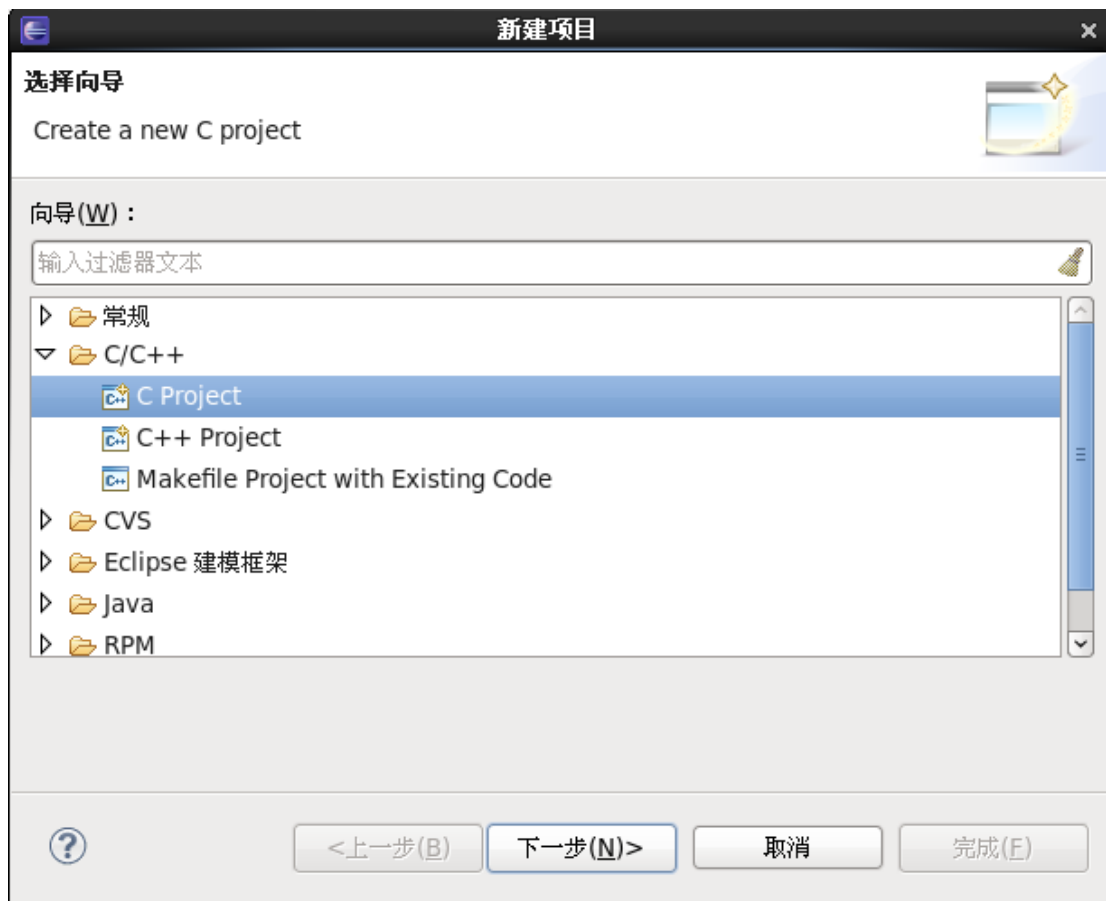


图 1-2 新建项目向导

【导入】向导主要适用于在 Eclipse 中创建或之前编辑过的项目，也就是包含必要 Eclipse 元文件的项目。



图 1-3 导入向导

1.2 Eclipse 中的帮助

Eclipse 的特色是它拥有内部的综合帮助库，该帮助库几乎覆盖了集成开发环境(IDE)的每个方面。每个 Eclipse 文档插件都将内容安装到该库中。在库中为这些内容建立相应的索引。若要访问该库，请使用【帮助】菜单。



图 1-4 帮助

若要打开主帮助菜单，请导航至【帮助】→【帮助内容】。该【帮助】菜单显示了【内容】字段中已安装的文档插件提供的所有可用内容。

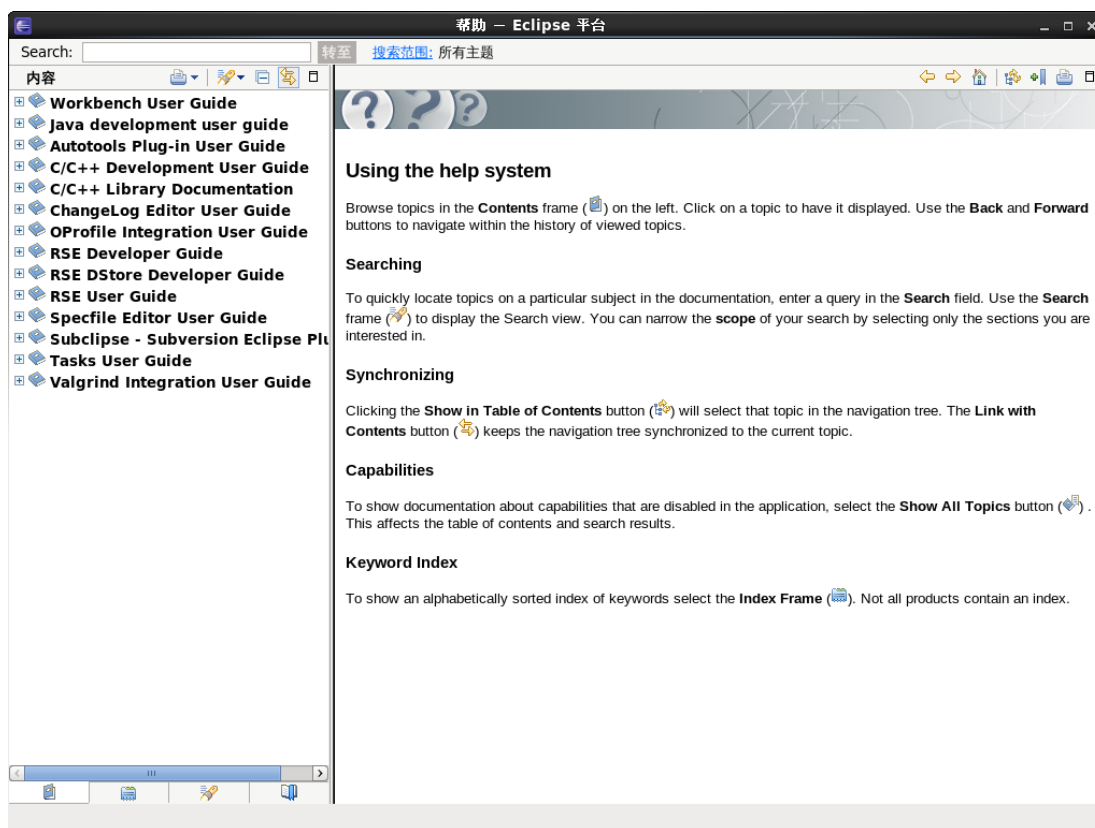


图 1-5 帮助菜单

【内容】 字段底部的选项卡提供了访问 Eclipse 文档的不同选项。您可以通过每本手册按章节或标题进行浏览，

或通过 Search 字段简单搜索进行浏览。还可在每本手册的章节上做标签并通过 Bookmarks 选项卡进行访问。

Workbench User Guide 广泛记载了 Eclipse 用户界面的每个方面。它包含关于 Eclipse 工作区、透视图和不同概念的非常底层的信息，这些信息对于理解 Eclipse 的工作原理是非常有用的。总的来说，Workbench User Guide 对于几乎不具备 Eclipse 或 IDE 中等经验的用户来说是理想资源。该文档插件是默认安装的。

Eclipse 帮助系统也包括动态帮助特性。此特性在工作区中打开一个新窗口，这个窗口显示了与选定界面元素相关的文档。若要激活动态帮助，请浏览**【帮助】** → **【动态帮助】**。

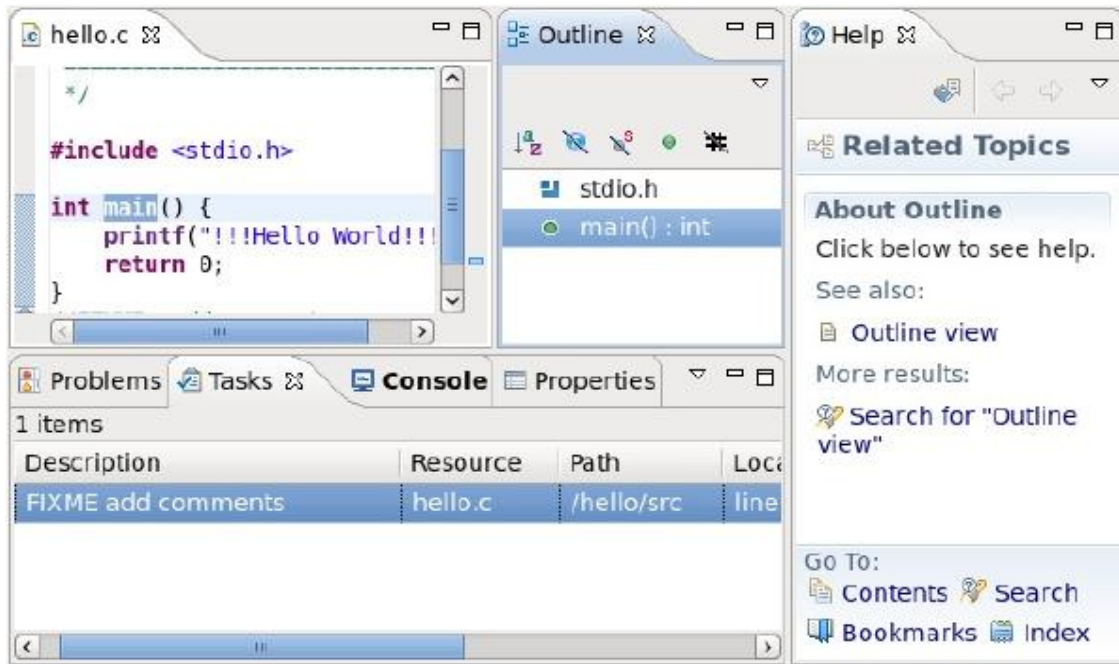


图 1-6 动态帮助

图 1-6 动态帮助中最右面的窗口显示了与**【Outline】**视图相关的帮助主题。

【Outline】视图是选定的用户界面元素。

1.3 开发工具包

C/C++(CDT)和 Java(JDT)是两个主要的 Eclipse 开发工具包。这些工具包提供了一套针对他们各自语言的集成工具。这两个工具包提供为 Eclipse GUI 界面提供用于编辑、组建、运行和调试源代码的所需工具。

每个工具包为其各自语言提供了定制的编辑器。CDT 和 JDT 也为项目中所使用的多种文件类型提供了多个编辑器。例如，CDT 提供了针对 C/C++ 头文件和源文件的不同编辑器，以及 Makefile 编辑器。

所提供的工具包编辑器为一些文件类型（不需组建）提供错误解析，但是这对存在文件相互关联的项目来说是不可用的。

例如，CDT 源文件编辑器提供在单一文件上下文中进行错误解析的功能，但只有在组建完整项目时，有些错误才可见。工具包提供的编辑器的其他常见特性是着色、代码折叠和自动化对齐格式。另外，其他插件提供先进的编辑器功能，例如，自动代码完成、悬浮帮助和上下文搜索；这种插件的一个很好的例子就是 libhover，它将这些扩展功能添加到 C/C++ 编辑器中。

（更多信息请参阅“2.2.2 libhover 插件”）

创建一个项目目标（binary 文件、库等）的多数（并不是所有的）步骤的用户界面是由每个工具包的组件功能提供的。每个工具包也为 Eclipse 提供使组建过程尽可能多的实现自动化的方法，帮助您集中更多精力编写代码，较少精力用于组建它。其中有两个工具包还为查找代码中的问题添加有用的 UI 元素，例如，Eclipse 会将编译错误发送至视图。对于大多数错误类型，只需要单击视图的入口便允许您直接导航至错误的原因（文件与代码片段）。

编辑器和其他插件也提供扩展的组建项目的功能，例如，Autotools 插件允许您为 C/C++ 项目中增加可移植性，允许其他开发人员在很多不同环境中组建项目（更多信息，请参阅“4.3 Autotools”）。

对于具有可执行或二进制目标文件的项目，每个工具包还为 Eclipse 提供运行或调试功能。在多数项目中，运行只是简单地被执行为无中断的调试操作。两个工具包将视图捆绑至 Eclipse 编辑器，允许设置断点。可以触发断点打开进入编辑器代码中的相应函数。还可以通过单击代码中的变量名查找变量值。

对于一些项目，重新完善构建或者动态地打补丁也是可行的。通过这种功能，Eclipse 可在您调试会话期间编辑代码时，自动重新组建项目或安装补丁。这使得调试-改正过程更加流畅，这是开发人员所喜欢的。

Eclipse 帮助菜单为 CDT 和 JDT 提供扩展文档。有关每个工具包的详细信息，请参阅 Eclipse 帮助内容中的 Java Development User Guide 或 C/C++ Development User Guide。

2 Eclipse 集成开发环境(IDE)

图 2-1Eclipse 用户界面(默认)中的整个用户界面被称为 Eclipse 工作区。它通常由代码编辑器、项目资源管理器窗口和几个视图组成。Eclipse 工作区中的所有元素都是可配置的，并在 Workbench User Guide(帮助内容)中有完整的文档描述。有关定制用户界面的简要概述，请参阅“2.2 有用提示”。

Eclipse 可以展现不同的透视图。一个透视图是对特定类型的任务和项目非常有用的一组视图和编辑器；Eclipse 工作台可包含一个或更多透视图。图 2-1“Eclipse 用户界面(默认)”展现了 C/C++的默认透视图。

Eclipse 也将许多功能分为几类，封装在不同的菜单项中。例如，菜单封装了与编译/组建项目有关的功能。菜单包含了创建和定制透视图、菜单项及其他界面元素的选项。有关每个主菜单项的简要概述，请参阅 C/C++ Development User Guide 中的 Reference> C/C++ Menubar 或 Java Development User Guide 中的 Reference> Menus and Actions。

以下章节提供了 Eclipse 集成开发环境(IDE)默认用户界面中可见的不同元素的更高层次概述。

2.1 用户界面

Eclipse 工作台为开发过程中的每个阶段提供了所需特性和工具的用户界面。本节提供了 Eclipse 的主要用户界面的概述。

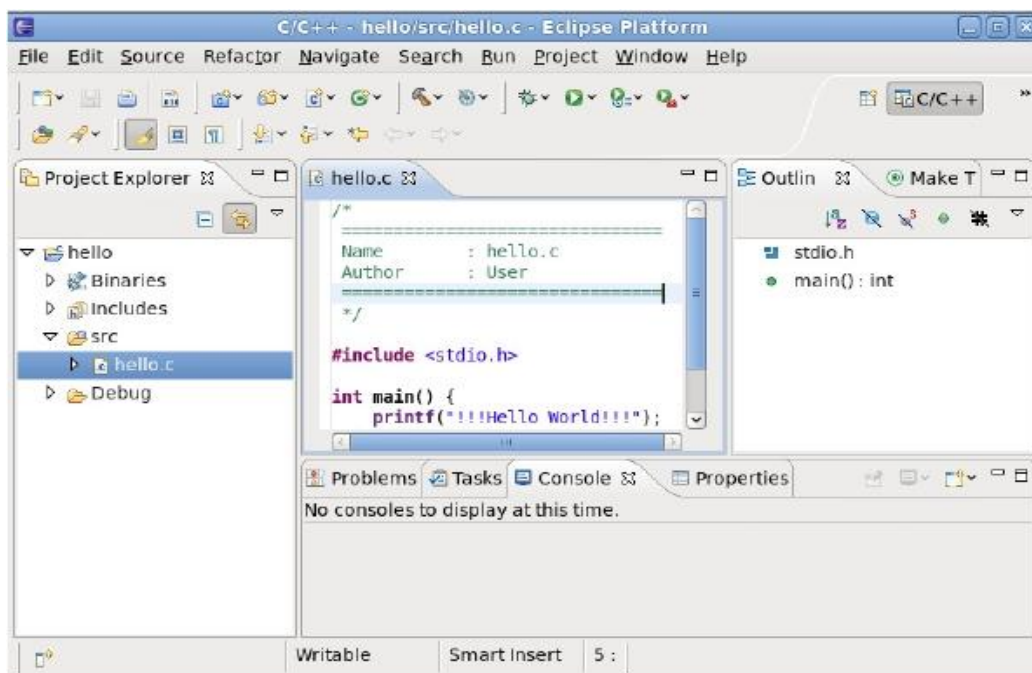


图 2-1 Eclipse 用户界面（默认）

图 2-1Eclipse 用户界面(默认)显示了 C/C++项目的默认工作台。若要在工作台中不同透视图间进行切换，请按 **Ctrl+F8**。有关透视图定制的提示信息，请参考 2.2 有用的提示信息。下图描述了默认 C/C++透视图中的每个基本元素。



图 2-2 Eclipse 编辑器

【编辑器】 用于编写和编辑源文件。Eclipse 能自动检测并为多数类型的源文件加载合适的语言编辑器（例如，对以.c 为后缀名的文件提供 C 编辑器）。若要配置**【编辑器】**的设置，请导航到**【窗口】→【首选项】→【语言】(e.g.Java,C++) →【Code Style】**。

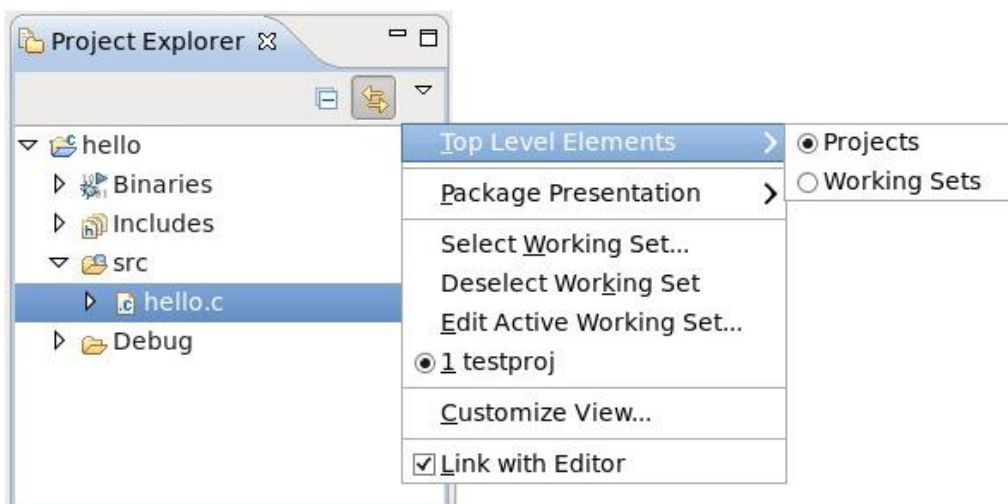


图 2-3 项目资源管理器

【项目资源管理器】 给出了所有项目资源（二进制文件、源文件等）的层次

视图。您可从该视图打开、删除，或编辑所有文件。

【资源管理器】 视图中的 **【视图菜单】** 按钮允许您配置项目或工作集是否为 **【资源管理器】** 视图中的顶层项目。工作集是被强制分为单个集合的一组项目，使得工作集在组织相关或相关联项目时更加便利。

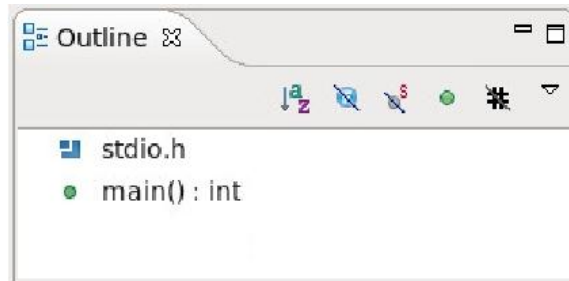


图 2-4 Outline 窗口

【Outline】 窗口提供了源文件中代码的简要框架。它详细说明了编辑器中所选文件不同变量、函数、库和其他结构化元素，它们全部都是面向特定编辑器的。

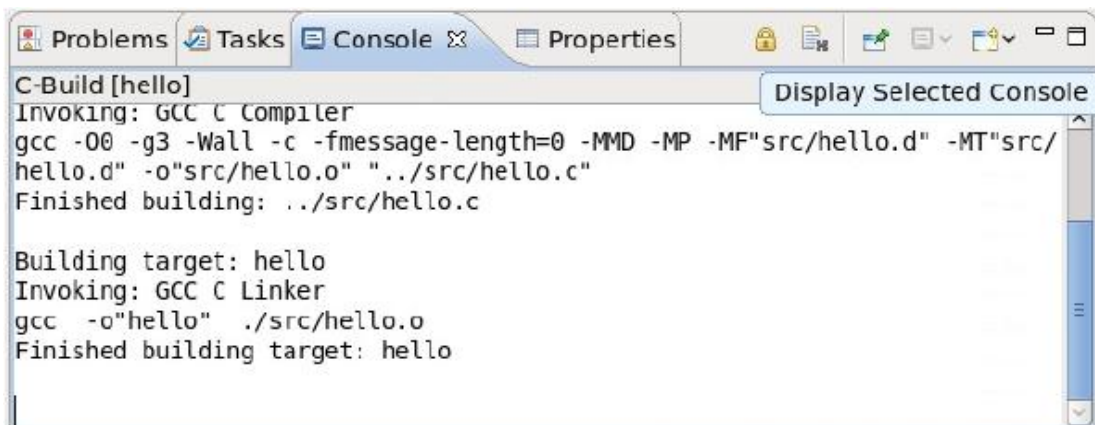


图 2-5 控制台视图

Eclipse 中的一些函数和插件程序将它们的输出发送到 **【控制台】** 视图。该视图的 **【打开控制台】** 按钮可在不同控制台之间进行切换。

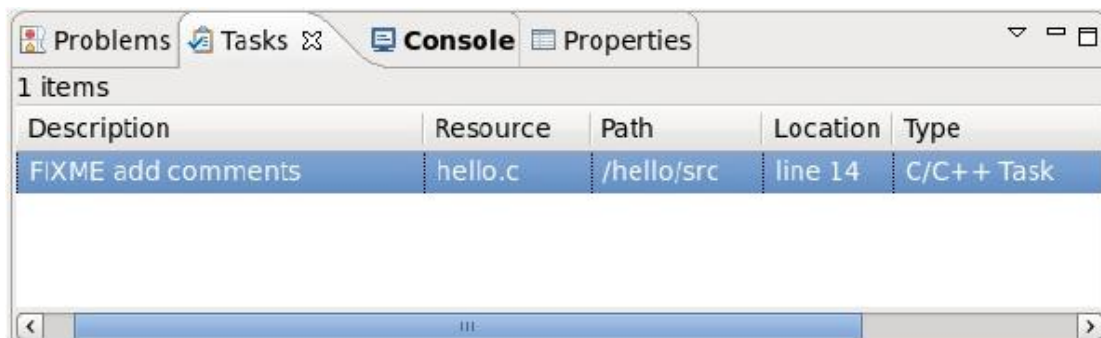


图 2-6 任务视图

【任务】视图可追踪代码中特别标注的提醒注释。该视图可显示每个任务注释的位置，允许您以多种方式对它们进行排序。



图 2-7 跟踪注释示例

多数 Eclipse 编辑器跟踪标注为//FIXME 或//TODO 标签的注释。可以跟踪注释不同于另一种语言编写的源文件。若要添加或配置任务标签，请导航到**【窗口】**→**【首选项】**并使用关键词 task tags 显示指定编辑器或语言的任务标签配置菜单。

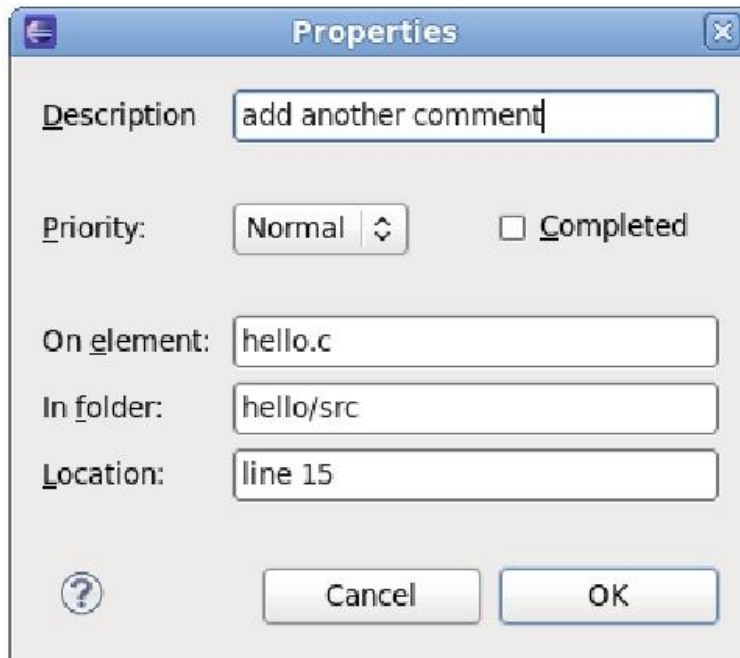


图 2-8 任务属性

您也可使用**【编辑】**→**【添加任务】**打开任务**【特性】**菜单(图 2-8 任务属性)。这样，您就可以将任务添加至源文件的特定位置，无需使用任务标签。

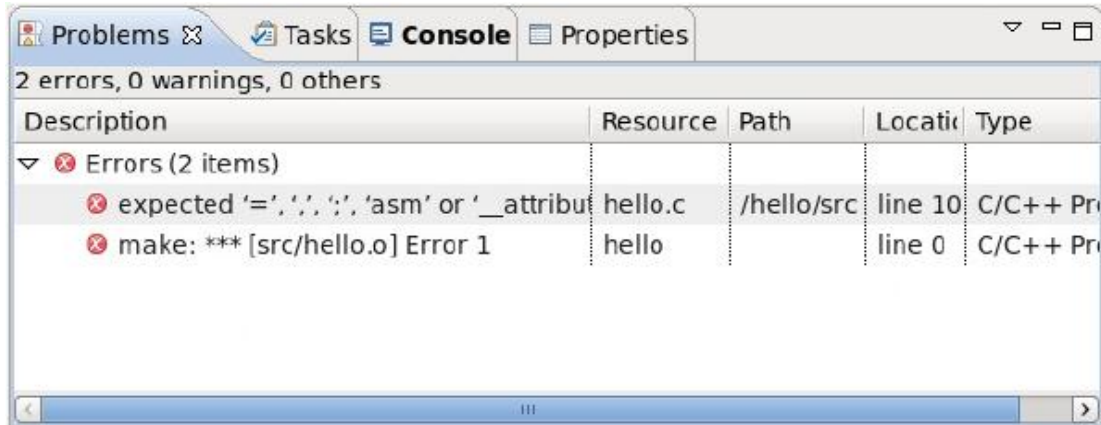


图 2-9 问题视图

【问题】视图显示特定操作（如组建、清除或配置文件运行）执行过程中出现的错误或警告。若要显示特定问题的快速纠正建议，请选中它并按下 **Ctrl+1**。

2.2 有用提示

很多 Eclipse 用户仅仅通过使用 Eclipse 用户界面的经验来学习高效的操作和故障排除技巧。本节提供了一些更有用的提示，初学者可能对此很感兴趣。Workbench User Guide 中的章节“技巧与诀窍”包含更多 Eclipse 技巧的扩展信息。

2.2.1 快速访问菜单

最有用的 Eclipse 技巧之一就是使用 **【快速访问】** 菜单。在菜单中键入一个单词，将显示一系列视图、命令、帮助文件以及与该词相关的其他操作。若要打开该菜单，请按下 **Ctrl+3**。

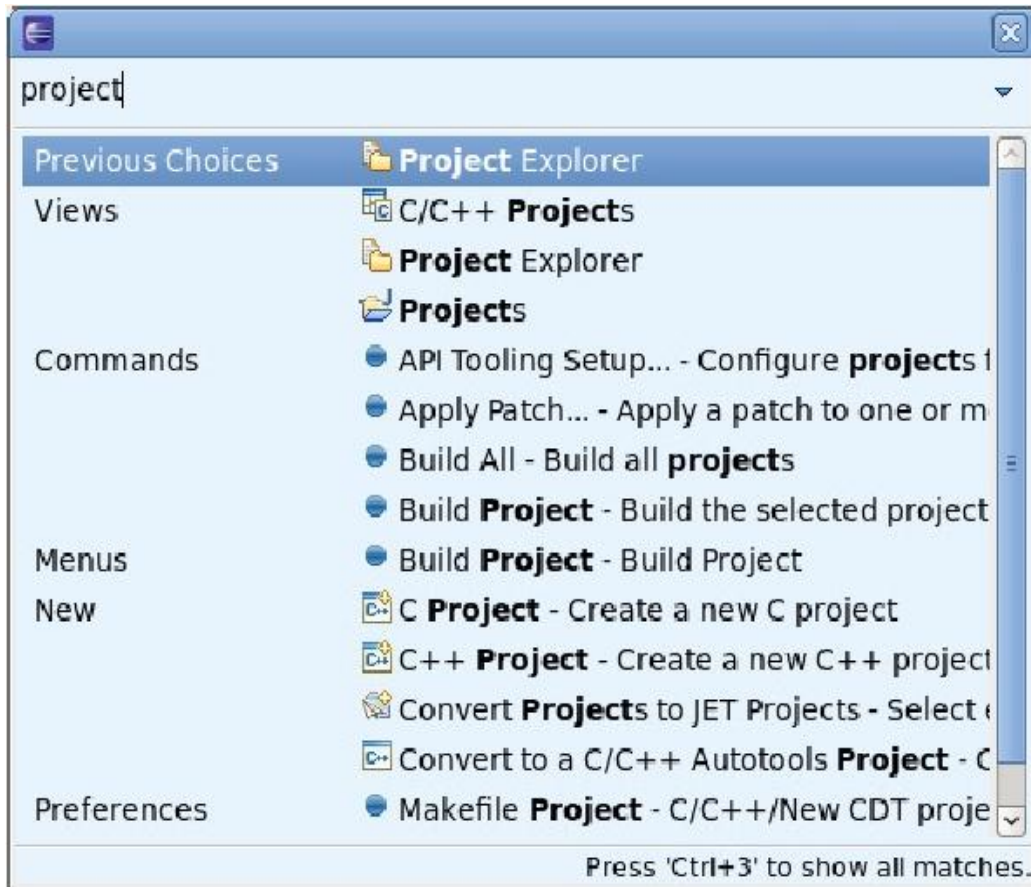


图 2-10 快速访问菜单

在图 2-10 “快速访问菜” 单中，单击【视图】→【项目资源管理器】将选择项目资源管理器窗口。单击【命令集】→【菜单】→【新建】，或【首选项】→【类】中的任何一项来运行所选项目。这类似于导航到或单击各自菜单选项或任务栏上的图标。您也可使用方向键浏览【快速访问】菜单。

还可查看所有快捷键命令的列表，请您按下 Shift+Ctrl+L。

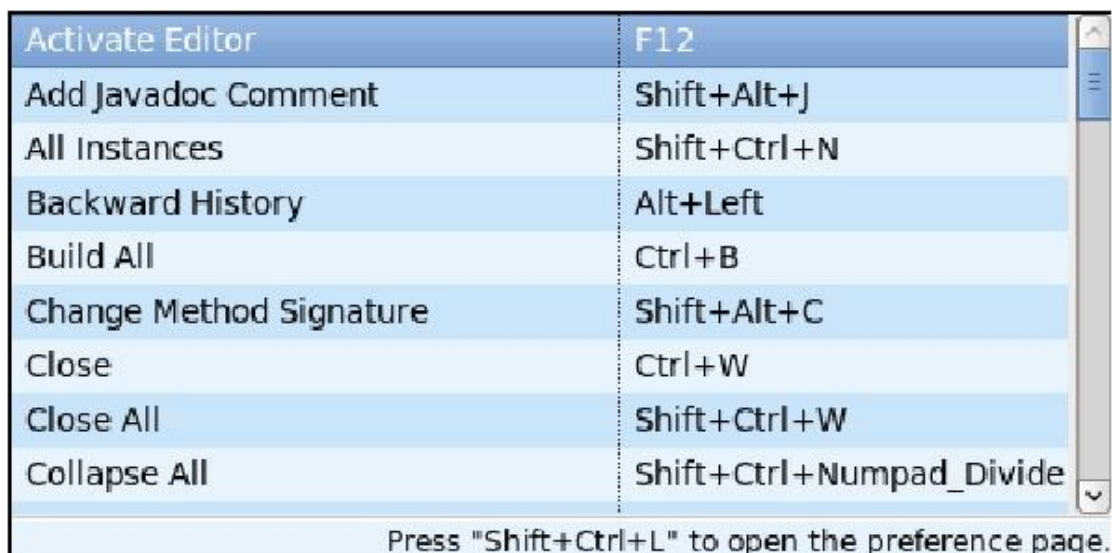


图 2-11 快捷键

若要配置 Eclipse 快捷键，请在打开列表的同时再次按下 Shift+Ctrl+L。



图 2-12 配置快捷键

若要定制当前透视图，请导航到【窗口】→【定制透视图】。这将打开【定制透视图】菜单，您可以配置可见的工具栏、主菜单项、命令组和快捷键。

通过单击视图标题，并将其拖到所需位置，可在工作台中定制每个视图的位置。

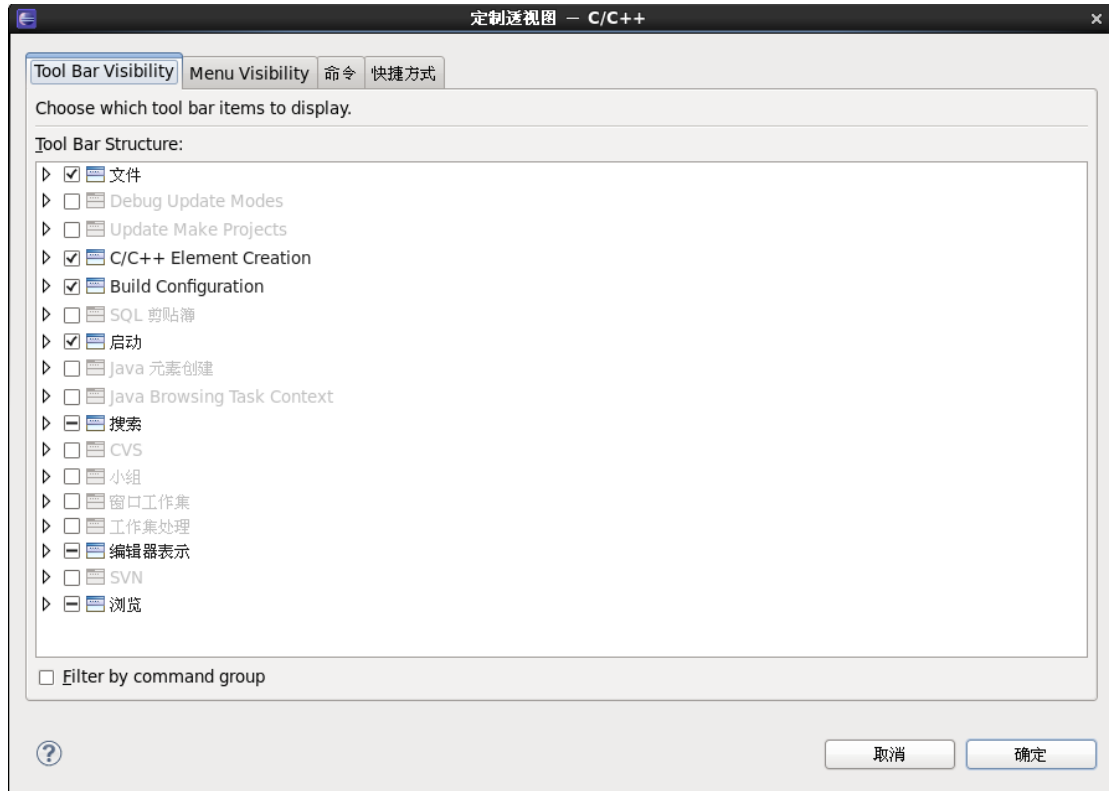


图 2-13 定制透视图菜单

图 2-13 “定制透视图菜单”显示工具栏可视性 Tool Bar Visibility 标签。正如其名，该标签可设置工具栏(图 2-14 “工具栏”)的可见性。



图 2-14 工具栏

下图显示了 Customize Perspective 菜单的其他选项卡：

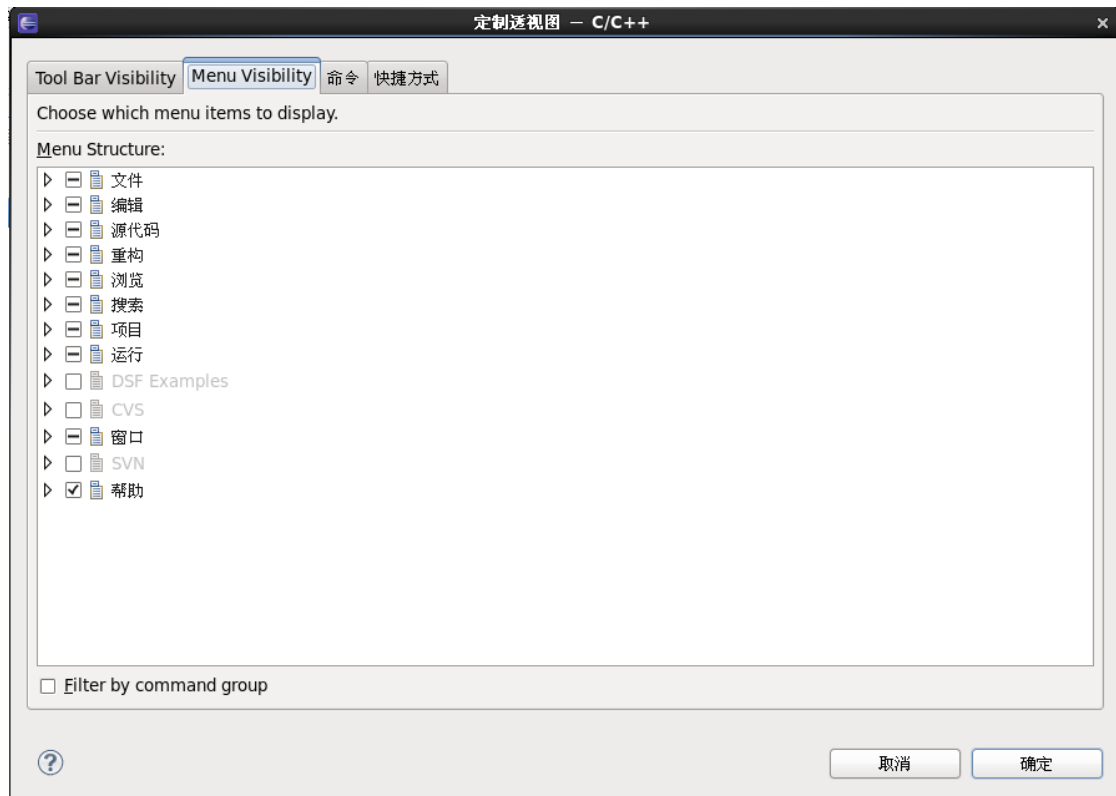


图 2-15 菜单可视性标签

【菜单可视性】标签可配置每个主菜单项中可见的功能。有关每个主菜单项的简要概述,请参阅 C/C++开发用户指南中的 Reference> C/C++ Menubar, 或 Java 开发用户指南中的 Reference> Menus and Actions。

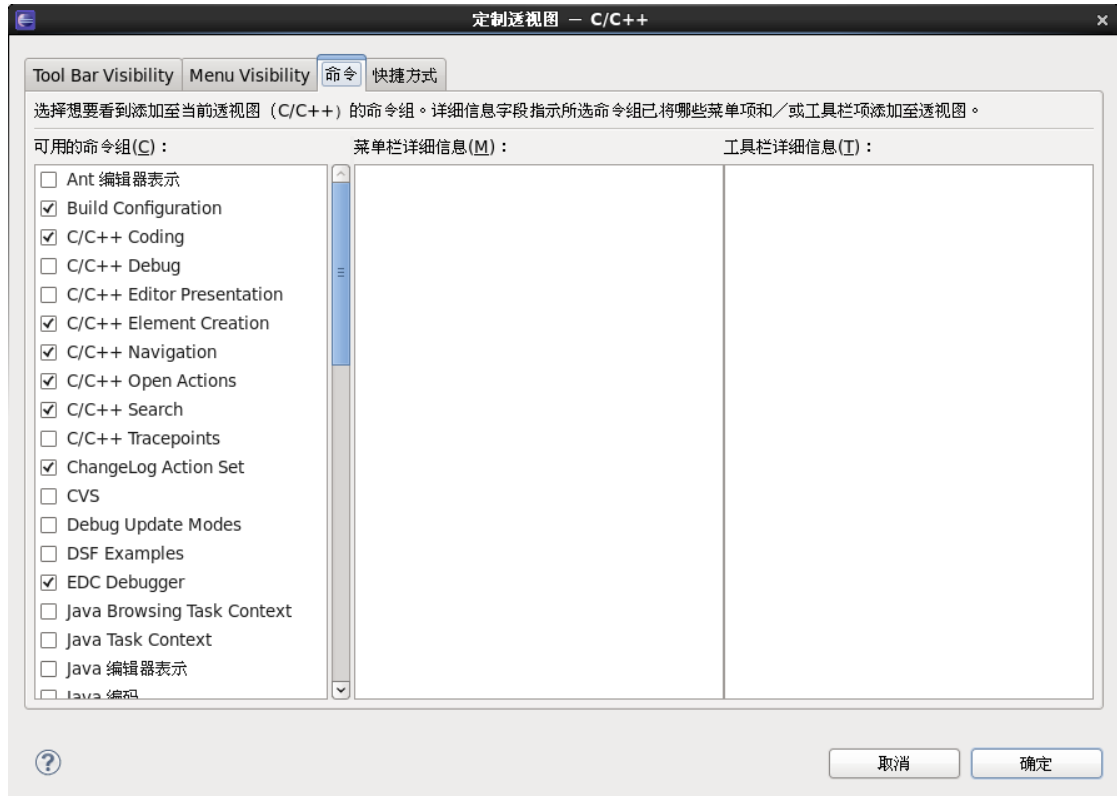


图 2-16 命令标签

命令组可将功能或选项添加至主菜单或工具栏区。使用【命令】标签可添加或删除命令组。【菜单栏详细信息】和【工具栏详细信息】分别显示命令组向主菜单或工具栏区添加的功能或选项。

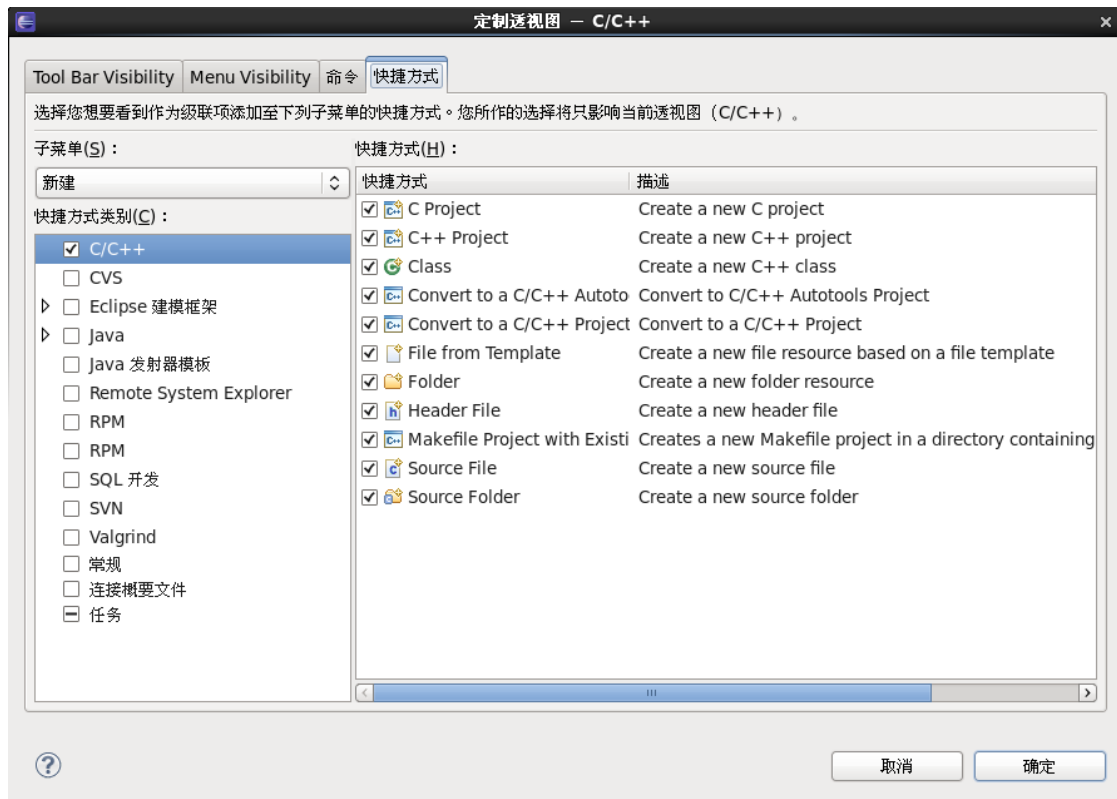


图 2-17 快捷方式选项卡

【快捷方式】选项卡可配置以下子菜单中可用的菜单项：

- 1) 【文件】→【新建】
- 2) 【窗口】→【打开透视图】
- 3) 【窗口】→【显示视图】

2.2.2 libhover 插件

Eclipse 的 libhover 插件为 GNU C 库和 GNU C++ 标准库提供即插即用的悬浮帮助支持。这允许开发人员通过悬浮帮助和代码完成更加无缝和方便地参考 Eclipse IDE 内 glibc 和 libstdc++ 库中现有的文档。

对于 C++ 库资源，libhover 需要使用 CDT 索引器索引文件。索引在整个组建中解析给定的文件；组建上下文决定了头文件的位置，以及类型、宏和相似项目的处理方法。若要索引 C++ 源文件，libhover 通常要求您首先执行实际的组建操作，即使在可能已知道头文件位置的情况下。

之所以 libhover 插件可能需要索引 C++ 源文件，这是由于 C++ 成员函数名信息不足以查找出它的文档。对于 C++，还需要类名和函数参数特征来准确确定被引用的成员。这是由于 C++ 允许不同类具有相同名称的成员，甚至在同一个类中

成员也可同名，但具有不同的方法特征。

另外，C++还有需要处理的类型定义和模板类。该信息需要解析整个文件以及相关的 include 文件，libhover 只能通过索引来完成解析。

同样，和 C++的机制类似，通过给悬浮帮助提供代码，可以实现只根据在文件中的名称来引用 C 函数，libhover 无需索引 C 源文件。只需要为选项选择所包含的相应 C 头文件。

2.2.2.1 设置与用法

默认情况下已启用所有已安装 libhover 库的悬浮帮助，可根据项目需要禁用。若要禁用或启用特定项目的悬浮帮助，可右击项目名并单击【属性】。在出现的菜单上，导航至【C/C++General】=>【Documentation】。选中或取消选中 Help books 部分的库启用或禁用特定库的悬浮帮助。

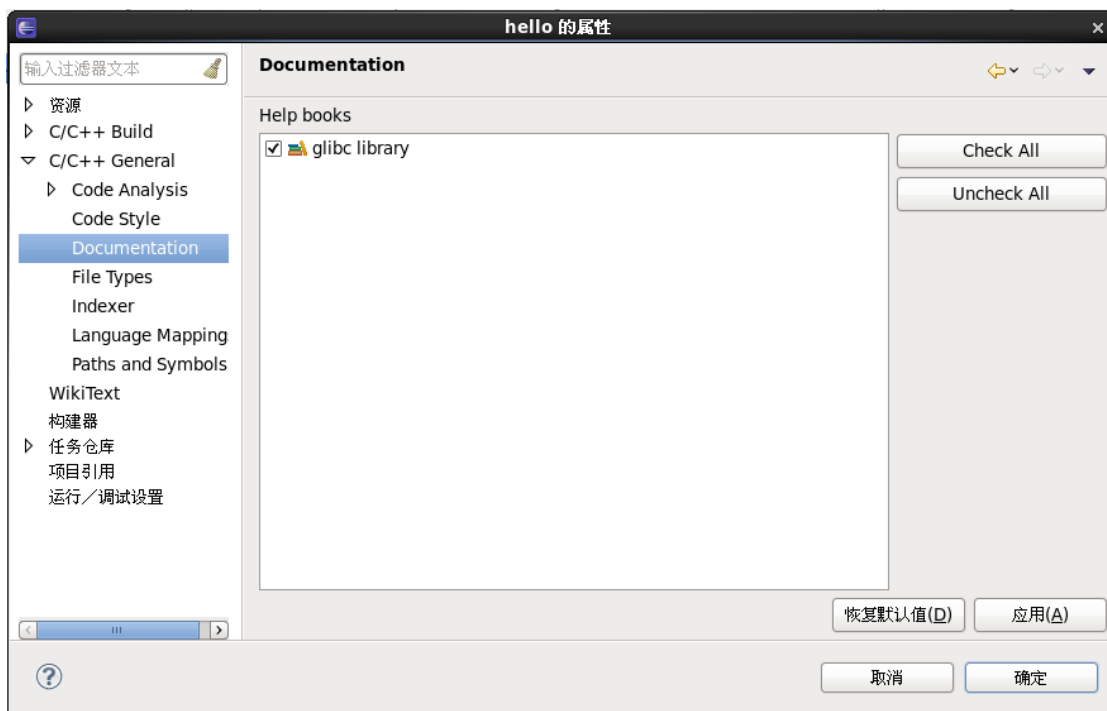


图 2-18 启用或禁用悬浮帮助

禁用特定库的悬浮帮助也许更合适，尤其当多个 libhover 库在功能上相互重叠时。例如，newlib 库包含函数名与 GNUC 库（默认提供）中函数名可能会重叠；为 newlib 和 glibc 安装 libhover 插件就意味着不得不禁用一个。

当启用多个 libhover 库，且库之间存在功能重叠时，可以从 Help books 悬浮帮助中的帮助内容来首先列出的库的函数。对于代码的完成，libhover 将提供来自所有启用 libhover 库中所有可能的选择。

若要使用悬浮帮助，只需将鼠标悬浮在 C/C++编辑器的函数名或成员函数名旁边。几秒钟后，libhover 将显示有关所选 C 函数或 C++成员函数的库文档信息。

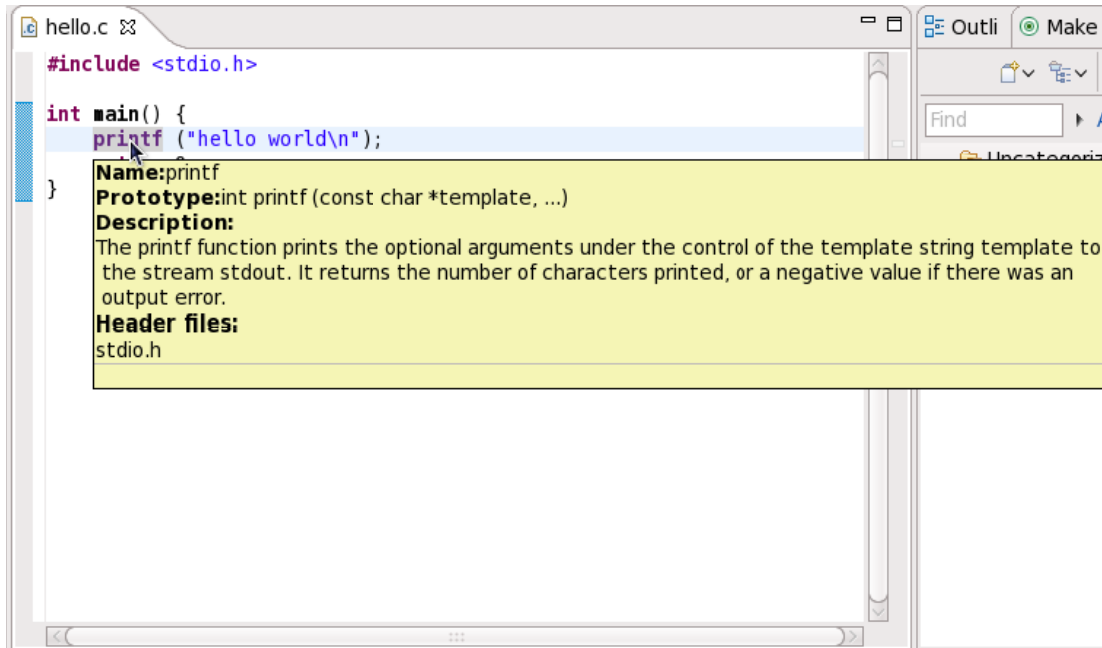


图 2-19 使用悬浮帮助

若要使用代码完成，请选择代码中的字符串并按 Ctrl+空格。给定所选字符串，将显示所有可能的函数：

单击一个可能函数查看它的描述。

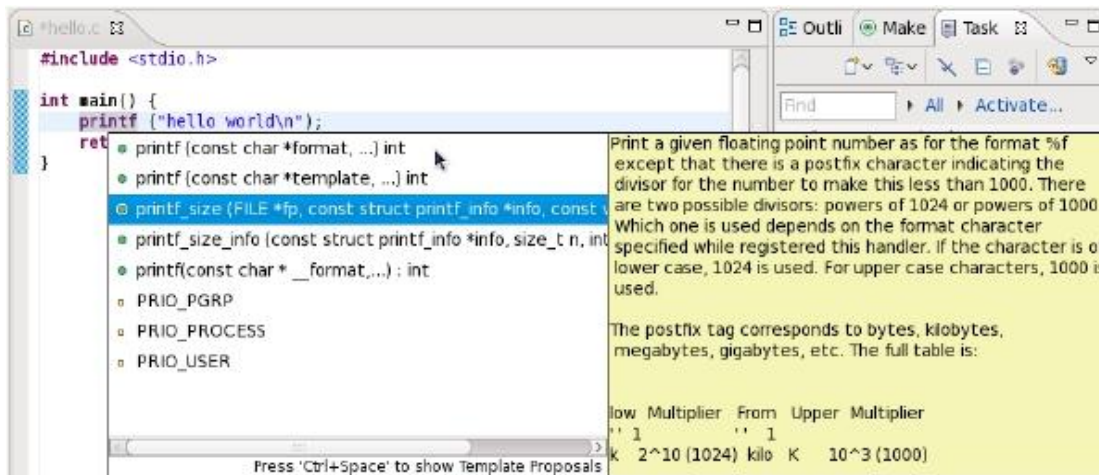


图 2-20 使用代码完成

3 库与运行时支持

3.1 版本信息

下表对中标麒麟高级服务器操作系统 V6.4 和中标麒麟高级服务器操作系统 V5.0 所支持编程语言的运行时支持包的版本信息进行了对比。

该列表并不详尽。相反，这是关于中标麒麟高级服务器操作系统 V6.4 所开发软件的标准语言运行时以及关键依赖性的调查。

表 3-1 语言和运行时库版本

Package Name	v6.4	v5.0
glibc	2.12	2.5
libstdc++	4.4	4.1
boost	1.41	1.33
java	1.5 (IBM), 1.6 (IBM, OpenJDK, Oracle Java)	1.4, 1.5, and 1.6
python	2.6	2.4
php	5.3	5.1
ruby	1.8	1.8
httpd	2.2	2.2
postgresql	8.4	8.1
mysql	5.1	5.0
nss	3.12	3.12
openssl	1.0.0	0.9.8e
libX11	1.3	1.0
firefox	3.6	3.6
Kdebase	4.3	3.5
gtk2	2.18	2.10

3.2 兼容性

兼容性是指二进制对象和源代码在不同计算机操作环境中的可移植性。在中标麒麟高级服务器操作系统 V5.0 上组建的应用程序可继续在中标麒麟高级服务器操作系统 V6.4 上运行。

兼容性有两种：

1) 源代码兼容性

源代码兼容性是指，在不同操作环境的实例上，代码都将以一致且可预见的

方式进行编译和运行。此类兼容性由特定应用程序编程接口(API)进行定义。

2) 二进制兼容性

二进制兼容性是指编译后的二进制可行性文件，且 Dynamic Shared Objects(DSOs)在不同操作环境实例上正常运行。此类兼容性由特定应用二进制接口(ABIs)进行定义。

3.2.1 API 兼容性

源代码兼容性可启用大量应用程序源代码进行编译来验证，并跨一个或更多硬件架构，在操作环境的多个实例上正常运行，只要对每个特定硬件架构单独编译源代码。

源代码兼容性是由应用程序编程接口(API)定义的，API 是一套提供给应用开发人员的编程接口和数据结构。C 语言中的 API 编程语法在头文件中进行定义。头文件指定数据类型和编程功能。程序员可在应用程序中使用它们，并由操作系统或库来实现。API 语法在编译时或者当源代码被编译成可执行二进制目标代码时必须满足。

API 可分为：

- 1) De facto standards 未正式规定但由特定实现暗示。
- 2) De jure standards 在标准文档中正式指定。

在所有情况下，应用开发人员应保证他们所依靠的任何行为都在正式 API 文档中有所描述，尽量避免引入对不明确实现特定语义的依赖性，或引入一种 API 的特定实现中的错误依赖。例如，如果依赖于旧版本的行为违反了新版本中的规范，GNU C 库的新发布版本将并不保证与旧版本兼容。

3.2.2 ABI 兼容性

二进制兼容性可启用单个编译二进制文件在操作环境的多个实例上正确运行，该环境共享硬件架构（无论此架构支持是在硬件还是虚拟层实现），而不是不同的底层软件架构。

二进制兼容性由应用二进制接口(ABI)进行定义。ABI 是所有工具所遵守的一系列运行时约定，这些工具处理一个程序编译后的二进制表示。该工具示例包括编译器、连接器、运行时库和操作系统本身。ABI 不仅包括二进制文件格式，也包括库函数的语法和应用程序使用的语法。

3.2.3 策略

理想情况下，为每个主要发布版本重新组建和重新打包应用程序。这使得编译器包含新的优化，也使得最新的工具带来新的特性。

3.2.4 静态链接

所有的中标麒麟高级服务器操作系统发行版本都非常不鼓励静态链接。静态链接引起的问题比它能解决的问题更多，所以无论如何都应避免静态链接。

静态链接的主要弊端在于它只能在组建静态链接的系统上工作，甚至只能持续到 `glibc` 或 `libstdc++`(在 C++情况下)的下次发布的时候。静态组建设没有向前兼容或向后兼容的问题。而且，在库后续更新中的任何安全修复都无效，除非重新连接受影响的静态链接可执行文件。

应避免静态链接的更多原因有：

- 1) 内存占用更大。
- 2) 应用程序启动时间更慢。
- 3) 由于静态链接而会降低的 `glibc` 特性。
- 4) 无法使用如加载地址随机化等安全手段。
- 5) 不支持 `Glibc` 外共享对象的动态加载。
- 6) 有关避免静态链接的其他原因，请参见：[Static Linking Considered Harmful](#)。

3.3 库与运行时详情

3.3.1 GNU C 库

`glibc` 包含有 GNU C 库。该包定义了 ISO C 标准、POSIX 特殊特征，一些 Unix 衍生工具，以及 GNU 特有的扩展所指定的所有功能。GNU C 库中最重要共享库集合是标准 C 和 `math` 库。

GNU C 库在特定的头文件中定义了它的功能，您可以在源代码中声明头文件。每个头文件都包含了一组相关功能的定义；例如，`stdio.h` 头文件定义了 I/O 特有的功能，而 `math.h` 定义了数学运算上的功能。

3.3.1.1 GNU C 库更新

GNU C 库的中标麒麟高级服务器操作系统 V6.4 在中标麒麟高级服务器操作系统 V5.0 基础上展现了如下进步：

1) 增加的输入法区域设置, 包括:

bo_CN
bo_IN
shs_CA
ber_DZ
ber_MA
en_NG
fil_PH
fur_IT
fy_DE
ha_NG
ig_NG
ik_CA
iu_CA
li_BE
li_NL
nds_DE
nds_NL
pap_AN
sc_IT
tk_TM

2) 新增接口, 即:

preadv
preadv64
pwritev
pwritev64
malloc_info
mkostemp
mkostemp64

3) 新增 Linux 特有的接口, 即:

epoll_pwait
sched_getcpu
accept4

fallocate
fallocate64
inotify_init1
dup3
epoll_create1
pipe2
signalfd
eventfd
eventfd_read
eventfd_write

4) 新增检查功能, 即:

asprintf
dprintf
obstack_printf
vasprintf
vdprintf
obstack_vprintf
fread
fread_unlocked
open*
mq_open

有关 GNU C 库更新的详细列表, 请参阅 `/usr/share/doc/glibc-version/NEWS`。

自版本 2.12 起的所有变化都适用于中标麒麟高级服务器操作系统 V6.4 的 GNU C 库。

3.3.1.2 GNU C 库文档

GNU C 库完全被记录在 GNU C Library 手册中, 若要本地访问该手册, 请安装 `glibc-devel` 并运行 `info libc`。该书的最新版本也可访问: http://www.gnu.org/software/libc/manual/html_mono/libc.html。

3.3.2 GNU C++ 标准库

`libstdc++` 包含有 GNU C 标准库, 此库是一个持续进行的项目, 实现 ISO 14882 标准 C++ 库。

安装 `libstdc++` 包会提供刚好满足连接依赖性的文件 (例如, 只有共享库文

件)。要充分运用所有可用库和 C++ 开发的头文件,您必须也安装 `libstdc++-devel`。
`libstdc++-devel` 包也含有标准模板库(STL)的 GNU 特有的实现。

对于中标麒麟高级服务器操作系统 V5.0 和 V6.4, C++ 语言和运行时实现保持稳定,同样,对于 `libstdc++`, 无需兼容库。

3.3.2.1 GNU C++ 标准库更新

GNU C++ 库的中标麒麟高级服务器操作系统 V6.4 在中标麒麟高级服务器操作系统 V5.0 基础上突出了如下改进:

1) 增加了对 ISO C++ TR1 元素的支持, 即:

- <tr1/array>
- <tr1/complex>
- <tr1/memory>
- <tr1/functional>
- <tr1/random>
- <tr1/regex>
- <tr1/tuple>
- <tr1/type_traits>
- <tr1/unordered_map>
- <tr1/unordered_set>
- <tr1/utility>
- <tr1/cmath>

2) 增加对即将到来的 ISO C++ 标准, C++0x 的元素的支持这些元素包括:

- <array>
- <chrono>
- <condition_variable>
- <forward_list>
- <functional>
- <initializer_list>
- <mutex>
- <random>
- <ratio>
- <regex>
- <system_error>

<thread>

<tuple>

<type_traits>

<unordered_map>

<unordered_set>

3) 增加的对-fvisibility 命令的支持。增加的如下扩展:

__gnu_cxx::typelist

__gnu_cxx::throw_allocator

更多关于 libstdc++ 的更新信息, 请参阅 Runtime Library 章节。

GCC 4.2 Release Series Changes, New Features, and Fixes:

<http://gcc.gnu.org/gcc-4.2/changes.html>

GCC 4.3 Release Series Changes, New Features, and

Fixes:<http://gcc.gnu.org/gcc-4.3/changes.html>

GCC 4.4 Release Series Changes, New Features, and Fixes:

<http://gcc.gnu.org/gcc-4.4/changes.html>

3.3.2.2 GNU C++ 标准库文档

若要使用库中组件的 man 页, 请安装 libstdc++-docs 包。这个包可以为几乎库中所有资源提供 man 页信息; 例如, 要查看关于向量容器的信息, 请使用完全合格的组件名:

man std::vector

将显示如下信息 (缩减的信息):

```
std::vector(3)      std::vector(3)

NAME
std::vector -

A standard container which offers fixed time access to individual
elements in any order.

SYNOPSIS
nherits std::_Vector_base< _Tp, _Alloc >.

Public Types
typedef _Alloc allocator_type
typedef __gnu_cxx::__normal_iterator< const_pointer, vector >
const_iterator
typedef _Tp_alloc_type::const_pointer const_pointer
```

```
typedef _Tp_alloc_type::const_reference const_reference  
typedef std::reverse_iterator< const_iterator >
```

libstdc++-docs 包也提供了 HTML 格式的手册和参考信息,请访问如下目录:

file:///usr/share/doc/libstdc++-docs-*version*/html/spine.html

libstdc++开发的主要网站为 gcc.gnu.org.<http://gcc.gnu.org/libstdc++>

3.3.3 Boost

boost 包含有大量免费的同行审阅的 C++源文件库。这些库适合很多任务,如文件系统和时间/日期提取序列化、单元测试、创建线程和多进程同步、解析、绘图、正则表达式处理以及许多其他任务。

安装 boost 包将提供刚好满足连接依赖的文件(例如,只有共享库文件)。若要充分运用 C++开发的所有可用库和头文件,

您必须也安装 boost-devel。

boost 包实际上是个元包,包含很多库的子包。也可安装这些子包来提供更好的包间依赖性跟踪。元包包括所有如下子包:

boost-date-time

boost-filesystem

boost-graph

boost-iostreams

boost-math

boost-program-options

boost-python

boost-regex

boost-serialization

boost-signals

boost-system

boost-test

boost-thread

boost-wave

元包中不包括支持静态链接的包或依赖底层消息传递界面(MPI)支持的包。

通过两种形式提供 MPI 支持:一种是默认的开放 MPI 实现(MPI 支持在 IBM 系统 Z 机器上,此机器上开放 MPI 不可用),而另一种是可选的 MPICH2 实现。所使用的底层 MPI 库的选择由用户决定,并取决于特定的硬件细节和用户喜好。

请注意，可并行安装这些包，因为安装文件具有唯一目录位置。

关于开放 MPI:

boost-openmpi

boost-openmpi-devel

boost-graph-openmpi

boost-openmpi-python

关于 MPICH2:

boost-mpich2

boost-mpich2-devel

boost-graph-mpich2

boost-mpich2-python

若无法避免静态链接，boost-static 包将安装必要的静态库。通用线程和单线程库均会提供。

3.3.3.1 Boost 更新

Boost 的中标麒麟高级服务器操作系统 V6.4 具有很多打包改进技术和新特性。

boost 包的几个方面发生了变化。如上所述，更加独立的较小子包已扩充了庞大的 boost 包。这样做允许用户更多的依赖控制，以及在打包使用 Boost 自定义应用程序时允许较少的二进制包。

另外，所有库的单线程和多线程版本均已打包。根据通用的 Boost 约定，多线程版本包含 mt 后缀。

Boost 的特点还有如下新库:

Foreach

Statechart

TR1

Typeof

Xpressive

Asio

Bimap

Circular Buffer

Function Types

Fusion

GIL
Interprocess
Intrusive
Math/Special Functions
Math/Statistical Distributions
MPI
System
Accumulators
Exception
Units
Unordered
Proto
Flyweight
Scope Exit
Swap
Signals2
Property Tree

许多现有的库已进行改进、修正错误等改进。

3.3.3.2 Boost 文档

boost-doc 包提供了 HTML 格式的手册和参考信息，请访问下面的目录：

`file:///usr/share/doc/boost-doc-version/index.html`

Boost 开发的主要网站位于 boost.org。

3.3.4 Qt

Qt 包提供了 GUI 程序开发中所使用的 Qt（发音为 cute）跨平台应用开发框架。除了作为流行的小工具包外，Qt 还用于开发非 GUI 程序，如控制台工具和服务器。Qt 已用于开发了一些著名项目，如 Google Earth、KDE、Opera、OPIE、VoxOx、Skype、VLC mediaplayer 和 VirtualBox。它是由 Nokia 的 Qt 开发框架部门于 2008 年 6 月 17 日发布的，该产品诞生于 Nokia 收购 Norwegian 公司 Trolltech 后，该公司是 Qt 的原始制作人。

Qt 使用 C++ 标准，但广泛使用称之为 Meta Object Compiler(MOC)的特殊预处理使语言更丰富。Qt 还通过语言捆绑的方式用于其他编程语言。它在所有主要平台上运行，具有广泛的国际化支持。非 GUI Qt 的特性包括 SQL 数据库访

问，XML 解析，线程管理，网络支持，和用于文件处理的统一跨平台 API。

根据 GNU Lesser General Public License（除了别的之外）条款，Qt 是免费开放的源代码软件。Qt 的中标麒麟高级服务器操作系统 V6.4 支持许多种不同的编译器，包括 GCC C++ 编译器和 Visual Studio suite。

3.3.4.1 Qt 更新

Qt 的中标麒麟高级服务器操作系统 V6.4 包含的一些改进如下：

- 1) 高级的用户体验
- 2) 高级的图形效果：支持不透明、投影、模糊化、着色及其他类似效果的选项
- 3) 动画与状态机：创建简单或复杂动画，无需管理复杂代码
- 4) 手势及多点触控支持
- 5) 对新平台的支持
- 6) 现在支持 Windows 7、Mac OSX 10.6、以及其他桌面平台
- 7) 增加对移动开发的支持，Qt 对即将发布的 Maemo 6 平台进行优化，并迅速移植到 Maemo 5。另外，Qt 现在支持 Symbian 平台，与 S60 框架实现整合。
- 8) 增加实时操作系统的支持，如 QNX 和 VxWorks
- 9) 改进的性能，增加对硬件加速渲染（以及其他渲染升级）的支持
- 10) 升级的跨平台 IDE

3.3.4.2 Qt Creator

Qt Creator 是专为满足 Qt 开发人员需要的跨平台 IDE。它包含以下图形工具：

- 1) 先进的 C++ 代码编辑器
- 2) 集成的 GUI 布局和表格设计器
- 3) 项目与组建管理工具
- 4) 完整的上下文相关帮助系统
- 5) 可视的调试器
- 6) 快速代码导航工具

有关 Qt Creator 的详细信息，请参阅如下链接：

<http://qt.nokia.com/products/appdev/developer-tools/developer-tools#qt-tools-at->

a

3.3.4.3 Qt 库文档

qt-doc 包提供 Qt Reference Documentation，对于初学 Qt 框架开发来说，它是不错的起点。

您也可以安装从 qt-demos 和 qt-examples 安装示例。若要总体了解 Qt 框架的功能，请参阅/usr/bin/qtdemo-qt4(demos 提供)。

更多关于 Qt 开发的信息，请参阅下面的在线资源：

Qt 开发人员博客：<http://labs.trolltech.com/blogs/>

Qt 开发人员论坛：<http://qt.nokia.com/developer/developer-zone>

Qt 电子邮件列表：<http://lists.trolltech.com/>

3.3.5 KDE 开发框架

kdelibs-devel 包提供了 KDE 库，该库建立于 Qt 之上，提供使应用程序开发更容易的框架。KDE 开发框架也有助于保持 KDE 桌面环境的一致性。

3.3.5.1 KDE4 架构

中标麒麟高级服务器操作系统 V6.4 中的 KDE 开发框架的架构使用了 KDE4，KDE4 基于如下技术：

1) Plasma

Plasma 取代了 KDE4 中的 KDesktop。它的实现基于 Qt Graphics View Framework，后者将在 Qt4.2 中进行介绍。更多关于 Plasma 的信息，请参阅 <http://techbase.kde.org/Development/Architecture/KDE4/Plasma>

2) Sonnet

Sonnet 是多语言拼写检查应用程序，支持自动的语言检测、主要/备份字典和其他有用的特性。它取代了 KDE4 中的 kspell2

3) KIO

KIO 库为网络透明文件处理提供了框架，允许用户通过网络透明协议很方便地访问文件。它也有助于提供标准文件对话框。

4) KJS/KHTML

KJS 和 KHTML 是成熟的 JavaScript 和 HTML 引擎，基于 KDE4(如 konqueror) 开发出的不同应用程序使用这两个引擎。

5) Solid

Solid 是一个硬件与网络认知框架，该框架允许您开发与硬件交互的应用程

序。它的全面 API 提供支持跨平台应用程序开发的必要摘要。更多信息，请参阅 <http://techbase.kde.org/Development/Architecture/KDE4/Solid>

6) Phonon

Phonon 是一个多媒体框架，它帮助您开发具有多媒体功能的应用程序。它使 KDE 内多媒体功能的使用更为容易。更多信息，请参阅 <http://techbase.kde.org/Development/Architecture/KDE4/Phonon>。

7) Telepathy

Telepathy 提供 KDE4 内的实时通信与协作框架。它的主函数使 KDE 内不同组件间的集成更为紧密。关于项目的概述，请参阅 http://community.kde.org/Real-Time_Communication_and_Collaboration。

8) Akonadi

Akonadi 提供并行架构管理(PIM)组件的中央存储框架。更多信息，请参阅 <http://techbase.kde.org/Development/Architecture/KDE4/Akonadi>。

9) KXMLGUI

KXMLGUI 是使用 XML 设计用户界面的框架。此框架允许您基于动作（由开发人员定义）设计 UI 元素，而无需修改源代码。更多信息，请参阅 <http://developer.kde.org/documentation/library/kdeqt/kde3arch/xmlgui.html>。

10) Strigi

Strigi 是桌面搜索后台程序，它与许多桌面环境和操作系统相兼容。它使用自己的 jstream 系统，该系统允许文件的深层索引。更多关于 Strigi 开发的信息，请参阅 <http://www.vandenoever.info/software/strigi/>。

11) KNewStuff2

KNewStuff2 是许多 KDE4 应用程序所使用的协作式数据共享库。更多信息，请参阅 <http://techbase.kde.org/Projects/KNS2>。

KDE4 中的在线帮助

KDE4 的特点还有便于使用的基于 Qt 的框架，该框架向应用程序增加在线帮助功能。这种功能包括提示、悬浮帮助信息和 khelppcenter 手册。关于 KDE4 内在线帮助的概述，请参阅 http://techbase.kde.org/Development/Architecture/KDE4/Providing_Online_Help。

3.3.5.2 kdelibs 文档

kdelibs-apidocs 包为/usr/share/doc/HTML/en/kdelibs4-apidocs/中的 KDE 开发框架提供 HTML 文档。下面的链接也提供与 KDE 有关的编程任务详细信息：

<http://techbase.kde.org/>

<http://techbase.kde.org/Development/Tutorials>

<http://techbase.kde.org/Development/FAQs>

<http://api.kde.org>

3.3.6 NSS 共享数据库

在 NSS3.12 中介绍的 NSS 共享数据库格式现在可用于中标麒麟高级服务器操作系统 V6.4 中。该数据库包含了大量提高可访问性和可用性的新功能和组件。包含在内的有 NSS 证书和密钥数据库，二者目前是轻量级数据库并允许并发访问。原有 key3.db 和 cert8.db 也由称之为 key4.db 和 cert9.db 的新 SQL 数据库所取代。这些新数据库将存储 PKCS #11 标记对象，与目前 cert8.db 和 key3.db 中存储的标记对象相同。

对共享数据库的支持启用了系统级的 NSS 数据库。它存在于全局信任的 CA 证书适用于所有应用程序的 /etc/pki/nssdb 中。命令 `rv=NSS_InitReadWrite(sql:/etc/pki/nssdb)` 初始化应用程序的 NSS。若应用程序以 root 权限运行，则系统级数据库具有读写权限。但是，若以普通用户权限运行，则它只具有读权限。

另外，NSS 的 PEM PKCS #11 模块允许应用程序加载 PEM 格式文件中保存的内存证书和密钥（例如，由 openssl 生成的文件）。

3.3.6.1 反向兼容

中标麒麟高级服务器操作系统 V6.4 的 NSS 保留了由 NSS 上游部门所作的二进制兼容性保证。该保证声明 NSS 3.12 于所有旧版本 NSS 3.x 共享库反向兼容。因此，与旧版本 NSS 3.x 共享库连接的程序将无需重新编译和重新连接即可运行，并且限制使用 NSS 公共函数的 NSS API 的所有应用程序仍与 NSS 共享库的未来版本保持兼容。

中标麒麟高级服务器操作系统 V5.0 与中标麒麟高级服务器操作系统 V6.4 相同版本的 NSS 上运行，因此没有 ABI 或 API 变化。但是，仍存在差异，因为中标麒麟高级服务器操作系统 V6.4 的 NSS 内部加密模块来源于 NSS 3.12，而中

标麒麟高级服务器操作系统 V5.0 仍使用来源于 NSS 3.15 的旧版本。这意味着 NSS 3.12 引入的新功能（如共享数据库）在 NSS 的中标麒麟高级服务器操作系统 V6.4 中已可用。

3.3.6.2 NSS 共享数据库文档

Mozilla 的 wiki 页详细说明了系统级数据库基本原理。

3.3.7 Python

python 包增加了对 Python 编程语言的支持。该包提供了启动基本 Python 程序的运行时支持所需的对象和缓冲字节码文件。它还包含 python 解释器和 pydoc 文档工具。python-devel 包包含开发 Python 扩展程序所需的库和头文件。

中标麒麟高级服务器操作系统还装有大量与 python 相关的包。按照惯例，这些包的名称应含有 python 前缀或后缀。这种包或者是库扩展，或者是对现有库的 python 绑定。例如，dbus-python 是对 D-Bus 的 Python 语言绑定。

注意，两种缓冲字节码(*.pyc/*.pyo 文件)和编译扩展模块(*.so 文件)在 Python2.4 和 Python2.6 之间不兼容。同样，您需要重新组建不属于中标麒麟高级服务器操作系统部分的所有扩展模块。

3.3.7.1 Python 更新

Python 在中标麒麟高级服务器操作系统 V6.4 具有多种语言变化的特点。更多关于这些变化的信息，请参阅下列项目资源：

Python2.5 中的新内容：<http://docs.python.org/whatsnew/2.5.html>

Python2.6 中的新内容：<http://docs.python.org/whatsnew/2.6.html>

这两个资源还包含有关使用以前 Python 版本开发的移植代码建议。

3.3.7.2 Python 文档

更多关于 Python 的信息，请参阅 man python。还可安装提供 HTML 格式的手册和参考文献的 python-docs，请参阅下列位置：

`file:///usr/share/doc/python-docs-version/html/index.html`

更多关于库和语言组件的详细信息，请使用 `pydoc component_name`。例如，`pydoc math` 将显示如下关于 math Python 模块的信息。

Help on module math:

NAME

math

FILE

/usr/lib64/python2.6/lib-dynload/mathmodule.so

DESCRIPTION

This module is always available. It provides access to the mathematical functions defined by the C standard.

FUNCTIONS

acos[...]

acos(x)

Return the arc cosine (measured in radians) of x.

acosh[...]

acosh(x)

Return the hyperbolic arc cosine (measured in radians) of x.

asin(...)

asin(x)

Return the arc sine (measured in radians) of x.

asinh[...]

asinh(x)

Return the hyperbolic arc sine (measured in radians) of x.

Python 开发项目的主要网站位于 python.org。

3.3.8 Java

java-1.6.0-openjdk 包增加了对 Java 编程语言的支持。该包提供了 java 解释器。java-1.6.0-openjdk-devel 包含有 javac 编译器，

和开发 Java 扩展所需的库及头文件。中标麒麟高级服务器操作系统还装有大量与 java 相关的包。按照惯例，这些包的名称具有 java 前缀或后缀。

3.3.8.1 Java 文档

更多关于 Java 的信息，请参阅 `man java`。一些辅助工具也有它们自己单独的 man 页。

您也可以安装其他 Java 文档包来获得关于特定 Java 工具的更多信息。按照惯例，这些文档包的名字以 javadoc 为后缀（例如，`dbus-java-javadoc`）。

Java 开发的主要网站位于 <http://openjdk.java.net/>。Java 运行时库的主要网站位

于 <http://icedtea.classpath.org/>。

3.3.9 Ruby

ruby 包提供了 Ruby 解释器，并增加了对 Ruby 编程语言的支持。ruby-devel 包含有开发 Ruby 扩展所需的库及头文件。

中标麒麟高级服务器操作系统 V6.4 还装有大量与 ruby 相关的包。按照惯例，这些包的名称含有 ruby 或 rubygem 前缀或后缀。这种包即既可是库扩展，也可对现有库的 Ruby 绑定。

与 ruby 相关的包的示例包括：

ruby-flexmock

rubygem-flexmock

rubygems

ruby-irb

ruby-libguestfs

ruby-libs

ruby-qpidd

ruby-rdoc

ruby-ri

ruby-saslwrapper

ruby-static

ruby-tcltk

更多关于中标麒麟高级服务器操作系统 V6.4 中 Ruby 语言更新的信息，请参阅下列资源：

`file:///usr/share/doc/ruby-version/NEWS`

`file:///usr/share/doc/ruby-version/NEWS-version`

3.3.9.1 Ruby 文档

更多关于 Ruby 的信息，请参阅 `man ruby`。您也可以安装 `ruby-docs`，它提供 HTML 格式的手册和参考文献，请访问如下位置：

`file:///usr/share/doc/ruby-docs-version/`

Ruby 开发的主要站点位于 <http://www.ruby-lang.org>、<http://www.ruby-doc.org/doc.org> 站点也有 Ruby 文档。

3.3.10 Perl

perl 包增加了对 Perl 编程语言的支持。该包提供 Perl 核心模块、Perl 语言解

释器和 PerlDoc 工具。

中标麒麟高级服务器操作系统还以包的形式提供各种不同的 perl 模块, 这些包都以 perl-*前缀命名。这些模块提供独立的应用程序、语言扩展、Perl 库和外部库绑定。

3.3.10.1 Perl 更新

中标麒麟高级服务器操作系统 V6.4 装载了 perl-5.10.1。若要运行较老的系统, 可重新组建或更改相应的扩展模块和应用程序, 以确保最佳性能。

关于 Perl 版本间差异的完整列表, 请参阅下列文档:

Perl 5.10 delta: <http://perldoc.perl.org/perl5100delta.html>

Perl 5.10.1 delta: <http://perldoc.perl.org/perl5101delta.html>

3.3.10.2 安装

通过安装附加模块可扩展 Perl 的功能。这些模块具有如下形式:

1) RPM

使用 yum 或 rpm, 可以安装官方模块包。它们被安装到/usr/share/perl5, 或者对 32 位架构安装到/usr/lib/perl5, 对 64 位架构安装到/usr/lib64/perl5。

2) CPAN 模块

请使用 perl-CPAN 包提供的 cpan 工具, 直接从 CPAN 网站安装模块。它们被安装到/usr/local/share/perl5, 或者对 32 位架构安装到/usr/local/lib/perl5, 对 64 位架构安装到/usr/local/lib64/perl5。

3) 第三方模块包

第三方模块被安装到/usr/share/perl5/vendor_perl, 或者对 32 位架构安装到 perl5/vendor_perl, 对 64 位架构安装到/usr/lib64/perl5/vendor_perl。

4) 定制的模块包/手工安装的模块

这些包应被放置在与第三方模块相同的目录中。也就是 perl5/vendor_perl, 或者对 32 位架构来说是/usr/lib/perl5/vendor_perl, 或者对 64 位架构来说是/usr/lib64/perl5/vendor_perl。



警告: 若已安装正式版本的模块, 安装非正式版本则会引起/usr/share/man 目录冲突。

3.3.10.3 Perl 文档

perldoc 工具提供了关于语言和核心模块的文档。若要详细了解模块, 请使用 perldoc module_name。例如, perldoc CGI 将显示关于 CGI 核心模块的如下信

息:

```

NAME
CGI - Handle Common Gateway Interface requests and responses

SYNOPSIS
use CGI;

my $q = CGI->new;

[...]

DESCRIPTION
CGI.pm is a stable, complete and mature solution for processing and preparing HTTP
requests
and responses.      Major features including processing form submissions, file uploads,
reading
and writing cookies, query string generation and manipulation, and processing and preparing
HTTP headers. Some HTML generation utilities are included as well.

[...]

PROGRAMMING STYLE
There are two styles of programming with CGI.pm, an object-oriented style and a function-
oriented style.      In the object-oriented style you create one or more CGI objects and then
use
object methods to create the various elements of the page.      Each CGI object starts out
with
the list of named parameters that were passed to your CGI script by the server.

[...]
    
```

更多关于 Perl 函数的详细信息，请使用 `perldoc -f function_name`。例如，`perldoc -f split` 将显示如下关于 `split` 函数的信息：

```

split /PATTERN/,EXPR,LIMIT
split /PATTERN/,EXPR
split /PATTERN/
split      Splits the string EXPR into a list of strings and returns that list.      By default,
empty leading fields are preserved, and empty trailing ones are deleted. (If all fields are
empty, they are considered to be trailing.)

In scalar context, returns the number of fields found. In scalar and void context it splits
into the @_ array.      Use of split in scalar and void context is deprecated, however, because
it clobbers your subroutine arguments.
    
```

If EXPR is omitted, splits the \$_ string. If PATTERN is also omitted, splits on whitespace (after skipping any leading whitespace). Anything matching PATTERN is taken to be a delimiter separating the fields. (Note that the delimiter may be longer than one character.)

[...]

当前 Perldoc 文档位于 perldoc.perl.org。

核心和外部模块文档位于 Comprehensive Perl Archive Network 。

4 编译与组建

中标麒麟高级服务器操作系统 V6.4 包括许多用于软件开发的包，其中包括用于编译和组建源代码的工具。本章讨论几个开发包和用于编译源代码的工具。

4.1 GNU 编译器集合(GCC)

GNU 编译器集合(GCC)是一套将各种编程语言（包括 C、C++、ObjectiveC、ObjectiveC++、Fortran 和 Ada）编译为高度优化的机器代码的编译工具。这些工具包括各种编译器（如 gcc 和 g++）、运行时库（如 libgcc、libstdc++、libgfortran 和 libgomp），以及其他实用工具。

4.1.1 GCC 状态与特性

中标麒麟高级服务器操作系统的 GCC 是基于 4.4.x 发布系列的，包括几个漏洞修复、增强功能，以及在即将到来的新版本(包括 GCC4.5)中对更新移植到旧版本的支持。然而，当中标麒麟高级服务器操作系统 V6.4 特性确定时，GCC4.5 并没充分考虑到企业级发行。

该标准化意味着，当对 4.4 系列的更新可用时(4.4.1、4.4.2 等等)，它们将作为更新被包含到中标麒麟高级服务器操作系统 V6.4 的编译器中。中标麒麟高级服务器操作系统也支持把即将发布版本中的更新移植到旧版本中，4.4 系列外的即将发布的版本不违反 Linux 发行物的企业版兼容性的。偶尔，不符合标准的代码也许会编译失败，或它的功能可能会在漏洞修复故障或保持标准的兼容行为的过程中发生变化。

自早期的中标麒麟高级服务器操作系统版本起，GCC 有三个重要的版本：4.2.x、4.3.x、和 4.4.x。下面是它们差异列表的选择概要。

现在，内部无用代码删除规则、编译时间，以及内存使用代码现在都已进行改进。此版本还具有新寄存器分配器、指令调度程序和软件管道等特点。

现在已提供 OpenMP 规范的 3.0 版本对 C、C++和 Fortran 编译器的支持。包含对即将到来的 ISO C++标准(C++0x)的实验支持。这包含对自动/内联命名空间、字符类型和作用域枚举的支持。要启用这种支持，请使用编译器选项-std=c++0x（此选项禁用 GNU 扩展）或-std=gnu++0x。

更多关于 C++0x 改进状况的详细列表，请参阅 http://gcc.gnu.org/gcc-4.4/cxx0x_status.html GCC 现在已包含作业变量跟踪(VTA)架构。这允许 GCC 在优化期

间更好的跟踪变量，以便它能为项目调试器、SystemTap 和其他工具生成改进的调试信息(即 DWARF)。关于 VTA 的概述，请参阅“作业的变量跟踪”

您可以使用 VTA 调试优化代码，比使用早期 GCC 版本调试的效果更好，无需用-O0 编译，这提供了更好的调试体验。

现在支持 Fortran 2008，同时扩展了对 Fortran 2003 的支持。

更多关于 GCC 改进的详细列表，请参阅：

4.2 系列中的更新：<http://gcc.gnu.org/gcc-4.2/changes.html>

4.3 系列中的更新：<http://gcc.gnu.org/gcc-4.3/changes.html>

4.4 系列中的更新：<http://gcc.gnu.org/gcc-4.4/changes.html>

除了通过 GCC4.4 rebase 功能所引入的变化，GCC 的中标麒麟高级服务器操作系统 V6.4，也支持把后来发布版本（即版本 4.5 及其后版本）中的几个漏洞修复和功能增强移植到旧版本中。这些改进包含下面列出的一些：

- 1) 为调试优化++代码而改进的 DWARF3 调试
- 2) Fortran 优化改进
- 3) 对 ix86、Intel 64 和 AMD64，以及 s390 更准确地指令长度信息。
- 4) Intel Atom 支持
- 5) POWER7 支持
- 6) C++原生串支持，u/U/u8 字符串字符支持

4.1.2 语言兼容性

GNU C、C++、Fortran 和 Java 编译器指定的应用二进制接口包括：

- 1) 调用约定：指定参数传递给函数的方法，以及从函数返回结果的方法。
- 2) 寄存器使用约定：指定分配和使用处理器寄存器的方法。
- 3) 目标文件格式：指定二进制目标代码的表示。
- 4) 数据类型的大小、布局和对齐。指定数据在内存中的排列方式。
- 5) 运行时环境提供的接口。该接口中内部存档的语义不因版本不同而发生变化，它们必须保持可用，并始终使用相同名字。

包含在中标麒麟高级服务器操作系统 V6.4 中的默认系统 C 编译器在很大程度上与 C99 ABI 保持兼容。C99 标准与 GCC4.4 的差异可以在[这里](#)查询。³

除 C ABI 外，GNU C++编译器的应用二进制接口还规定了支持 C++语言所需的二进制接口，如：

- 1) 名字改编与识别解码
- 2) 异常情况的创建与传播
- 3) 运行时类型信息格式化
- 4) 构造函数与析构函数
- 5) 类与派生类的布局、对齐和填充
- 6) 虚拟函数实现细节，如虚拟表的布局与对齐


中标麒麟高级服务器操作系统 V6.4 所包含的默认系统 C++ 编译器符合由 Itanium C++ ABI (1.86)⁴ 定义的 C++ ABI。

虽然力争使 GCC 的每个版本都与早期版本保持兼容，但仍存在一些不兼容性。

中标麒麟高级服务器操作系统 V6.4 和中标麒麟高级服务器操作系统 V5.0 之间的 ABI 不兼容性

下面是中标麒麟高级服务器操作系统 V6.4 和 V5.0 工具链之间已知的一些不兼容性列表。

- 1) 在 Intel 64 和 AMD64 上传递/返回包括灵活数组成员的结构体，某些情况下数值有所改变。
- 2) 在 Intel 64 和 AMD64 上传递/返回包括长精度成员的联合体，某些情况下数值有所改变。
- 3) 在 Intel 64 和 AMD64 上传递/返回包括复杂的浮点成员的结构体，某些情况下数值有所改变。
- 4) 当使用 `-mavx` 时，x86、Intel 64 和 AMD64 平台上的 256 位向量传递有所改变。
- 5) `_Decimal{32,64,128}` 类型和聚合体在传递过程中发生了多种变化，该聚合体上包含对几个目标的 `_Decimal{32,64,128}` 传值。
- 6) 一些情况下，对压缩字符位段的打包发生变化。

 **警告：**以上不兼容性使得维持不同版本间的 ABI 共享库难上加难，尤其当核心库之外的多种依赖关系来开发定制库的时候。因此，若开发共享库，强烈推荐为每个中标麒麟高级服务器操作系统 V6.4 组建一个新版本的共享库。

4.1.3 对象兼容性与互操作性

两个重要事项是编译器使用的基础工具的更新与改进，以及语言编译器不同

版本之间的兼容性。

诸如 ld(作为 binutils 包的一部分发行)等工具或动态加载器(ld.so、作为 glibc 包的一部分发行)的更改和新功能可稍微改变编译器生成的目标文件。这些改变意味着,中标麒麟高级服务器操作系统从以前版本移至当前版本的目标文件可能会丢失功能,在运行时表现不同,或降低互操作能力。已知问题包括以下方面:

ld --build-id

默认情况下,在中标麒麟高级服务器操作系统 V6.4 中传递给 ld,而中标麒麟高级服务器操作系统 V5.0 ld 不识别它。

as .cfi_sections 支持

中标麒麟高级服务器操作系统 V6.4 中,该指令允许.debug_frame、.eh_frame 或两者.cfi*指令中发出。中标麒麟高级服务器操作系统 V5.0 中只发出.eh_frame。

As、ld、ld.so 和 gdb STB_GNU_UNIQUE 和%gnu_unique_symbol 支持

中标麒麟高级服务器操作系统 V6.4 中,生成更多的调试信息,并存储在目标文件中。该信息依赖于 DWARF 标准中详述的新特性和尚未标准化的新扩展。中标麒麟高级服务器操作系统 V5.0 中,诸如 as、ld、gdb、objdump 和 readelf 的工具并不是为此新信息而设计,可能无法与使用较新工具创建的对象进行互操作。另外,目标文件生成的中标麒麟高级服务器操作系统 V5.0 不支持这些新特性,这些目标文件可由中标麒麟高级服务器操作系统 V6.4 工具通过次优方式进行处理。

该改进的调试信息的结果是,装有系统库的 debuginfo 包可在系统库(如已安装)中执行有用的源水平调试。更多关于 debuginfo 包的信息,请参阅“5.1 安装 Debuginfo 包”。

如上所列的一些目标文件改变可能会干扰 prelink 的便捷使用。

4.1.4 反向兼容性包

为将源代码或可执行文件从中标麒麟高级服务器操作系统的较早版本移至当前版本,中标麒麟高级服务器操作系统提供了几个包作为实现此操作的辅助工具。这些包通常用于将源文件移至具有不同表现的较新编译器的临时辅助工具,或作为独立系统环境与编译环境之间差异的便捷方式。

下面的包提供了兼容性运行时库,对于在中标麒麟高级服务器操作系统 V5.0 上被编译的 Fortran 可执行文件,它们可以在中标麒麟高级服务器操作系统

V6.4 的当前版本上运行，无需重新编译：

`compat-libgfortran-41`

请注意，并没有为所有支持系统库提供反向兼容性库包，只为与编译器和 C/C++ 标准库有关的系统库提供了反向兼容性库包。

4.1.5 预览中标麒麟高级服务器操作系统 V5.0 中的 V6.4 编译器功能

中标麒麟高级服务器操作系统 V5.0 中，我们包含了 `gcc44` 包作为更新。这是编译器的向后移植，允许运行中标麒麟高级服务器操作系统 V5.0 的用户用中标麒麟高级服务器操作系统 V6.4 编译器编译他们的代码，并在他们将系统升级到下一个主要版本前体验新功能和优化。产生的二进制将与中标麒麟高级服务器操作系统 V6.4 向前兼容，故程序员可以在中标麒麟高级服务器操作系统 V5.0 中用 `gcc44` 编译，而在中标麒麟高级服务器操作系统 V5.0、中标麒麟高级服务器操作系统 V6.4、及其后版本中运行。

中标麒麟高级服务器操作系统 V5.0 `gcc44` 编译器将与装有中标麒麟高级服务器操作系统 V6.4 的 GCC 4.4.x 保持同步，以缓解转变难度。但是，为获得最新功能，推荐使用中标麒麟高级服务器操作系统 V6.4 进行开发。`gcc44` 仅作为转换过程中的辅助工具。

4.1.6 运行 GCC

若使用 GCC 工具进行编译，请先安装 `binutils` 和 `gcc`，这样做还会安装一些依赖关系。

简言之，该工具通过 `gcc` 命令工作。这是编译器的主要驱动程序。从命令行到预处理或编译源文件，连接目标文件和库，或执行这些组合操作均使用该工具。默认情况下，`gcc` 注意 `libgcc` 库中提供的细节和连接。

由 GCC 提供的编译器功能作为 CDT 的一部分也集成至 Eclipse IDE 中。这样做带来很多益处，尤其是对于喜欢图形界面和完全集成环境的开发人员。更多关于在中编译的信息，请参阅“开发工具包”。

相反，从命令行界面使用 GCC 工具消耗更少的系统资源。这也允许用户细粒度的控制编译器，GCC 的命令行工具甚至可以在图形模式（运行级别 5）外使用。

4.1.6.1 简单的 C 用法

使用 GCC 编译 C 语言程序非常简单。请从如下简单的程序开始。

```
//hello.c

#include <stdio.h>
int main ()
{
    printf ("Hello world!\n");
    return 0;
}
```

下面的过程用最基本的形式描述了 C 语言编译过程。

编译'Hello World'C 程序

1) *hello.c* 编译为一个可执行文件，使用：

```
gcc hello.c -o hello
```

2) 保证产生的 *hello* 二进制文件与 *hello.c* 在同一目录中。

执行 *hello* 文件，即 *hello*。

4.1.6.2 简单 C++用法

使用 GCC 编译 C++ 语言程序与以上所述类似。请从下面的简单程序开始：

```
//hello.cc

#include <iostream>
using namespace std;
int main(void)
{
    cout << "Hello World!" << endl;
    return 0;
}
```

下面的过程用最基本的形式描述了 C++ 语言编译过程。

编译'Hello World' C++程序

1) *hello.cc* 编译为一个可执行文件，使用：

```
g++ hello.cc -o hello
```

请保证产生的二进制文件 *hello* 与 *hello.cc* 在同一目录中。

2) 执行 *hello* 文件，即 *hello*。

4.1.6.3 简单多文件用法

要使用涉及多个文件或目标文件的基本编译，请从如下两个源文件开始：

```
//one.c

#include <stdio.h>
void hello()
```

```
{
    printf("Hello world!\n");
}
```

```
//two.c
extern void hello();
int main()
{
    hello();
    return 0;
}
```

以下程序以最基本形式说明了简单、多文件编译过程。

编译多源文件程序

1) *one.c* 编译为可执行文件，使用：

```
gcc -c one.c -o one.o
```

请确保生成的二进制文件 *one.o* 与 *one.c* 在同一目录中。

2) *two.c* 编译为一个可执行文件，使用：

```
gcc -c two.c -o two.o
```

请确保生成的二进制文件 *two.o* 与 *two.c* 在同一目录中。

3) 两个目标文件 *one.o* 和 *two.o* 编译为一个可执行文件，使用：

```
gcc one.o two.o -o hello
```

请确保生成的二进制文件 *hello* 与 *one.o* 和 *two.o* 在同一目录中。

4) 运行 *hello* 二进制文件，即 *hello*。

4.1.6.4 推荐的优化选项

不同项目需要不同的优化选项。涉及优化问题时没有通用的方法，但是要牢记几条策略。

1) 指令选择与修订

对于指令调度来说，选择合适的架构非常重要。默认情况下，GCC 产生的代码针对最常用的处理器做了优化，但是，若知道您的代码运行的 CPU，则应使用相应的 `-mtune=` 选项和 `-march=` 选项，选项 `-mtune=` 用于优化指令调度，选项 `--march=` 用于优化指令选择。

`-mtune=` 选项用于优化指令调度，通过修订除 ABI 和可用指令集之外的任何事项，使指令调度适合您的架构。此选项不会关闭特定的指令，相反，它将按照某种方式修订您的程序，使得在特定架构上的执行被优化。例如，若主要使用

Intel Core2 CPU，请选择`-mtune=core2`。若选择错误，程序仍会运行，但不是在给定架构上的最优化运行。您应一直被选择程序最可能运行的架构。

`-march=`选项用于优化指令选择。同样，选择正确也是非常重要的，因为错误的选择会使您的程序无法运行。该选项选择产生代码时所使用的指令集。例如，若程序将运行在基于 AMD K8 核的 CPU 上，请选择 `march=k8`。用此选项指定架构将隐含`-mtune=`。

`-mtune=`和`-march=`命令只应用于在给定架构内修订和选择指令，而不是用于为不同架构产生代码（也叫做交叉编译）。例如，这不能用于从 Intel 64 和 AMD64 平台产生 PowerPC 代码。

关于`-march=`和`-mtune=`的可用选项的完整列表，请参阅文档：[GCC 4.4.4 Manual: Hardware Models and Configurations](#)。

2) 通用优化标记

编译器标记`-O2` 是生成快速代码的折中选择。当产生的代码规格不大时，它会产生最优的代码。当不确定哪种优化最合适时，请使用此标记。

当不需考虑代码大小时，`-O3` 更合适。此选项生成稍微大一些的代码，但是运行更快，这是因为生成更频繁的内联函数。这是浮点密集型代码的理想选择。

另一个通用优化标记是`-Os`。此标记也优化代码大小，并在由于较小封装而增加代码大小、从而降低缓存未命中情况时，组建快速代码。

当编译对象时，请使用`-frecord-gcc-switches`。它记录将对象组建到对象本身时所用到的选项。当组建目标后，它决定使用哪组选项来组建它。选项集就被记录在目标中叫做`.GCC.command.line` 的部分里，可以使用如下方法查看。

```
$ gcc -frecord-gcc-switches -O3 -Wall hello.c -o hello
$ readelf --string-dump=.GCC.command.line hello

String dump of section '.GCC.command.line':
[ 0]   hello.c
[ 8]   -mtune=generic
[17]   -O3
[1b]  -Wall
[21]  -frecord-gcc-switches
```

使用具有代表性的数据集测试和试验不同的选项非常重要。通常，为产生最优结果，可使用不同的优化标记编译不同的模块或对象。有关其他优化修订的信息，请参阅 4.1.6.5 使用配置文件反馈修订优化启发法。

4.1.6.5 使用配置文件反馈来调整启发式优化算法

将源代码转换为可执行文件的过程中，必须对码速度或代码大小进行成千上万次的选择。默认情况下，这些选择是由编译器完成的，通过使用合理的启发法，不断调整选择以产生最优的运行时性能。然而，GCC 也有指导编译器对特定生产环境中特定机器来优化可执行文件的方法。该功能称之为配置文件反馈。

配置文件反馈用于调整优化，例如：

- 1) 代码嵌入
- 2) 分支预测
- 3) 指令调度
- 4) 程序间常量传递
- 5) 决定常用或不常用函数

配置文件反馈首先编译程序来生成一个在运行时被分析的程序，然后再用收集到的数据做优化。

使用配置文件反馈

- 1) 应用程序必须被配置为生成配置文件信息，通过以下命令进行编译：

```
fprofile-generate.
```

- 2) 运行应用程序来积累并保存配置文件信息。
- 3) 用-fprofile-use 重新编译应用程序。

第 3 步使用第一步中收集的配置文件信息来调整编译器的启发式方法，同时将代码优化为最终的可执行文件。

用配置文件反馈编译程序

- 1) 编译 source.c，包含 profiling 配置：

```
gcc source.c -fprofile-generate -O2 -o executable
```

- 2) 运行可执行文件来收集配置文件信息：

```
./executable
```

- 3) 使用第一步中收集的配置文件信息重新编译并优化 source.c：


```
gcc source.c -fprofile-use -O2 -o executable
```

如同第二步所示，多数据集运行将把数据累加至配置文件，而非取代它。这样做允许第二步中的可执行文件以附加的代表性的数据运行多次，从而收集更多信息。

可执行文件必须以有代表性的级别运行，包括正在使用的机器的级别，以及

各自对所需输入的足够大的数据集。这保证可以达到最优的结果。

默认情况下，GCC 将把配置文件数据生成到执行第一步的目录里。若要在其他位置生成该信息，请用 `-fprofile-dir=DIR` 编译，其中 `DIR` 是选择的输出目录。

 **警告：**编译器反馈数据文件的格式因不同编译器版本而会有所不同，使用编译器的每个新版本重复编译程序是必要的。

4.1.6.6 在 64 位机器上使用 32 位编译器


在 64 位机器上，GCC 将组建只能运行于 64 位机器的可执行程序。然而，GCC 也可组建既可以在 64 位机器上运行，也可以在 32 位机器上运行的可执行程序。

若要在 64 位机器上组建 32 位文件，请先安装可执行文件所需的任何支持库的 32 位版本。这必须至少包括对 `glibc` 和 `libgcc` 的支持库，若程序是一个 C++ 程序，可能还要包括对 `libstdc++` 的支持库。在 Intel 64 和 AMD64 机器上，可以通过下面的命令完成：

```
yum install glibc-devel.i686 libgcc.i686 libstdc++-devel.i686
```

可能也存在这样的情况，需要安装一个程序所需的额外的 32 位库。例如，若一个程序使用 `db4-devel` 库来组建，则这些库的 32 位版本可以用如下命令安装：

```
yum install db4-devel.i686
```

 **说明：**x86 平台（与 x86-64 相对）上的 `i686` 后缀指定了给定包的 32 位版本。对于 PowerPC 架构，后缀是 `ppc`（与 `ppc64` 相对）。

在安装完 32 位库后，可将 `-m32` 选项传递给编译器和连接器来组建 32 位可执行程序。假设支持 32 位的库安装在 64 位系统上，此可执行程序将能既可以在 32 位系统上运行，也可以在 64 位系统上运行。

在 64 位机器上编译 32 位程序：

- 1) 在 64 位系统上，将 `hello.c` 编译为 64 位可执行程序，请使用：

```
gcc hello.c -o hello64
```

- 2) 请确保生成的可执行文件是 64 位二进制：

```
$ file hello64
hello64: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), dynamically linked
(uses shared libs), for GNU/Linux 2.6.18, not stripped
$ ldd hello64
linux-vdso.so.1 → (0x00007fff242dd000)
libc.so.6 → /lib64/libc.so.6 (0x00007f0721514000)
/lib64/ld-linux-x86-64.so.2 (0x00007f0721893000)
```

64 位可执行文件中的 file 命令将在它的输出中包括 ELF 64-bit, ldd 将列出 /lib64/libc.so.6 作为主要链接的 C 库。

3) 在 64 位系统中, 将 hello.c 编译为 32 位可执行程序, 请使用:

```
gcc -m32 hello.c -o hello32
```

4) 请确保生成的可执行文件是 32 位文件:

```
$ file hello32
hello32: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), dynamically
linked (uses shared libs), for GNU/Linux 2.6.18, not stripped
$ ldd hello32
linux-gate.so.1 → (0x007eb000)
libc.so.6 → /lib/libc.so.6 (0x00b13000)
/lib/ld-linux.so.2 (0x00cd7000)
```

64 位可执行文件中的 file 命令将在它的输出中包括 ELF 64-bit, ldd 将列出 /lib64/libc.so.6 作为主要链接的 C 库。

若您还没有安装 32 位支持库, 您将收到一个错误, 类似于针对 C 代码的错误:

```
$ gcc -m32 hello32.c -o hello32
/usr/bin/ld: crt1.o: No such file: No such file or directory
collect2: ld returned 1 exit status
```

C++代码上将引发类似的错误:

```
$ g++ -m32 hello32.cc -o hello32-c++
In file included from /usr/include/features.h:385,
from /usr/lib/gcc/x86_64-neokylin-linux/4.4.4/../../../../include/c++/4.4.4/x86_64-neokylin-
linux/32/bits/os_defines.h:39,
from /usr/lib/gcc/x86_64-neokylin-linux/4.4.4/../../../../include/c++/4.4.4/x86_64-neokylin-
linux/32/bits/c++config.h:243,
from /usr/lib/gcc/x86_64-neokylin-linux/4.4.4/../../../../include/c++/4.4.4/iostream:39,
from hello32.cc:1:
/usr/include/gnu/stubs.h:7:27: error: gnu/stubs-32.h: No such file or directory
```

这些错误表明没有按照本节开始所讲述的那样正确安装支持 32 位的库。

同样重要的是, 注意用 -m32 组建将不适用或转换一个程序达到解决由 32/64 不兼容性引起的问题的效果。关于编写可移植代码的技巧和从 32 位到 64 位的转换, 请参阅论文 [Porting to 64-bit GNU/Linux Systems in the Proceedings of the 2003 GCC Developers Summit](#)。

4.1.7 GCC 文档

更多关于 GCC 编译器的信息, 请参阅 `cpp`、`gcc`、`g++`、`gcj` 和 `gfortran` 的

man 页。

也可以参阅下面的在线用户手册：

GCC 4.4.4 Manual

GCC 4.4.4 GNU Fortran Manual

GCC 4.4.4 GCJ Manual

GCC 4.4.4 CPP Manual

GCC 4.4.4 GNAT Reference Manual

GCC 4.4.4 GNAT User's Guide

GCC 4.4.4 GNU OpenMP Manual

GCC 开发的主要网站位于 gcc.gnu.org。

4.2 分布编译

中标麒麟高级服务器操作系统也支持分布式编译。这涉及到将一个编译作业转换为很多较小的作业，这些作业分布在机器集群上，这将加快组建时间（尤其是对于使用大型代码库的程序）。`distcc` 包提供该功能。

若要设置分布式编译，请先安装如下包：

`distcc`

`distcc-server`

更多关于分布式编译的信息，请参阅 `distcc` 和 `distccd` 的 man 页。

下面的链接也提供了关于 `distcc` 开发的详细信息：

<http://code.google.com/p/distcc>

4.3 Autotools

GNU Autotools 是一个命令行工具包，允许开发人员在不同系统上组建应用程序，而无需考虑所安装的包，或甚至是 Linux 版本。这些工具辅助开发人员创建 `configure` 脚本。此脚本在组建前运行，并创建组建应用程序所需的顶层 `Makefile`。`configure` 脚本根据组建器提供的参数在当前系统中执行测试，创建附加文件，或运行其他指令。

Autotools 工具包中最常用工具有：

autoconf

从一个输入文件（例如，`configure.ac`）生成 `configure` 脚本。

automake

在特定系统上为一个项目创建 `Makefile`。

autoscan

生成初步的输入文件（即 `configure.scan`），可编辑此文件来创建被 `autoconf` 使用的最终 `configure.ac`。

Autotools 工具包中所有工具都是 Development Tools 包组的一部分。您可以通过安装此包组来安装完整的 Autotools 工具包，或根据您的需要，只是使用 `yum` 来安装工具包中的任何工具。

4.3.1 Eclipse 的 Autotools 插件

通过 Autotools 插件，Autotools 工具包也集成至 Eclipse IDE 中。此插件提供 Autotools 的 Eclipse 图形用户界面，适合多数 C/C++ 项目使用。

自中标麒麟高级服务器操作系统 V6.4 起，该插件仅支持 C/C++ 新项目的两个模板：

空项目

"hello world"应用程序

将项目导入已支持 Autotools 的 C/C++ 开发工具包时使用空项目模板。将来对 Autotools 插件的更新将包括用于创建共享库和其他复杂情境的额外的图形用户界面（例如，向导）。

Autotools 插件的中标麒麟高级服务器操作系统 V6.4 也没有将 `git` 或 `mercurial` 集成至 Eclipse 中。同样，使用 `git` 软件库的 Autotools 项目需在工作区之外被检查。然后，您可以为 Eclipse 中此类项目指定源文件位置。任何软件库操作（例如，委托、更新）均需通过命令行来完成。

4.3.2 配置脚本

Autotools 最关键的功能就是创建 `configure` 脚本。此脚本分析系统信息、输入文件还有其他有用的信息以组建项目。¹⁵`configure` 脚本组建 `Makefile`，这允许 `make` 工具来组建基于系统配置的项目。

若要创建 `configure` 脚本，请先创建输入文件。然后将它提供给 Autotools 实用工具，以便创建 `configure` 脚本。此输入文件有代表性的是 `configure.ac` 或 `Makefile.am`，前者常由 `autoconf` 处理，后者提供给 `automake`。

若 `Makefile.am` 输入文件可用，则 `automake` 实用工具创建 `Makefile` 模板（即，`Makefile.in`），它引用配置时期所收集的信息。例如，

`Makefile` 需要连接到一个特定库，当且仅当已经安装了那个特定库。当

configure 脚本运行时，automake 将使用 Makefile.in 模板来创建一个 Makefile。

反而，若一个 configure.ac 文件是可用的，则 autoconf 将基于被 configure.ac 激活的宏来自动创建 configure 脚本。若要创建一个基本 configure.ac，请使用 autoscan 实用工具并相应地编辑文件。

4.3.3 Autotools 文档

中标麒麟高级服务器操作系统 V6.4 包括 autoconf、automake、autoscan 和 Autotools 工具包中多数工具的 man 页。另外，Autotools 社区提供了大量关于 autoconf 和 automake 的文档，请访问以下网站：

<http://www.gnu.org/software/autoconf/manual/autoconf.html>

<http://www.gnu.org/software/autoconf/manual/automake.html>

下面是描述 Autotools 使用的一本在线书籍。虽然推荐阅读上面的在线文档，而且它们都保持了关于 Autotools 的最新信息，但是下面这本书也是一个很好的选择和入门读物。

<http://sourceware.org/autobook/>

关于如何创建 Autotools 输入文件的信息，请参阅：

<http://www.gnu.org/software/autoconf/manual/autoconf.html#Making-configure-Scripts>

<http://www.gnu.org/software/autoconf/manual/automake.html#Invoking-Automake>

下面的示例也描述了 Autotools 在简单 hello 程序中的使用：

<http://www.gnu.org/software/hello/manual/hello.html>

Eclipse Autotools 插件白皮书也提供了关于 Autotools 插件的中标麒麟高级服务器操作系统 V6.4 的更多细节。该白皮书也包含一个示例分析，使您了解关于插件的典型使用情况。

4.4 Eclipse 内置 Specfile 编辑器

Eclipse Specfile 编辑器插件提供了帮助开发人员管理.spec 文件的有用功能。该插件允许用户在编辑.spec 文件时平衡几种 Eclipse GUI 功能，例如自动完成、高亮显示、文件超链接和折叠。

另外，Specfile 编辑器插件也将 rpmlint 工具整合到 Eclipse 界面中。rpmlint 是命令行工具，帮助开发人员检测常见 RPM 包错误。Eclipse 界面提供丰富的可

视化有助于开发人员迅速检测、查看和改正由 rpmlint 报告的错误。

Eclipse Specfile 编辑器由 eclipse-rpm-editor 包提供。更多关于此插件的信息，请参阅 Eclipse Help Contents 中的 Specfile 编辑器用户指南。

关于 configure 能执行的测试的信息，请参阅下面的链接

<http://www.gnu.org/software/autoconf/manual/autoconf.html#Existing-Tests>

5 调试

写得好的有用软件通常都经历应用开发的几个不同阶段，因此带来大量犯错的机会。一些阶段具有一套自我检测错误的机制。例如：编译期间，经常执行基本语法分析确保诸如变量和函数等对象描述正确。

在应用开发中的每个阶段，所执行的错误检测机制力争发现代码中简单和明显的错误。调试阶段有助于发现更多细微的、在常规代码检查中不能发现的错误。

5.1 安装 Debuginfo 包

中标麒麟高级服务器操作系统 V6.4 还为操作系统中所有独立架构的 RPMS 提供 `-debuginfo` 包。`-debuginfo` 包含有相应包的准确调试信息。若要安装包（即典型的 `packagename-debuginfo`）的 `-debuginfo` 包，

请使用如下命令：

```
debuginfo-install packagename
```



注意：试图调试未安装 `-debuginfo` 的等价包的软件包可能会导致失败，尽管 GDB 尝试尽可能提供所有有用的诊断。

5.2 GDB

实际上，正如多数调试器一样，GDB 在非常紧密的受控环境中管理编译代码的执行。该环境使 GDB 操作所需的以下基本机制成为可能：

- 1) 检查并修改所调试代码的内存（例如，读取和设置变量）。
- 2) 控制所调试代码的执行状态，主要指是否运行或已停止。
- 3) 检测特定代码段的执行（例如，当代码运行到程序员感兴趣的指定区域时，停止正在运行的代码）。
- 4) 检测对内存特定区域的访问（例如，当代码访问特定变量时，停止正在运行的代码）。
- 5) 以受控方式执行（从另一个停止的程序中）部分代码。
- 6) 发现各种程序异步事件，如信号。

这些机制的运行主要取决于编译器生成的信息。例如，要查看变量值，GDB 必须知道：

- 1) 变量在内存中的位置
- 2) 变量的特征

这意味着显示双精度浮点数需要与显示字符串完全不同的过程。对于复杂结构体，GDB 不仅必须知道结构体中每个元素，还必须了解结构体的构成。

为完善功能，GDB 需要以下项目：

1) 调试信息

多数 GDB 操作依赖于程序的调试信息。虽然该信息通常来自于编译器，但该信息只对调试程序很有必要，也就是说，在程序正常运行中并不使用该信息。为此，编译器并不总是保证该信息在默认情况下可用，例如，必须明确指示 GCC 提供带有 -g 标志的调试信息。

为充分使用 GDB 功能，首先确保该调试信息可用于 GDB 是明智之举。当 GDB 运行时，若没有可用的调试信息，GDB 只能发挥有限的作用。

2) 源代码

GDB（或其他调试器）最有用的一个功能是将程序执行中事件及环境关联到它们在源代码中对应位置的能力。该位置通常指源文件的特定行或行集。当然，这要求程序源代码在调试时可用于 GDB。

5.2.1 简单 GDB

GDB 实际上包含许多命令。本节描述最基本的命令。

br(breakpoint)

一旦到达执行中的指定点，断点命令就指示 GDB 停止执行。该断点可用多种方式指定，但最常用的只是源文件中的行号，或函数名。许多断点可以同时生效。这经常是启动 GDB 后发出的第一个命令。

r(run)

run 命令启动程序的执行。若 **run** 执行时有参数，则那些参数被传递给可执行文件，就好像程序已正常启动一样。用户在设置断点后通常发出该命令。

在启动可执行文件前，或当可执行文件停止（如在断点处）时，可以检查程序很多方面的状态。以下命令是几个较常用的检查方法。

p(print)

print 命令显示给定参数的值，该参数可以是与程序有关的任何事情。通常，参数只是从简单单个值变量到结构变量的任意复杂变量的名称。参数还可是当前语言中的合法表达式，包括使用程序变量和库函数，或被测试的程序中所定义的函数。

bt(backtrace)

backtrace 显示了终止执行前所使用的函数调用链。这对于调查由隐蔽原因引起的严重错误（如分段错误）非常有用。

l(list)

当停止执行时，**list** 命令显示了与程序停止位置所对应的源代码的行。停止的执行程序可以用多种方式重新开始执行。下面是最常见的方式。

c(continue)

continue 命令只是重新启动程序的执行，继续执行直到它遇到断点、碰到特定或意外条件（例如错误）

或终止。

n(next)

与 **continue** 一样，**next** 命令也重新启动执行，然而，除了 **continue** 命令中隐含的停止条件外，**next** 还会在当前源文件的下一行代码序列行处终止执行。

s(step)

与 **next** 一样，**step** 命令也在当前源文件的每个代码序列行处停止执行。但是，若在包含函数调用的源文件行处停止执行，在进入函数调用（而不是执行它）后 GDB 停止执行。

fini(finish)

与 **aforementioned** 命令一样，**finish** 命令继续执行，但当从函数返回时就停止执行。

最后，两个非常重要的命令：

q(quit)

此命令终止执行。

h(help)

help 命令提供了访问扩展的内部文档的权限。该命令执行参数：例如：**help breakpoint**(或 **h br**)显示有关 **breakpoint** 命令的详细描述。更多详细信息，请参阅每个命令的 **help** 输出。

5.2.2 运行 GDB

本节通过以下简单程序描述 GDB 的基本执行过程：

```
//hello.c
```

```
#include <stdio.h>
char hello[] = { "Hello, World!" };
int a;
main()
{
    fprintf(stdout, "%s\n", hello);
    return (0);
}
```

下面描述了最基本的调试过程。

调试'Hello World'程序

1) 使用调试标识集将 *hello.c* 编译为可执行文件:

```
gcc -g -o hello hello.c
```

请确保生成的二进制文件 *hello* 与 *hello.c* 在同一目录中。

2) 在 *hello* 上运行 *gdb*, 即


```
gdb hello
```

3) 几个介绍性注释后, *gdb* 将显示默认的 GDB 提示符:

```
(gdb)
```

4) 开始执行前可以完成一些事情。可变 *hello* 是全局变量, 因此在 *main* 过程开始前可见。

```
(gdb) p hello
$1 = "Hello, World!"
(gdb) p hello[0]
$2 = 72 'H'
(gdb) p *hello
$3 = 72 'H'
(gdb)
```

 注意: *print* 将 *hello[0]* 和 **hello* 作为目标, 都需要计算表达式, 例如, **(hello + 1)*:

```
(gdb) p *(hello + 1)
$4 = 101 'e'
```

5) 下一步, 列出源代码:

```
(gdb) l
1 #include <stdio.h>
2
3 char hello[] = { "Hello, World!" };
4
5 int
6 main()
7 {
8 fprintf(stdout, "%s\n", hello);
```

```

9 return (0);
10 }
  
```

list 显示了 fprintf 调用在第 8 行。在那里设置断点并重启代码：

```

(gdb) br 8
Breakpoint 1 at 0x80483ed: file hello.c, line 8.
(gdb) r
Starting program: /home/moller/tinkering/gdb-manual/hello
Breakpoint 1, main () at hello.c:8
8 fprintf (stdout, "%s\n", hello);
  
```

6) 最后，使用“next”命令来单步跳过 fprintf 调用，并执行它：

```

(gdb) n
Hello, World!
9 return (0);
  
```

下面章节描述了 GDB 更加复杂的应用。

5.2.3 条件断点

在很多现实情况下，程序可能在最初的几千次很好地执行任务，然后也许在第八千次反复执行任务时，就开始发生故障、出现错误。调试这种程序很困难，因为很难想象程序员有耐心几千次发出 continue 命令就为了达到引发故障的重复条件。

类似情况在现实生活中很常见，这正是 GDB 允许程序员附加对断点的条件的的原因。例如，考虑如下程序：

```

//simple.c

#include <stdio.h>
main()
{
    int i;
    for (i = 0;; i++) {
        fprintf (stdout, "i = %d\n", i);
    }
}
  
```

要在 GDB 提示符处设置条件断点：

```

(gdb) br 8 if i == 8936
Breakpoint 1 at 0x80483f5: file iterations.c, line 8.
(gdb) r
  
```

有了该条件，程序执行将最终停止并给出如下输出：

```

I=8931
i=8932
i=8933
  
```

```
i=8934
i=8935
```

```
Breakpoint 1, main () at iterations.c:8
8  fprintf (stdout, "i = %d\n", i);
```

检查断点信息(使用 info br)查看断点状态:

```
(gdb) info br
Num      Type      Disp Enb Address      What
1  breakpoint keep y  0x080483f5 in main at iterations.c:8
      stop only if i == 8936
      breakpoint already hit 1 time
```

5.2.4 Forked 执行

在很多具有挑战性的错误中，程序员面对的是一个程序(父)进行独立复制(*fork*)的位置。该 *fork* 然后创建子进程，该进程接下来运行失败。调试父进程也许有用，也许没用。通常，获得错误的唯一方法可能就是通过调试子进程，但这并不总是可以做到的。

`set follow-fork-mode` 特性常用于克服这种问题，允许程序员跟踪子进程，而不是父进程。

```
set follow-fork-mode parent
```

在 *fork* 后调试原进程。子进程运行顺畅。这是默认的。

```
set follow-fork-mode child
```

在 *fork* 后调试新进程。子进程运行顺畅。

```
show follow-fork-mode
```

显示当前响应 *fork* 调用的调试器。

请使用 `set detach-on-fork` 命令调试 *fork* 后的父进程和子进程，或对二者保持调试器控制。

```
set detach-on-fork on
```

子进程（或父进程，取决于 `follow-fork-mode` 的值）将被分离并允许它独立运行。这是默认的。

```
set detach-on-fork off
```

两个进程将处于 **GDB** 的控制下。一个进程（子进程或父进程，取决于 `follow-fork-mode` 的值）可如常调试，而其他进程则被暂停。

```
show detach-on-fork
```

显示 `detach-on-fork` 模式是否为 `on` 或 `off`。

考虑如下程序：

```
//fork.c

#include <unistd.h>
int main()
{
    pid_t  pid;
    const char *name;

    pid = fork();
    if (pid == 0)
    {
        name = "I am the child";
    }
    else
    {
        name = "I am the parent";
    }
    return 0;
}
```

此程序以 `gcc -g fork.c -o fork -lpthread` 命令编译，并受到 GDB 监控，它将显示：

```
gdb ./fork
[...]
(gdb) run
[...]
Breakpoint 1, main () at fork.c:8
8  pid = fork();
(gdb) next
Detaching after fork from child process 3840.
9  if (pid == 0)
(gdb) next
15 name = "I am the parent";
(gdb) next
17 return 0;
(gdb) print name
$1 = 0x400717 "I am the parent"
```

GDB 跟从父进程，并允许子进程(进程 3840)继续执行。

下面是使用 `follow-fork-mode child` 所做的相同测试。

```
(gdb) set follow-fork-mode child
(gdb) break main
Breakpoint 1 at 0x4005dc: file fork.c, line 8.
```

```
(gdb) run
[...]
Breakpoint 1, main () at fork.c:8
8  pid = fork();
(gdb) next
[New process 3875]
[Thread 调试 using libthread_db enabled]
[Switching to Thread 0x7fff7fd5720 (LWP 3875)]
9  if (pid == 0)
(gdb) next
11      name = "I am the child";
(gdb) next
17 return 0;
(gdb) print name
$2 = 0x400708 "I am the child"
(gdb)
```

GDB 切换到此处的子进程。通过对.gdbinit 增加合适的设置保持不变。

例如，若向~/.gdbinit 增加设置 set follow-fork-mode ask，则 ask mode 变为默认模式。

5.2.5 调试单个线程

GDB 能够调试单个线程，并分别进行处理和检查。默认情况下，不启用该功能。若要执行，可使用 set non-stop on 和 set target-async on。可将这些设置添加到.gdbinit。打开该功能后，GDB 就准备进行线程调试。

例如，下面程序创建了两个线程。这两个线程与执行 main 的原线程一起共有三个线程。

```
//three-threads.c

#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

pthread_t thread;

void* thread3 (void* d)
{
    int count3 = 0;

    while(count3 < 1000){
        sleep(10);
        printf("Thread 3: %d\n", count3++);
        return NULL;
    }
}
```

```
}

void* thread2 (void* d)
{
    int count2 = 0;

    while(count2 < 1000){
        printf("Thread 2: %d\n", count2++);
        return NULL;
    }

int main (){

    pthread_create (&thread, NULL, thread2, NULL);
    pthread_create (&thread, NULL, thread3, NULL);

    //Thread 1
    int count1 = 0;

    while(count1 < 1000){
        printf("Thread 1: %d\n", count1++);
    }

    pthread_join(thread,NULL);
    return 0;
}
```

编译此程序，以便在 GDB 下对其进行检验。

```
gcc -g three-threads.c -o three-threads -lpthread
gdb ./three-threads
```

首先在所有线程函数上设置断点，thread1、thread2 和 main。

```
(gdb) break thread3
Breakpoint 1 at 0x4006c0: file three-threads.c, line 9.
(gdb) break thread2
Breakpoint 2 at 0x40070c: file three-threads.c, line 20.
(gdb) break main
Breakpoint 3 at 0x40074a: file three-threads.c, line 30.
```

然后运行程序。

```
(gdb) run
[...]
Breakpoint 3, main () at three-threads.c:30
30
pthread_create (&thread, NULL, thread2, NULL);
[...]
(gdb) info threads
```

```
* 1 Thread 0x7ffff7fd5720 (LWP 4620)      main () at three-threads.c:30
(gdb)
```

注意，`info threads` 命令总结了程序的线程和关于他们当前状态的详细信息。本例中，目前只创建了线程。

继续执行更多部分。

```
(gdb) next
[New Thread 0x7ffff7fd3710 (LWP 4687)]
31 pthread_create (&thread, NULL, thread3, NULL);
(gdb)
Breakpoint 2, thread2 (d=0x0) at three-threads.c:20
20 int count2 = 0;
next
[New Thread 0x7ffff75d2710 (LWP 4688)]
34 int count1 = 0;
(gdb)
Breakpoint 1, thread3 (d=0x0) at three-threads.c:9
9 int count3 = 0;
info threads
3 Thread 0x7ffff75d2710 (LWP 4688)      thread3 (d=0x0) at three-threads.c:9
2 Thread 0x7ffff7fd3710 (LWP 4687)thread2 (d=0x0) at three-threads.c:20
* 1 Thread 0x7ffff7fd5720 (LWP 4620)      main () at three-threads.c:34
```

这里，创建了另外两个线程。**Star** 表示当前关注的线程。另外，新创建的线程在初始化函数中遇到为它们设置的断点。也就是 `thread2()` 和 `thread3()`。

要开始真正的线程调试，请使用 `thread<thread number>` 命令将焦点切换到另一线程。

```
(gdb) thread 2
[Switching to thread 2 (Thread 0x7ffff7fd3710 (LWP 4687))]#0      thread2 (d=0x0)
at three-threads.c:20
20 int count2 = 0;
(gdb) list
15 return NULL;
16 }
17
18 void* thread2 (void* d)
19 {
20     int count2 = 0;
21
22     while(count2 < 1000){
23         printf("Thread 2: %d\n", count2++);
24 }
```

Thread 2 在函数 `thread2()` 第 20 行处停止。

```

(gdb) next
22 while(count2 < 1000){
(gdb) print count2
$1 = 0
(gdb) next
23 printf("Thread 2: %d\n", count2++);
(gdb) next
Thread 2: 0
22 while(count2 < 1000){
(gdb) next
23 printf("Thread 2: %d\n", count2++);
(gdb) print count2
$2 = 1
(gdb) info threads
3 Thread 0x7fff75d2710 (LWP 4688)      thread3 (d=0x0) at three-threads.c:9
* 2 Thread 0x7fff7fd3710 (LWP 4687)  thread2 (d=0x0) at three-threads.c:23
1 Thread 0x7fff7fd5720 (LWP 4620)main () at three-threads.c:34
(gdb)
  
```

上面，thread2 的几行代码打印出计数器 count2 的值，并在第 23 行离开 thread2，这可以从'info threads'的输出中看到。

现在来看看 thread3:

```

(gdb) thread 3
[Switching to thread 3 (Thread 0x7fff75d2710 (LWP 4688))]#0      thread3 (d=0x0)
at three-threads.c:9
9  int count3 = 0;
(gdb) list
4
5  pthread_t thread;
6
7  void* thread3 (void* d)
8  {
9  int count3 = 0;
10
11      while(count3 < 1000){
12          sleep(10);
13          printf("Thread 3: %d\n", count3++);
(gdb)
  
```

稍有不同，它有一条休眠语句，且执行缓慢。可将其作为乏味的 IO 线程的典型代表。既然此线程很乏味，请使用 continue 继续不间断的执行此线程。

```

(gdb) continue&
(gdb) Thread 3: 0
Thread 3: 1
Thread 3: 2
  
```

Thread 3: 3

请注意 `continue` 结尾处的 `&` 符号。这使 GDB 命令提示符可以返回，从而可以执行其他命令。使用 `interrupt`，可以停止当前的执行，以重新使 `thread 3` 有意义。

```
(gdb) interrupt
[Thread 0x7fff75d2710 (LWP 4688)] #3 stopped.
0x000000343f4a6a6d in nanosleep () at ../sysdeps/unix/syscall-template.S:82
82 T_PSEUDO (SYSCALL_SYMBOL, SYSCALL_NAME, SYSCALL_NARGS)
```

也有可能回到原始 `main` 线程并做更多检查。

```
(gdb) thread 1
[Switching to thread 1 (Thread 0x7fff7fd5720 (LWP 4620))]#0
at three-threads.c:34
main ()
34 int count1 = 0;
(gdb) next
36 while(count1 < 1000){
(gdb) next
37 printf("Thread 1: %d\n", count1++);
(gdb) next
Thread 1: 0
36 while(count1 < 1000){
(gdb) next
37 printf("Thread 1: %d\n", count1++);
(gdb) next
Thread 1: 1
36 while(count1 < 1000){
(gdb) next
37 printf("Thread 1: %d\n", count1++);
(gdb) next
Thread 1: 2
36 while(count1 < 1000){
(gdb) print count1
$3 = 3
(gdb) info threads
3 Thread 0x7fff75d2710 (LWP 4688)      0x000000343f4a6a6d in nanosleep ()
at ../sysdeps/unix/syscall-template.S:82
2 Thread 0x7fff7fd3710 (LWP 4687)thread2 (d=0x0) at three-threads.c:23
* 1 Thread 0x7fff7fd5720 (LWP 4620)  main () at three-threads.c:36
(gdb)
```

因此，从 `info` 线程输出可看出，其他线程位于 `thread 1` 调试留下的，不受影响的位置。

5.2.6 可选的 GDB 用户界面

GDB 使用命令行作为默认界面。但是，还有称之为人机界面(MI)的 API。MI 允许 IDE 开发人员创建 GDB 的其他用户界面。

这些界面中的一些示例为：

Eclipse(CDT)

与 Eclipse 开发环境集成的图形调试器界面。更多信息请参考 Eclipse 官网。

Nemiver

适合 GNOME 桌面环境的图形调试器界面。更多信息请参考 Nemiver 官网。

Emacs

与 emacs 集成的 GDB 界面。更多信息请参阅 Emacs 官网。

5.2.7 GDB 文档

更多关于 GDB 的详细信息，请参阅 GDB 手册：

info gdb 和 man gdb 命令将提供更多 gdb 已安装版本最新的精确信息。

5.3 作业的变量跟踪

作业变量跟踪(VTA)是 GCC 中包含的新架构，GCC 用于在优化过程中提高变量跟踪。这样可使 GCC 为 GDB、SystemTap 和其他调试工具生成更精确、更有意义、且更有用的调试信息。

启用优化后 GCC 编译代码后，变量可重命名、调整位置、甚至被一起删除。同样，优化编译可引发调试器报告一些已优化的变量。启用 VTA 后，在内部注释优化代码，以确保优化容易跟踪每个变量的值，而不管变量是否移动或删除。

在调试包含内联函数的应用程序时，VTA 的好处就显而易见。没有 VTA 时，为防止调试器检查函数值，优化可完全删除内联函数的一些参数。有了 VTA，优化仍会运行，并且将会为缺失参数生成相应的调试信息。

默认情况下，当启用优化和调试信息来编译代码时，启用 VTA。若要在编译时禁用 VTA，请增加 fno-var-tracking-assignments。另外，VTA 架构包含新的 gcc 选项-fcompare-debug。此选项测试包含调试信息和不包含调试信息的 GCC 编译的代码：若两个二进制文件一致，则测试通过。此测试确保可执行代码不受任何调试选项的影响，进而保证调试代码中没有隐藏的错误。注意，-fcompare-debug 增加了编译时的大量成本。关于此选项的详细信息，请参阅 man gcc。

更多关于 VTA 架构和开发的信息，请参阅 [A Plan to Fix LocalVariable Debug Information in GCC](http://gcc.gnu.org/wiki/Var_Tracking_Assignments)，请访问下面链接：http://gcc.gnu.org/wiki/Var_Tracking_Assignments

5.4 Python Pretty-Printers

GDB 的 `print` 命令输出目标应用程序的综合调试信息。GDB 力争为用户提供尽可能多的调试信息。但是，这意味着对于非常复杂的应用程序，大量数据变得晦涩难懂。

另外，GDB 并不提供有助于解密 GDB `print` 输出的任何工具。GDB 甚至不允许用户轻易创建一些有助于解密程序数据的工具。这样使得读取和理解调试数据的实践操作非常晦涩难解，尤其对于大型复杂项目。

对多数开发人员来说，定制 GDB `print` 输出（并使其更有意义）的唯一方法就是修改并重编译 GDB。但是，只有非常少的开发人员可以这样做。另外，该操作无法很好地扩展，尤其当开发人员还需要调试其他异构程序，并且这些程序包含同等复杂的调试数据的时候。

为解决该问题，GDB 的中标麒麟高级服务器操作系统 V6.4 现在与 Python `pretty-printers` 兼容。这使检索更有意义的调试数据成为可能，主要是通过自我评估、打印和格式化逻辑交给第三方 Python 脚本。

与 Python `pretty-printers` 的兼容性为您按照需要真正定制 GDB 输出提供了机会。这使 GDB 成为更多项目的更可行调试解决方案，因为您现在有了适应 GDB 的所需灵活性，而且更方便使用。另外，精通项目与特定编程语言的开发人员最有资格决定哪种输出是有意义的，从而允许他们改善输出的有用性。

Python `pretty-printers` 使用户能根据规格说明来自动检查、格式化并打印程序数据。这些规格说明被记为规则，并通过 Python 脚本实现。这提供了如下好处：

1) 安全性

为将程序数据传递给一套已注册的 Python `pretty-printers`，GDB 开发组将 `hooks` 添加至 GDB 打印代码。这些 `hooks` 的实现贯穿了对安全性的考虑：内置 GDB 打印代码仍是不完整的，所以它作为一个默认的低效率运行打印逻辑。就起本身而言，若无特定打印终端可用，GDB 仍按通常方式打印调试数据。这保证了 GDB 反向兼容，无 `pretty-printers` 需求的用户仍可继续使用 GDB。

2) 高度可定制

该新的"Python-scripted"方法使用户将所需知识尽可能多的提取到特定打印终端中。同样，项目通过针对用户需求的唯一方式拥有完整的打印终端脚本库。用户可为特定项目构建的打印终端数量没有限制。而且，用脚本定制调试数据的能力，使用户更快捷地重新使用和重新计划打印终端脚本——甚至它们整个库。

3) 易于学习

该方法最大的好处在于它降低了入门的难度。Python 脚本语言非常容易学习（相比之下，至少），而且有很大的在线免费文档库。另外，大多数程序员已有 Python 脚本语言的初级中级经验，或其它常用的脚本语言经验。

这里是 pretty printer 的一个小例子。考虑以下 C++程序：

```

//fruit.cc

enum Fruits {Orange, Apple, Banana};

class Fruit
{
    int fruit;

    public:
    Fruit (int f)
    {
        fruit = f;
    }
};

int main()
{
    Fruit myFruit(Apple);
    return 0;          // line 17
}
  
```

这是使用命令 `g++ -g fruit.cc -o fruit` 编译的。现在使用 GDB 检查该程序。

```

gdb ./fruit
[...]
(gdb) break 17
Breakpoint 1 at 0x40056d: file fruit.cc, line 17.
(gdb) run

Breakpoint 1, main () at fruit.cc:17
17
return 0;          // line 17
(gdb) print myFruit
  
```

```
$1 = {fruit = 1}
```

{fruit = 1}输出是正确的，因为那是'fruit'在数据结构'Fruit'中的内部表示。然而，人类不容易读懂这些内容，因为它很难说清楚整数 1 代表哪种水果。

为解决这个问题，请写出如下的 pretty printer:

```
#fruit.py

class FruitPrinter:
    def __init__(self, val):
        self.val = val

    def to_string (self):
        fruit = self.val['fruit']

        if (fruit == 0):
            name = "Orange"
        elif (fruit == 1):
            name = "Apple"
        elif (fruit == 2):
            name = "Banana"
        else:
            name = "unknown"
        return "Our fruit is " + name

    def lookup_type (val):
        if str(val.type) == 'Fruit':
            return FruitPrinter(val)
        return None

gdb.pretty_printers.append (lookup_type)
```

自下而上查看该打印。

`gdb.pretty_printers.append (lookup_type)`行将 `lookup_type` 函数添加至打印终端查找函数的 GDB 列表中。

`lookup_type` 函数负责检查要打印的对象类型，并返回相应的 pretty printer。该对象由 GDB 在参数 `val` 中传递。`Val` 类型表示 pretty printer 类型的属性。

`FruitPrinter` 是实际完成工作的地方。更具体的说，在该类的 `to_string` 函数中。该函数中，使用 python 字典语法 `self.val['fruit']` 来检索整数 `fruit`。然后使用该值确定名称。该函数所返回的字符串为将要为用户打印的字符串。

GDB 和 Python Pretty-Printers 白皮书提供了更多关于此特性的详细信息。此白皮书也包含如何编写您自己的 Python pretty-printer 的详细信息和示例，

6 性能分析 Profiling

开发人员分析程序性能，从而集中精力关注程序中最影响性能的方面。所收集的数据类型包括程序中消耗处理器时间最多的部分和内存分配情况。Profiling 从程序实际执行中收集数据。这样，程序正在执行的任务将影响被收集数据的质量。Profiling 时执行的任务应代表实际使用，这保证了在开发中会重视由程序实际使用引起的问题。

中标麒麟高级服务器操作系统 V6.4 包括众多工具(Valgrind、OProfile、perf 和 SystemTap)，这些工具收集 profiling 数据。每种工具适合于执行特定类型的 profile runs，在下面的章节中将对此进行讨论。

6.1 Eclipse 中的 Profiling

若要启动 (profile run) 概要分析，请导航到【运行】→【概要分析】。这样将打开【概要分析 为】对话框，您可以从中选择概要分析的工具。



图 6-1 概要分析 为

若要为 profile run 配置每个工具，请导航到【运行】→【概要分析配置】。这样将打开【概要分析配置】菜单。

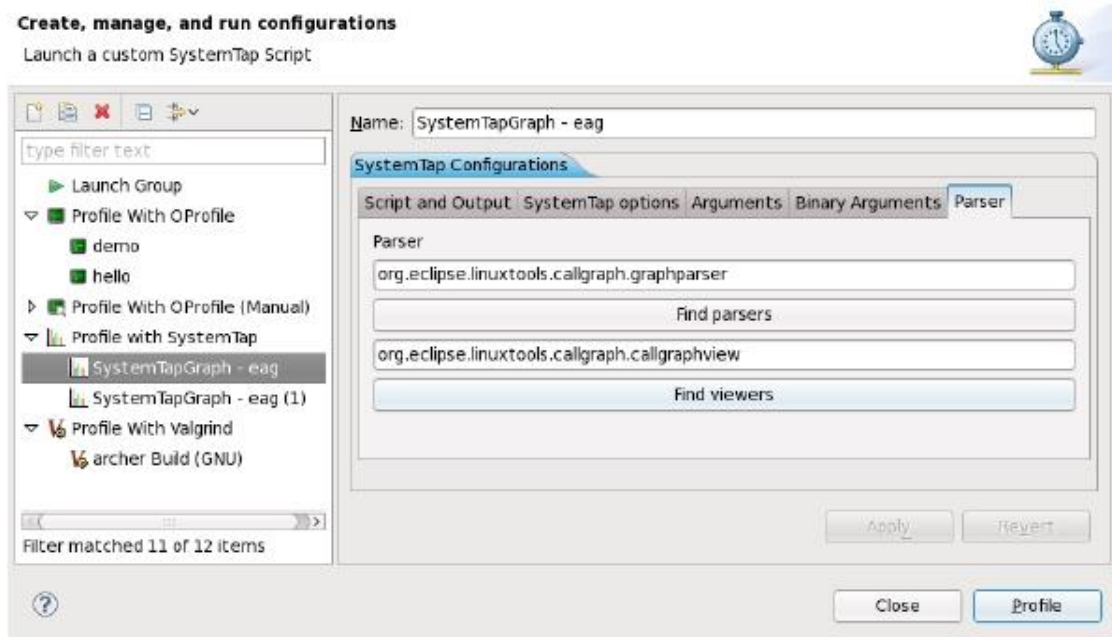


图 6-2 概要分析配置

更多关于 Eclipse 中配置和执行 profile run 每个工具的信息，请参阅章节“6.2.3 Eclipse Valgrind 插件”和“6.3.3 Eclipse OProfile 插件”，和章节“6.5Eclipse 调用图”。

6.2 Valgrind

Valgrind 是用于构建动态分析工具的测量框架，这些工具可用于详细分析应用程序。Valgrind 工具通常用于自动检测内存管理和线程问题。Valgrind 套件还包含必要时构建新 profiling 工具的软件。

Valgrind 为用户空间二进制文件提供测量工具来检查错误，如使用未初始化的内存、不正确的内存分配/释放、以及对系统调用的错误参数。常规用户可对多数二进制文件使用 profiling 工具；但是，与其他 profilers 相比，Valgrind profile runs 明显慢一些。若要分析二进制文件，Valgrind 重写可执行文件并重新检测重写的二进制文件。Valgrind 的工具对于寻找用户空间程序中与内存相关的问题非常有用，它不适合调试特定时间的问题，或内核空间检测或调试。

以前，Valgrind 不支持 IBM 系统 z 架构。但自 6.1 起，增加了对它的支持，这意味着 Valgrind 现在支持所有中标麒麟高级服务器操作系统 V6.x 所支持的硬件架构。

6.2.1 Valgrind 工具

Valgrind 软件包由以下工具组成：

1) memcheck

此工具通过检查对内存的读写来检测程序中的内存管理问题，并拦截所有对 malloc、new、free 和 delete 的调用。Memcheck 可能是最常用的 Valgrind 工具，因为使用其他方法都难于检测内存管理问题。这些问题很长一段时期内都无法检测到，最终引起难于诊断的故障。

2) cachegrind

Cachegrind 是缓存分析器，通过细致的模拟 CPU 中 I1、D1 和 L2 缓存，它能准确找到代码中的缓存故障。它显示缓存故障的数量、内存引用和源代码中每行的指令；Cachegrind 还提供每个函数、每个模块和整个程序的摘要，甚至能给出对每个机器指令的计数。

3) callgrind

与 cachegrind 一样，callgrind 可以模拟缓存行为。但是，callgrind 主要是为了记录可执行代码的调用图数据。

4) massif

Massif 是堆分析器；它测量程序使用的堆内存的大小，提供堆块信息，堆管理开销和栈大小。堆分析器有助于找到减少堆内存使用的方法。在使用虚拟内存的系统中，使用最优堆内存的程序造成内存耗尽的可能较小，而且，由于他们需要较少的页调度，因此运行较快。

5) helgrind

在使用 POSIX pthreads 的程序中，Helgrind 检测同步错误。如：

- a) POSIX pthreads API 的错误使用
- b) 由锁排序问题引起的可能死锁
- c) 数据竞争（即，不具备合理锁时访问内存）

Valgrind 还可让您开发自己的 profiling 工具。为实现这一目标，Valgrind 包括 lackey 工具，可将此示例工具作为开发自己工具的模板。

6.2.2 使用 Valgrind

valgrind 软件包和它的相关产品安装所有执行 Valgrind profile run 的必要工具。若要使用 Valgrind 分析程序，请使用：

```
valgrind --tool=toolname program
```

关于 *toolname* 的参数列表，请参阅“6.2.1 Valgrind 工具”。除 Valgrind 工具包外，*none* 还是 *toolname* 的有效参数，该参数可让您在 Valgrind 下运行程序，无需执行任何 profiling。这对调试或校准本身非常有用。

您也可命令 Valgrind 将所有信息发送至指定文件。若要这样做，请使用选项——`log-file=filename`。例如，要检查可执行文件的内存使用情况，并将此配置文件信息发送到 *output*，请使用：

```
valgrind --tool=memcheck --log-file=output hello
```

更多关于的信息，请参阅章节 6.2.4 Valgrind 文档，以及 Valgrind 工具包中的其他可用文档。

6.2.3 Eclipse Valgrind 插件

Eclipse Valgrind 插件将几个 Valgrind 工具集成到 Eclipse 中。这样 Eclipse 用户可将 profiling 功能无缝包含到它们的工作流程中。目前，Eclipse Valgrind 插件支持三种 Valgrind 工具：

Memcheck

Massif

Cachegrind

Eclipse Valgrind 插件是由 `eclipse-valgrind` 包提供的。更多关于此插件的信息，请参阅 Eclipse 帮助内容中的 Valgrind Integration User Guide。

6.2.4 Valgrind 文档

更多关于 Valgrind 的信息，请参阅 `man valgrind`。中标麒麟高级服务器操作系统 V6.4 也提供一本全面的书籍 Valgrind Documentation，该书有 PDF 和 HTML 两种形式，请参阅：

`file:///usr/share/doc/valgrind-version/valgrind_manual.pdf`

`file:///usr/share/doc/valgrind-version/html/index.html`

Eclipse 帮助内容中的 Valgrind Integration User Guide 也提供关于 Eclipse Valgrind 插件设置和使用的详细信息。本指南由 `eclipse-valgrind` 包提供。

6.3 OProfile

OProfile 是系统级的 Linux 分析器，运行开销很小。它包括核心驱动程序和收集原始样本数据的后台程序，以及一套将数据解析为有意义信息的工具。开发人员通常使用 OProfile 来决定哪段代码消耗最多的 CPU 时间，以及相应原因。

在 `profile run` 期间，OProfile 使用处理器的性能监视硬件。Valgrind 重写应用的二进制文件并分析它。另一方面，OProfile 只简单按现状概要分析运行中的应用程序。它设置性能监控硬件每隔 `x` 个事件就抽取一次样本（例如，缓存故障或分支指令）。每个样本还包含程序中所出现位置的信息。

OProfile 的 `profiling` 方法比 Valgrind 所消耗的资源更少。但是，OProfile 需要 `root` 权限。OProfile 在寻找代码中的“热点”并查找原因（例如，低缓冲性能、错误分支预测）时很有用。

使用 OProfile 涉及启动 OProfile 后台程序(`oprofiled`)，运行概要分析的程序，收集系统性能分析数据，将其解析为容易理解的形式。OProfile 为此过程的每个步骤都提供了几种工具。

6.3.1 OProfile 工具

最有用的 OProfile 命令如下：

1) `opcontrol`

此工具用于启动或停止 OProfile 后台程序并配置概要分析会话。

2) `opreport`

`opreport` 命令从 OProfile `profiling` 会话输出二进制图像摘要或单符号数据。

3) `opannotate`

`opannotate` 命令从 OProfile 会话的概要分析数据中输出带注释的源代码和/或集合。

4) `oparchive`

`oparchive` 命令生成含有可执行文件、调试文件和 OProfile 样本文件的目录。该目录可移动至另一台可脱机分析的机器（通过 `tar`）。

5) `opgprof`

与 `opreport` 一样，`opgprof` 命令从 OProfile 会话中输出给定二进制图像的 `profile` 数据。`opgprof` 的输出是 `gprof` 形式。

关于 OProfile 命令的完备列表，请参阅 `man oprofile`。有关每个 OProfile 命令的详细信息，请参阅它对应的 `man` 页。其他关于 OProfile 的可用文档，请参阅“6.3.4 OProfile 文档”。

6.3.2 使用 OProfile

`oprofile` 包和相关产品安装了执行 OProfile `profile run` 所有必要的工具。若要

命令 OProfile 对所有系统上运行的应用程序进行概要分析，并为共享库和使用库的应用程序对样本数据分组，请作为根目录运行如下命令：

```
opcontrol --no-vmlinux --separate=library --start
```

您也可不收集系统数据启动 OProfile 后台程序。若要这样做，请使用选项 `--start-daemon` 代替。`--stop` 选项停止数据采集，而 `--shutdown` 终止 OProfile 后台程序。

请使用 `opreport`、`opannotate`、或 `opgprof` 显示收集的性能分析数据。默认情况下，OProfile 后台程序收集的数据存储在 `/var/lib/oprofile/samples/`。

6.3.3 Eclipse OProfile 插件

OProfile 工具包作为插件，提供了强大的调用概要分析功能，这些功能很好地表现在 Eclipse 用户界面上。OProfile 插件具有如下益处：

1) Targeted Profiling

OProfile 插件使 Eclipse 用户可以概要分析特定的二进制文件，包括相关的共享库或内核模块，甚至不包括二进制文件。这样生成了对每个二进制文件、函数和符号，乃至源代码中不同行号的非常定向的、详细的使用结果。

2) 用户界面完全整合到 CDT 中

正如其他所有插件一样，该插件通过 Eclipse 显示 OProfile 结果。双击结果中的源代码行直接将用户带到 Eclipse 编辑器的相应行。这样可让用户通过单个界面组建、概要分析和编辑代码，为 Eclipse 用户提供了更方便的体验。另外，可用与 Eclipse 中 C/C++ 应用程序相同的方式启动并配置 `profile runs`。

3) 完全可定制的 Profiling 选项

Eclipse 界面允许用户使用 Oprofile 命令行使用工具中可用的所有选项配置他们的 `profile run`。该插件支持基于处理器调试寄存器（即计数器）的事件配置，以及不支持硬件计数器的内核或处理器的基于中断的性能分析。

4) 易用性

OProfile 插件通常为所有选项提供有用的默认设置，可用于大多数 `profiling runs`。另外，它还具有使用这些默认设置执行 `profile run` 的“一键分析”的功能。用户可自始至终分析应用程序的性能，或通过人工控制对话框选择特定代码段。

Eclipse OProfile 插件由 `eclipse-oprofile` 软件包提供。更多关于此插件的信息，请参阅 Eclipse 帮助内容中的 OProfile Integration User Guide（还由 `eclipse-oprofile`

软件包提供。)

6.3.4 OProfile 文档

更多关于 OProfile 的扩展信息，请参阅 `man oprofile`。中标麒麟高级服务器操作系统 V6.4 也提供两个 OProfile 的全面指南，请访问 `file:///usr/share/doc/oprofile-version/`:

1) OProfile 手册

关于包含 OProfile 设置和使用详细指导信息的全面手册，请访问 `file:///usr/share/doc/oprofile-version/oprofile.html`

2) OProfile 内部构件

OProfile 内部构件操作文档，对一些程序员来讲非常有用，这些程序员对 OProfile 补丁更新很感兴趣，该文档在：`file:///usr/share/doc/oprofile-version/internals.html`

Eclipse Help Contents 中的 OProfile Integration User Guide 也提供关于 Eclipse OProfile 插件设置和使用的详细信息。本指南由 `eclipse-oprofile` 软件包提供。

6.4 SystemTap

SystemTap 是非常有用的测量平台，用于探查 Linux 系统的运行进程和内核活动。若要执行探查动作：

- 1) 编写 SystemTap scripts，指定应触发特定操作（例如，打印、解析或处理数据）的系统事件（例如，虚拟文件系统读取，包传输）。
- 2) SystemTap 将脚本翻译为 C 程序，将程序编译进内核模块。
- 3) SystemTap 加载内核模块执行实际探查。

SystemTap 脚本对于监控系统操作和诊断系统问题非常有用，而且对系统正常运转的侵犯最小。您可以快速测量运行系统试验假设，无需重编译和重安装被测量代码。若要编译 SystemTap 脚本，该脚本探测 `kernel-space`，SystemTap 使用三个不同内核信息包中的信息：

`kernel-variant-devel-version`

`kernel-variant-debuginfo-version`

`kernel-variant-debuginfo-common-arch-version`

这些内核信息包必须与所探查的内核相匹配。另外，要为多个内核编译 SystemTap 脚本，还必须安装每个内核的内核信息包。

自中标麒麟高级服务器操作系统 V6.4 起新增了非常重要的功能：`--remote` 选项。这样可让用户在本地构建 `SystemTap` 模块，然后通过 `SSH` 远程执行此模块。使用此选项的语法是 `--remote [USER@]HOSTNAME`，将执行目标设置为特定的 `SSH` 主机，随意使用不同的用户名。可以重复该选项锁定多个执行目标。正如平常一样，本地完成 `Passes1-4` 组建脚本，然后 `pass5` 将模块复制到目标并运行。

以下章节描述了中标麒麟高级服务器操作系统 V6.4 中 `SystemTap` 的其他新功能。

6.4.1 SystemTap 编译服务器

中标麒麟高级服务器操作系统 V6.4 中 `SystemTap` 支持编译服务器与客户端部署。有了这种设置，网络中 `all` 客户系统的内核信息包都被安装在一个编译服务器主机上（或几个）。当客户端系统试图从 `SystemTap` 脚本编译内核模块时，客户端远程访问中央服务器主机上所需的内核信息。


正确配置和维护的 `SystemTap` 编译服务器主机提供如下益处：

- 1) 在用户可用该软件包前，系统管理员可验证内核信息包的完整性。
- 2) 编译服务器的身份可通过安全套接层(`SSL`)进行验证。`SSL` 提供加密网络连接，防止传输时的窃听或破坏。
- 3) 不同用户可以运行他们自己的服务器并对服务器进行可信授权。
- 4) 系统管理员可对网络中的一个或更多服务器进行授权，使所有用户都有权使用服务器。
- 5) 忽略未明确授权的服务器，防止任何虚假服务器和类似攻击。

6.4.2 对非特权用户的 SystemTap 支持

为了安全，企业环境中的用户很少有特权（即 `root` 或 `sudo`）访问他们自己的机器。另外，全部 `SystemTap` 功能也应对特权用户有所限制，因为这样做能提供完全控制系统的能力。

中标麒麟高级服务器操作系统 V6.4 中 `SystemTap` 为 `SystemTap` 客户提供了新选项功能：`--unprivileged`。该选项允许非特权用户运行 `stap`。当然，对试图运行 `stap` 的非特权用户应用了几种约束。

 注意：非特权用户是 `stapusr` 组的一个成员，但不是 `stapdev` 组的成员（也不是 `root` 的）。

加载由非特权用户创建的任何内核模块前，SystemTap 使用标准数字（密码）签名技术验证模块的完整性。每次使用 `--unprivileged` 选项时，服务器检查违反施加给非特权用户约束的脚本。若检查成功，服务器编译脚本并使用自生成的证书对结果模块进行签名。当客户端试图加载模块时，`staprun` 首先验证模块的签名，依据由 `root` 维持和授权的可信签名证书数据库，检查此签名。

一旦签名的内核模块验证成功，`staprun` 保证：

- 1) 该模块通过可信 `systemtap` 服务器创建。
- 2) 该模块使用 `--unprivileged` 选项编译。
- 3) 该模块满足施加于非特权用户的约束。
- 4) 自创建后，该模块未被篡改过。

6.4.3 SSL 与证书管理

中标麒麟高级服务器操作系统 V6.4 中，SystemTap 通过证书和公有/私有密钥对实现验证与安全。系统管理员有责任向可信服务器数据库增加编译服务器的凭证（即证书）。SystemTap 使用此数据库来验证客户端试图访问的编译服务器的身份，同样，SystemTap 也使用此方法验证编译服务器使用 `--unprivileged` 选项创建的内核模块。

6.4.3.1 用于连接的编译服务器授权

编译服务器首次在服务器主机上启动，编译服务器自动生成证书。此证书在 SSL 授权和模块签名期间，验证编译服务器的身份。

为使客户访问编译服务器（无论是从相同服务器主机，或是从客户机），系统管理员必须将编译服务器证书添加至可信服务器数据库。每个想要使用编译服务器的客户端主机都拥有这样的数据库。这样可让不同用户定制可信服务器的数据库，该数据库包含只为满足它们自己使用而授权的编译服务器列表。

6.4.3.2 用于模块签名（对非特权用户）的编译服务器授权

非特权用户只能加载已被签名和授权的 SystemTap 内核模块。对于如此认证的模块，它们必定是由编译服务器创建的，此编译服务器的证书存在于可信服务器数据库中，而此数据库必须保存在模块被加载的每个主机上。

6.4.3.3 自动授权

使用 `stap-server` 初始化脚本启动的服务器自动授权接收来自相同主机的所有客户端的连接。

通过其他方法启动的服务器自动授权接收来自同一主机上客户端的连接，此主机由启动服务器的用户运行。这是考虑到方便而实现的，假如客户端和服务器在相同主机上运行，则用户自动授权连接到他们自己启动的服务器。

无论何时 root 启动编译服务器，运行于相同主机的 all 客户都自动将服务器认为是授权过的。然而，中标麒麟高级服务器操作系统建议您不要这样做。

同样，通过 stap-server 初始化的编译服务器自动授权为它所运行主机上的可信签名者。若编译服务器通过其他方法初始化，则不会自动授权。

6.4.4 SystemTap 文档

关于 SystemTap 的详细信息，请参阅下列书籍：

SystemTap Beginner's Guide

SystemTap Tapset Reference

SystemTap Language Reference（由 IBM 提供文档）

若您安装了 systemtap 包，也可以在您本机上找到 SystemTap Beginner's Guide 和 SystemTap Tapset Reference。

file:///usr/share/doc/systemtap-version/SystemTap_Beginners_Guide/index.html

file:///usr/share/doc/systemtap-version/SystemTap_Beginners_Guide.pdf

<file:///usr/share/doc/systemtap-version/tapsets/index.html>

<file:///usr/share/doc/systemtap-version/tapsets.pdf>

章节 6.4.1 SystemTap 编译服务器、章节 6.4.2 SystemTap 对非特权用户的支持以及章节 6.4.3 SSL 和证书管理是从 SystemTap Support for Unprivileged Users and Server Client Deployment 白皮书中摘录的。该白皮书也提供了关于每个特性的详细说明，并举例说明它们在现实环境中的应用。

6.5 Eclipse 调用图

中标麒麟高级服务器操作系统 V6.4 还包括提供程序可视函数跟踪的 Eclipse-Callgraph 插件。这样可直观地查看概要分析的应用程序使用的选定（或所有）函数。

Eclipse-Callgraph 使用 SystemTap 在程序内执行全面的函数跟踪。同样，您需要安装 SystemTap 和所需的内核信息包。

关于 SystemTap 的详细信息，请参阅章节 6.4 SystemTap 及其他 SystemTap 文档。

此插件可使您直接分析 Eclipse IDE 中的 C/C++项目，提供多种运行时的详细信息，如：

- 1) 函数调用之间的关系
- 2) 每个函数的调用次数
- 3) 每个函数实例占用的时间（与程序的执行时间有关）
- 4) 所有函数实例占用的时间（与程序的执行时间有关）

6.5.1 启动带有 Eclipse-Callgraph 的 Profile

要使用 Eclipse-Callgraph 分析应用程序，只需右击项目并导航至【概要分析方式】→【Function callgraph】。这样将会打开对话框，您可以从该对话框中选择要进行分析的可执行文件。



图 6-3 Eclipse Callgraph Profile

选择要概要分析的可执行文件后，Eclipse-Callgraph 将询问要查询的文件。

默认情况下，

将选择项目中所有源文件。

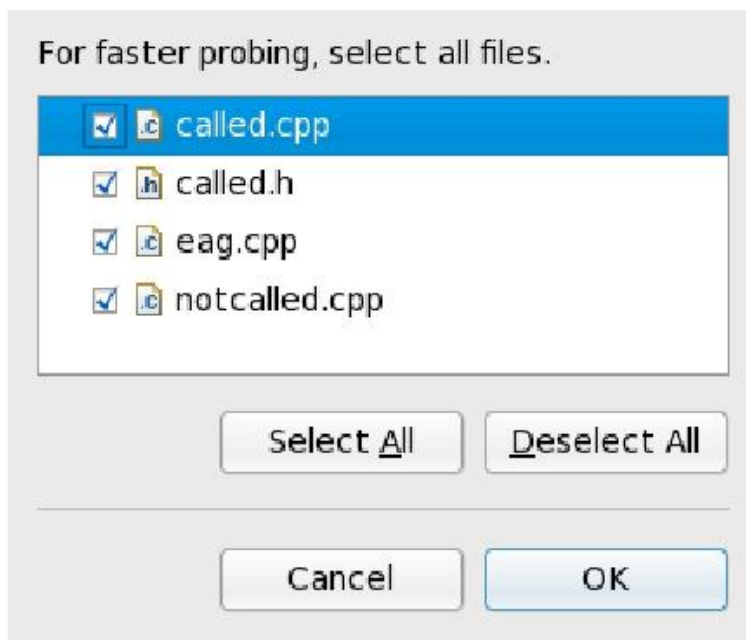


图 6-4 选择要查询的文件

6.5.2 Callgraph 视图

Callgraph 视图工具栏可让您选择透视图并执行其他函数。要以视图方式展示函数跟踪，请单击【视图菜单】按钮并导航到【Goto】。该菜单可让您在每个函数执行时暂停、单步调试或进行标记。

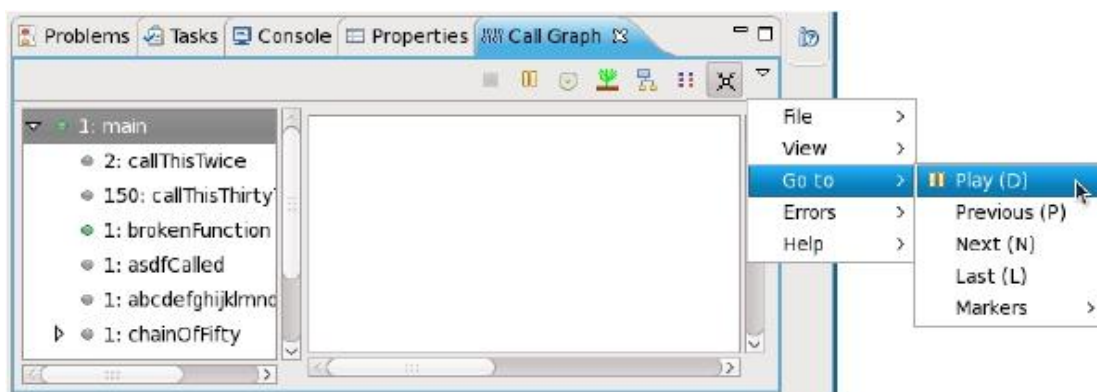


图 6-5 【View】→【Goto】

还可通过【视图菜单】保存或加载 profile run。若要这样做，请导航到【视图菜单】下的【File】，这样将显示与保存和加载该 profile run 相关的不同选项。

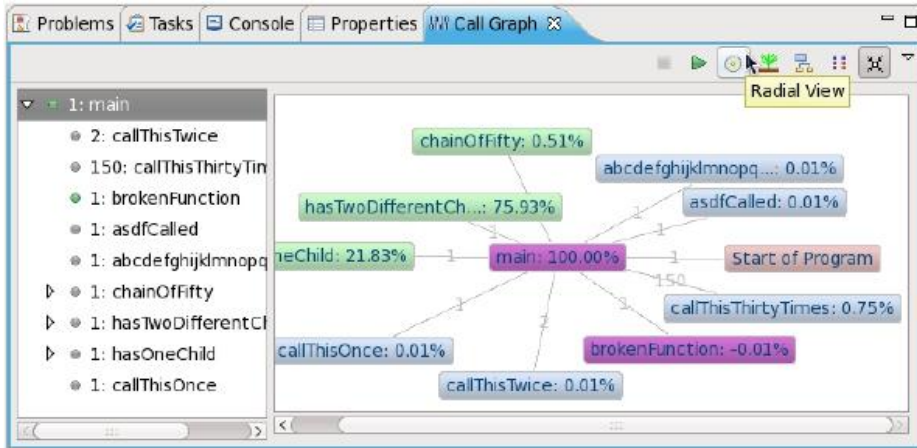


图 6-6 Radial View

Radial View 显示从 main()发出的的所有函数分支，将每个函数表示为一个节点。紫色节点表示程序终止于该函数。绿色节点表示函数调用含有嵌套函数，而灰色节点表示没有嵌套函数。双击一个节点，将显示它的父节点（着为粉色）和子节点。连接不同节点的连线也显示出 main()调用每个函数的次数。

Radial View 的左侧窗口列出了视图中显示的所有函数。若有嵌套函数，此窗口也允许您查看被嵌套的函数。绿色项目符号表示程序启动或终止该函数。

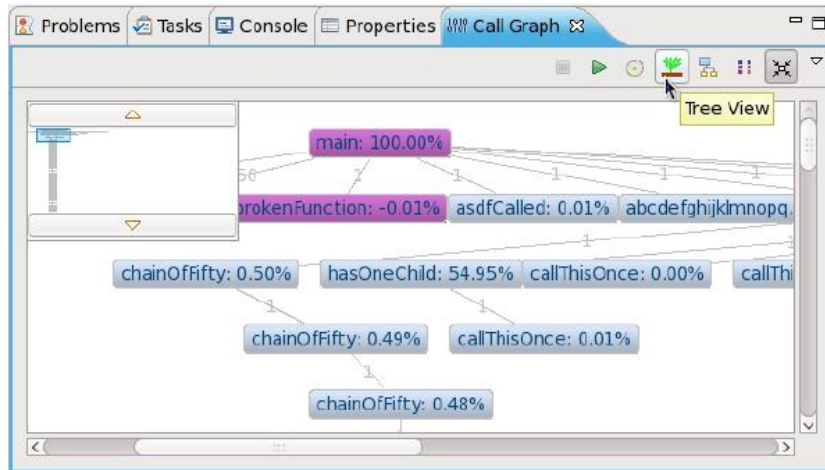


图 6-7 Tree View

Tree View 与 Radial View 相似，除了只显示选定节点的 *all* 后代节点（Radial View 只显示与选定节点调用深度为一步的函数）。Tree View 还包含缩略图查看器，可帮助您查看函数树的不同调用深度。

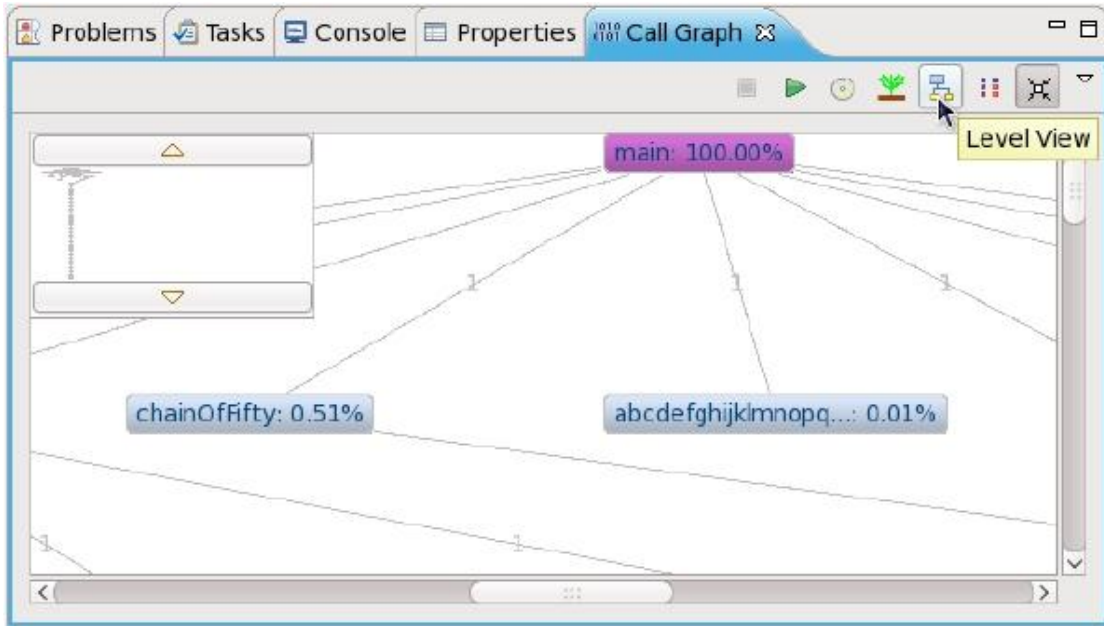


图 6-8 Level View

Level View 显示了从选定节点发出的所有函数调用和任意嵌套函数调用。但是，Level View 将所有调度深度相同的函数组合在一起，提供了更清晰的程序函数调用执行序列的视图。Level View 还允许您使用 More nodes above 和 More nodes below 按钮导航至不同的调用深度。



图 6-9 Thumbnail Viewer

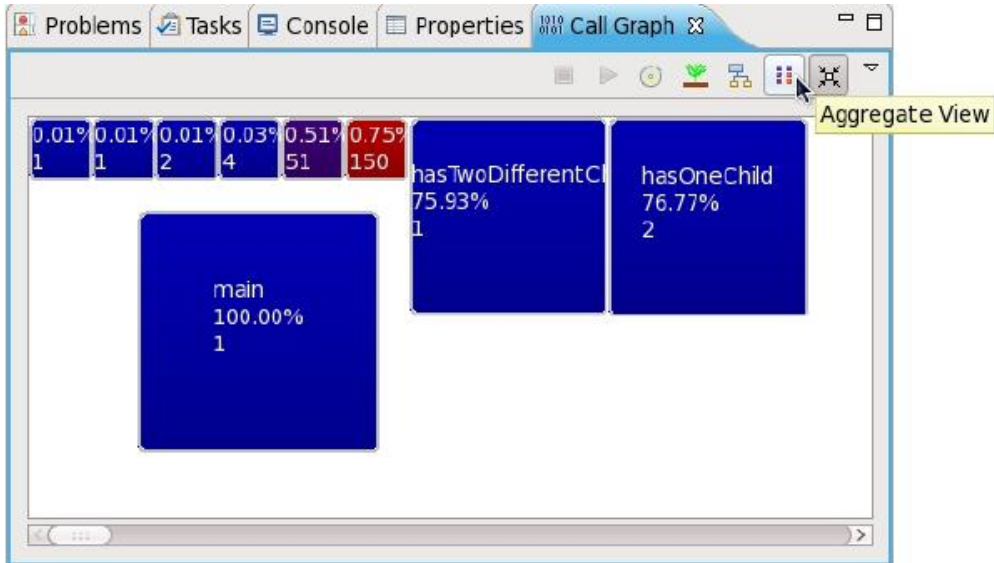


图 6-10 Aggregate 视图

Aggregate View 将所有函数表示为方框，每个方框的大小代表了函数与相对于程序整体运行时间的执行时间。颜色较暗的方框表示调用次数比其他函数更多的函数，例如，在图 6.10 Aggregate View 中，CallThisThirtyTimes 函数调用次数（150）最多。

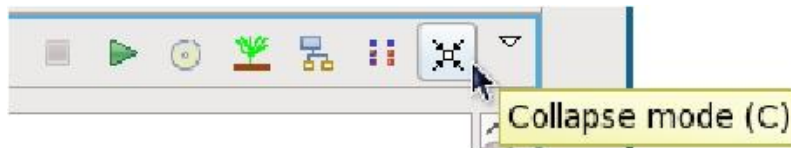


图 6-11 Collapse 模式

Callgraph 视图的工具栏也有 Collapse Mode 按钮。它将所有相同的函数分组（即，名字相同与调用历史相同的函数）到一个节点中。这样做有助于在程序中很多函数被多次调用时降低此视图的混乱。

Go to Code

若要从任何视图查看代码中的函数，请按 Ctrl 键，同时双击该节点。这样做将在 Eclipse 编辑器中打开对应的源文件，并在源文件中高亮显示函数的声明。

6.6 Linux(PCL)工具性能计数器和 perf

Performance Counters for Linux(PCL)是提供收集和分析性能数据的基于内核的新子系统。这些事件将根据系统中性能监控软件和软件配置而发生变化。中标麒麟高级服务器操作系统 V6.4 包含用于收集数据的此内核子系统，以及用于分析所收集性能数据的用户空间工具 perf。

PCL 子系统可用于测量硬件事件，包括已执行完指令和处理器时钟周期。它也可测量软件事件，包括主要页面错误和上下文切换。例如，根据已执行完指令的处理器计数和处理器时钟周期，PCL 计数器可以计算每个时钟指令数(IPC)。低 IPC 比率表示代码对 CPU 的使用效率很低。其他硬件事件也可用于诊断低 CPU 性能。

还可配置性能计数器记录数据样本。样本的相对频率可用于识别哪段代码对性能具有最大影响。

6.6.1 Perf 工具命令

有用的 perf 命令包括如下：

1) perf stat

perf 命令提供常见性能事件的全部统计信息，包括已执行指令和所用时钟周期。选项允许事件的选择，而不是默认测量事件。

2) perf record

此 perf 命令将性能数据记录在一个文件中，稍后可以使用 perf report 来分析该文件。

3) perf report

此 perf 命令从文件中读取性能数据并分析所记录的数据。

4) perf list

此 perf 命令列出特定机器上的事件。这些事件将基于系统中性能监控软件和软件配置而有所不同。

请使用 perf help 获取 perf 命令的完备列表。若要检索关于每个 perf 命令的 man 页信息，请使用 perf help 命令。

6.6.2 使用 Perf

使用基本的 PCL 架构来收集统计信息或程序执行样本是相对直接的方法。本节提供了全部统计信息和样本的简单示例。

若要收集关于 make 和子信息的统计信息，请使用如下命令：

```
perf stat -- make all
```

此 perf 命令将收集大量不同硬件和软件计数器。然后它将打印如下信息：

```
Performance counter stats for 'make all':
```

244011.782059	task-clock-msecs	#	0.925	
CPUs				
53328	context-switches	#	0.000	M/sec
515	CPU-migrations	#	0.000	M/sec
1843121	page-faults	#	0.008	M/sec
789702529782	cycles	#	3236.330	
M/sec				
1050912611378	instructions	#	1.331	IPC
275538938708	branches	#	1129.203	M/sec
2888756216	branch-misses	#	1.048	%
4343060367	cache-references	#	17.799	
M/sec				
428257037	cache-misses	#	1.755	M/sec
263.779192511	seconds time elapsed			

perf 工具也可记录样本。例如，要记录关于 make 命令和子信息的数据，请使用：

```
perf record -- make all
```

这样将打印出存储样本的文件，以及收集到的样本数目：

```
[ perf record: Woken up 42 times to write data ]
[ perf record: Captured and wrote 9.753 MB perf.data (~426109 samples) ]
```

然后，您可以分析 perf.data 来决定样本的相对频率。报告输出包括命令、对象和样本函数。请使用 perf report 来输出 perf.data 中的分析。例如，以下命令生成一份可执行文件的报告，报告生成过程需要较长时间：perf report --sort=comm

结果输出：

```
# Samples: 1083783860000
#
# Overhead Command
# -----
#
48.19%  xsltproc
44.48%  pdfxmtex
6.01%   make
0.95%   perl
0.17%   kernel-doc
0.05%   xmllint
0.05%   cc1
0.03%   cp
0.01%   xmlto
0.01%   sh
0.01%   docproc
```

```

0.01%   ld
0.01%   gcc
0.00%   rm
0.00%   sed
0.00%   git-diff-files
0.00%   bash
0.00%   git-diff-index
    
```

左列给出了样本的相对频率。此输出表明 `make` 在 `xsltproc` 和 `pdfxmltex` 中花费了大多数时间。若要降低完成 `make` 的时间，请关注 `xsltproc` 和 `pdfxmltex`。若要列出被 `xsltproc` 执行的函数列表，请运行：

```
perf report -n --comm=xsltproc
```

这会生成：

```

comm: xsltproc
# Samples: 472520675377
#
# Overhead SamplesShared Object      Symbol
# .....
#
  45.54%215179861044  libxml2.so.2.7.6      [.] xmlXPathCmpNodesExt
  11.63%54959620202   libxml2.so.2.7.6
    [.]xmlXPathNodeSetAdd__internal_alias
   8.60%40634845107  libxml2.so.2.7.6      [.]xmlXPathCompOpEval
   4.63%21864091080  libxml2.so.2.7.6      [.]xmlXPathReleaseObject
   2.73%12919672281  libxml2.so.2.7.6      [.]xmlXPathNodeSetSort__internal_alias
   2.60%12271959697  libxml2.so.2.7.6      [.]valuePop
   2.41%11379910918  libxml2.so.2.7.6      [.]xmlXPathIsNaN__internal_alias
   2.19%10340901937  libxml2.so.2.7.6      [.]valuePush__internal_alias
    
```

6.7 ftrace

`ftrace` 框架为用户提供几种跟踪功能，可通过比 `SystemTap` 更容易的界面使用这几种跟踪功能。该框架使用 `debugfs` 文件系统的一系列虚拟文件，这些文件可启动特殊的跟踪器。`ftrace` 函数跟踪器仅实时输出内核调用的每个函数，`ftrace` 框架内的其他跟踪器也能用于分析唤醒等待，任务切换，内核事件等等。

您也可为 `ftrace` 增加新的跟踪器，使它成为更灵活的内核事件分析解决方案。`ftrace` 框架对调试和分析发生在用户空间之外的等待时间和性能问题非常有用。不像本指南中所描述的其他分析器，`ftrace` 是内核的内置特色。

6.7.1 使用 ftrace

中标麒麟高级服务器操作系统 V6.4 内核已使用 `CONFIG_FTRACE=y` 选项

进行配置。此选项提供 ftrace 所需的接口。若要使用 ftrace，请按如下方法安装 debugfs 文件系统：

```
mount -t debugfs nodev /sys/kernel/debug
```

所有 ftrace 工具都位于 /sys/kernel/debug/tracing/ 中。请查看 /sys/kernel/debug/tracing/available_tracers 文件，可找到哪些跟踪器对您的内核是可用的。

```
cat /sys/kernel/debug/tracing/available_tracers  
power wakeup irqsoff function sysprof sched_switch initcall nop
```

若要使用特定跟踪器，请将其写入 /sys/kernel/debug/tracing/current_tracer。例如，wakeup 跟踪并记录任务唤醒后调度最优先任务所需最长时间。使用它：

```
echo wakeup > /sys/kernel/debug/tracing/current_tracer
```

要启动或停止跟踪，请写入 /sys/kernel/debug/tracing/tracing_on。如：

```
echo 1 > /sys/kernel/debug/tracing/tracing_on (启用跟踪)  
echo 0 > /sys/kernel/debug/tracing/tracing_on (禁用跟踪)
```

可以从如下文件中查看跟踪结果：

```
/sys/kernel/debug/tracing/trace
```

此文件包含人们可读的跟踪结果。

```
/sys/kernel/debug/tracing/trace_pipe
```

此文件包含与 /sys/kernel/debug/tracing/trace 相同的结果，但它意味着输出到命令中。不像 /sys/kernel/debug/tracing/trace，读取该文件使用该文件的结果。

6.7.2 ftrace 文档

ftrace 框架被完全记录在下面的文件中：

ftrace - Function Tracer:file:///usr/share/doc/kernel-doc-version/Documentation/trace/ftrace.txt

function tracer guts:file:///usr/share/doc/kernel-doc-version/Documentation/trace/ftrace-design.txt