基于 Eclipse 平台的交叉调试方案①

曹广旭[1], 孔 平[2]

[1](江苏自动化研究所 电子设备事业部, 连云港 222006)
[2](山东电力集团济宁供电公司, 济宁 272000)

摘 要: Eclipse 平台提供了开发 C/C++程序的插件 CDT, 但是针对嵌入式软件的交叉调试, CDT 需要过多的用户参与. 设计一种基于 Eclipse 平台的交叉调试方案, 通过向 Eclipse 平台无缝集成交叉调试插件, 对调试所需参数进行自动配置, 充分利用 Eclipse 平台的扩展性, 使用户可以直观的观察目标机信息, 从而实现交叉调试对用户的透明化, 达到自动化调试的目的. 通过实验表明, 该交叉调试方案能够简化用户的控制, 提高交叉调试的效率.

关键词: Eclipse 平台; 插件; CDT; 交叉调试

Cross Debugging Program Based on Eclipse Platform

CAO Guang-Xu[1], KONG Ping[2]

[1](Department of Electronic Equipment, Jiangsu Automation Research Institute, Lianyungang 222006, China)
[2](Jining Power Supply Company, Shandong Electric Power Group, Jining 272000, China)

Abstract: The Eclipse platform provides CDT plug-in for C/C++ program development, but for the embedded software cross debugging. CDT requires too much users to participate. We design a cross debugging program based on the Eclipse platform. It automatically configures parameters needed for debugging by means of seamless integrating a cross debugging plug-in to the Eclipse platform. It can make full use of extensibility of Eclipse platform to make the target machine information visible, and achieve automatic debugging purposes. Experiments show that the cross debugging program can simplify the user control, and improve the efficiency of cross debugging.

Key words: Eclipse platform; plug-in; CDT; cross debugging

在软件开发过程中, 软件调试是必不可少的环节, 调试时间大约占开发总时间的 50%, 甚至更多, 软件调试就是为了发现并排除软件程序中的错误, 而通过某种方法控制被调试程序的执行过程, 以便随时查看和修改被调试程序执行状态的方法[1]. 一个好的开发环境能大大提高调试速度, 加快开发进度.

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台. 就其本身而言, 它只是一个框架和一组服务, 用于通过插件组件构建开发环境. Eclipse 的插件机制是轻型软件组件化架构, 插件架构能够支持将任意的扩展加入到现有环境中[2]. CDT 是一组用来开发和调试 C/C++程序的插件, 致力于为 Eclipse 平台提供功能完全的 C/C++集成开发环境[3]. 使用 CDT 可以开发和交叉调试目标机应用程序, 但是存在一些不足:

(1)必须在目标机端手动发起调试代理(GDB Server). 对于嵌入式开发来说, 目标机端不存在显示设备或输入设备的情况较为普遍, 那么此时交叉调试非常不便;

(2)构建程序后, 需要手动下载并发起调试. 每次新建“调试配置”后要手动填写目标机 IP 和端口号, 使用“调试配置”发起调试时, 必须手动选择被调试的项目, 调试手段繁琐;

(3)CDT 调试插件不提供目标机端的任务信息, 不具备直观的任务查看及搭接调试功能.

针对以上不足, 设计开发了一个用于嵌入式开发中交叉调试的 Eclipse 插件, 简化了交叉调试的繁琐步

① 收稿时间:2014-05-06;收到修改稿时间:2014-06-20

骤, 实现了自动化调试功能, 提高了嵌入式开发调试的效率.

1 交叉调试技术

交叉调试系统通常包括主机调试器、目标机上的调试代理、调试协议三部分. 嵌入式系统调试中, 调试器能够通过某种方式(远程)控制目标机上被调试程序的运行模式, 并且具备查看和修改目标机上的内存、寄存器以及被调试程序中的变量等功能[4]. 主机调试器实现对源文件、目标文件和符号表的访问处理, 接收用户输入的调试命令, 并根据调试协议封装成调试命令请求包发送给调试代理, 同时, 接收调试代理返回的调试信息, 以获取目标程序的当前运行状态[5]. 目标机上的调试代理负责监控被调试目标程序的运行状态, 根据调试协议, 接收并解析调试命令, 将目标程序状态信息返回主机端. 调试协议则规定了主机调试器与调试代理之间的调试命令和调试信息的数据格式及通信过程. 嵌入式系统交叉调试体系结构如图 1 所示.

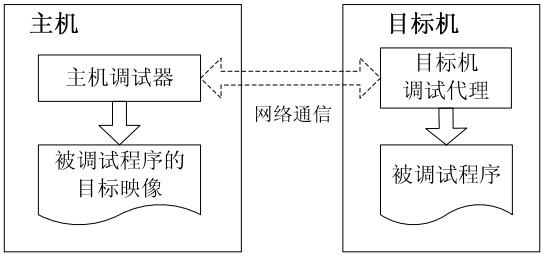

图 1 交叉调试体系结构图

2 Eclipse插件技术

Eclipse 是以 OSGi[6]规范为基础实现了一个小内核, 并集成了许多个插件共同形成的开发环境. 插件是一种遵循一定规范的应用程序接口编写出来的程序, 插件技术使得系统结构清晰, 增强可维护性和移植性, 系统的功能易于调整. 每个插件之间的连接关系是通过声明扩展点和扩展其它插件中的扩展点实现的. 构成 Eclipse 平台的子系统以插件的形式实现, 这些子系统在平台运行时引擎的基础上构建而成, 图 2 描述了各子系统的组成形式.

当 Eclipse 启动时, 平台运行时(Platform Runtime)系统将扫描 plugins 目录下所有的插件, 然后将插件添加到注册表中, 此时并不激活系统中所有插件, 而是缓存各个插件配置文件中的数据信息, 当用户真正使用到这个插件的功能时, 平台运行时系统才真正将插件调入内存并激活[7]. 这样可以避免未使用插件的资源消耗, 当插件不再使用后, 它就会在适当的时机被清除出内存.

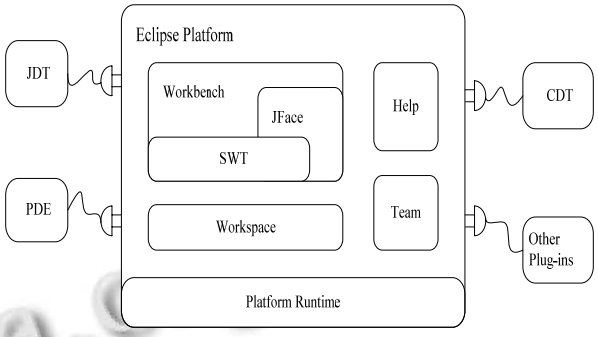

图 2 Eclipse 平台体系结构图

扩展点在 Eclipse 中作为一个松耦合的功能模块广泛使用. 插件在插件清单中声明扩展点, 提供接口和相关类的最小集合供他人使用. 其他插件声明该扩展点的扩展项, 实现合适的接口, 并引用提供的类或基于提供的类进行创建. CDT 提供了许多扩展点使用户可以添加自定义功能, 以符合用户的需求.

3 交叉调试方案设计

3.1 调试代理设计

运行在主机端的调试插件需要与目标机进行网络通信, 而调试服务程序是一个运行在目标机的后台进程(称为 Mon), 负责接收主机的指令并进行解析和在目标机发起相应的操作. Mon 的作用是服务于主机端的调试, 功能包括: 维护与主机的连接、向主机提供目标机的进程信息、发起 GDB Server 和调试任务等.

根据主机与 Mon 进行通信的功能需求, 设计了 4 种类型的 UDP 报文, 如表 1 所示.

表 1 报文类型

报文类型	报文用途	返回信息
Q 报文	主机指示与目标机进行连接	连接是否成功
P 报文	获取目标机运行的进程信息	目标机进程信息
K 报文	杀掉目标机的进程	操作是否成功
A 报文	开始调试	目标机调试使用的端口号等

Q 报文: 主机端发送 Q 报文给目标机的 Mon 程序, 指示主机正在建立与目标机的连接, 主机与目标机分

别保存各自的 Socket 及其他的初始化工作.

P 报文: 主机端有一个目标机进程的视图, 为获取目标机的所有进程, 主机向目标机发送一个 P 报文, Mon 将进程 ID、进程号等信息返回给主机.

K 报文: 用户可以在目标机进程视图中选择杀掉一个或多个进程, 主机向 Mon 发送包含相关进程号的 K 报文, Mon 执行相应的操作并返回结果.

A 报文包含了调试时主机使用的端口号、目标文件等信息, Mon 接收到 A 报文后使用主机的 IP、端口、目标文件等发起 GDB Server, 并将目标机使用的端口返回给主机, 主机使用目标机的端口发起 GDB 进行交叉调试.

3.2 网络文件系统

传统的交叉调试需要将主机端构建的目标文件下载到目标机后才能进行调试, 此下载过程通常采用网络传输的方式. 由于物理网卡传输速度限制和网络稳定性等原因, 通过网络传输目标文件并不能达到理想的速度, 因此每次对目标文件发起调试都需要花费一定时间等待目标文件下载完成. 对于频繁的调试操作, 浪费的下载时间非常可观.

网络文件系统(NFS)是文件系统之上的一个网络抽象, 允许远程客户端以与本地文件系统类似的方式通过网络进行访问[8], 如图 3 所示.

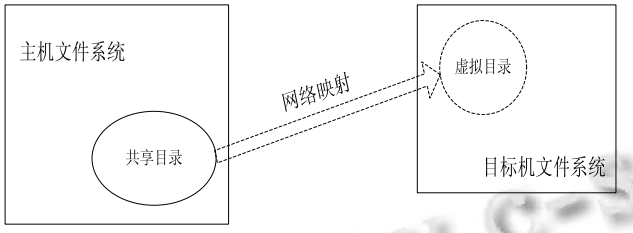

图 3　网络文件系统示意图

在主机端首先运行 NFS 服务, 将主机的目标文件所在目录设定为共享目录, 在目标机端将主机的共享目录挂载到目标机的文件系统中, 此时主机和目标机共享同一个目录. 每次主机构建生成目标文件, 目标机的文件系统立刻就可以发现, 因此可以节省网络传输目标文件的时间, 从而显著的减少调试等待的时间.

3.3 交叉调试插件设计

交叉调试插件包含了四个部分: 目标机进程视图、网络通信模块、调试发起模块和 GDB 调试器(GNU Debugger). 目标机进程视图用于向用户实时显示目标机运行的进程信息, 包括系统进程和发起的调试进程等. 网络通信模块负责建立与目标机的 Socket 链接, 并接收和发送与目标机间的网络报文; 调试发起模块通过 Eclipse 和 CDT 提供的 API 修改、重写调试发起文件的配置信息, 并使用此配置调用 GDB 发起调试. 各模块之间的关系如图 4 所示.

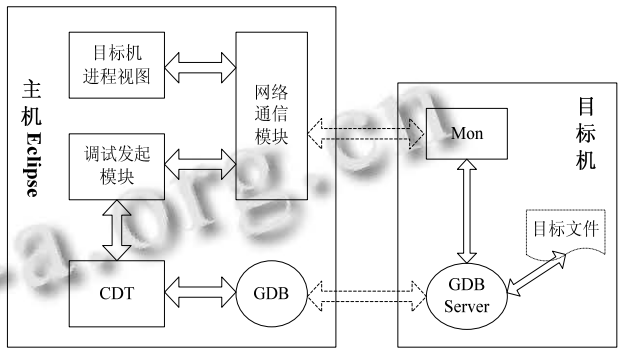

图 4　交叉调试插件模块之间的关系

交叉调试插件的运行流程如下: (1)联机. 首先用户通过网络通信模块与目标机建立连接, 调试插件在后台使用 IP 地址和默认端口号向目标机发送 Q 报文, 目标机调试代理 Mon 接收到报文后返回一个应答报文, 主机接收到应答后立刻再发送一个 P 报文, 由目标机将进程信息返回. 从用户的角度看, 联机成功后可以立刻看到目标机的所有进程; (2)调试. 当联机成功后, “调试”的按钮被使能, 当用户点击“调试”按钮时, 调试插件通过网络通信模块向目标机 Mon 发送一个 A 报文, 报文中包含主机调试时使用的端口号, Mon 接收到 A 报文后, 获取目标机中一个空闲的端口交给调试插件, 并使用此端口号发起一个 GDB Server 等待 GDB 进行连接调试. 调试插件收到的目标机端口号后, 使用 CDT 提供的 API 接口组建一个调试配置, 将目标机的 IP、端口号和目标文件名等信息附加到调试配置中, CDT 负责根据调试配置组建一个发起 GDB 的命令行并发起调试任务, 当 GDB 与 GDB Server 建立连接后, 用户就可以进行调试.

交叉调试插件的联机运行流程如图 5(a)所示, 调试运行流程如图 5(b)所示. 调试插件提供了直观的目标机进程信息, 并可以一键进入交叉调试的界面, 将用户手动发起 GDB Server、填写目标机 IP 和端口等操作屏蔽掉, 使用户从诸多繁琐操作中解脱出来, 可以显著提高用户的劳动生产率.

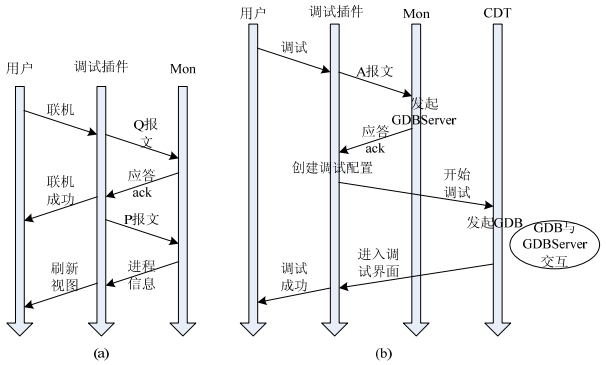

图 5 交叉调试方案的时序流程

3.4 扩展点实现

插件可以通过其他插件定义的扩展点来增加系统新的功能, 提供扩展点的插件称为父插件, 实现扩展点的插件称为子插件, Eclipse 的插件结构是由父插件管理子插件, 通过扩展点的连接实现插件之间的交互. 如果父插件预留了某一个功能接口, 那么子插件就可以通过此接口添加自定义功能.

交叉调试系统插入接口是以扩展点的形式呈现的, 各功能插件只需对这些扩展点加以扩展即可. 交叉调试模块是以插件的形式添加到集成开发环境中, 具体使用的扩展点如表 2 所示.

表 2 应用的扩展点

扩展点	功能描述
org.eclipse.ui.commands	添加菜单项和工具栏的项
org.eclipse.ui.handlers	“连接目标机”和“调试”动作的实现
org.eclipse.ui.menus	添加菜单
org.eclipse.ui.views	添加目标机进程视图
org.eclipse.ui.viewActions	为视图添加下拉菜单和工具栏

扩展点“org.eclipse.ui.commands”、“org.eclipse. ui. handlers”和“org.eclipse.ui.menus”共同实现了向 Eclipse 系统中添加自定义的菜单项和工具栏项, 并控制这些项显示的时机, 如 “调试”和“连接目标机”按钮动作的实现.

扩展点“org.eclipse.ui.views”和“org.eclipse. ui.view Actions”向 Eclipse 系统中添加一个目标机视图, 并为此视图单独实现一个工具栏. 目标机视图使用表格“org.eclipse.jface.viewers.TableViewer”作为内容以显示目标机中的进程信息. 视图工具栏包含三个动作: “刷新”、“搭接进程”和“杀掉进程”, 用来实现对目标机进程的控制.

调试发起模块通过对“ILaunchConfiguration”的属性设置是整个调试过程的关键. “Ilaunch Configuration”包含几个重要属性: ATTR_HOST(主机 IP)、ATTR_PORT(端口号)、ATTR_ DEBUGGER _START_MODE(调试模式)和 ATTR_REMOTE _TCP(远程 TCP 方式)等. 用户通过扩展点提供的按钮触发调试时才会对“ILaunchConfiguration”属性进行配置.

4 实验结果与系统实现

为了检验上述交叉调试方案的有效性, 将使用 Eclipse CDT 和交叉调试插件两种调试方法进行对比, 实验结果证明了交叉调试插件可行性与优越性. 测试环境的配置如表 3 所示.

表 3 测试环境配置

测试环境	参数
主机操作系统	Windows XP
主机集成开发环境	Eclipse 3.6.2
C/C++开发工具集	CDT 7.0
目标机操作系统	Linux 2.6.30
主机与目标机连接方式	以太网络

使用传统的交叉调试方式时, 首先将目标文件下载到目标机, 再手动发起 GDB Server 等待主机连接, 然后通过 CDT 填入 IP 和端口号后才可以在主机端发起调试. 使用交叉调试插件只需联机和下载两步操作即可发起调试.

交叉调试插件从功能上简化了用户的操作, 使调试工作更加简便直观, 与传统的交叉调试方式相比, 交叉调试方案的优点如表 4 所示.

表 4 交叉调试方案比较

对比项	传统的交叉调试	交叉调试插件
下载目标文件	手动	自动
调试启动时间	> 5s	< 2s
观察目标机任务	不具备	直观
操作复杂度	繁琐	简便

在 Eclipse 中使用交叉调试插件进行调试的界面如图 6 所示, 其中包含了 A、B 和 C 三个方框:

(1)A 框中显示的“下载”选项, 用户点击时, 可以直接进入调试界面.

(2)B 框中显示的是目标机的进程视图, 进程视图包含了一个工具栏和一个两栏的表格. 工具栏包含三个按钮: “刷新”、“搭接进程”和“杀掉进程”, 表格的第一栏表示目标机进程的 ID 号, 第二栏表示进程名称.

(3)C 框中显示的是添加到 Eclipse 工具栏的两个按钮: “连接目标机”和“显示目标机进程视图”. 点击“连接目标机”会打开一个对话框, 用户可以输入目标机的 IP 地址. 点击“显示目标机进程视图”按钮会打开 B 框中显示的目标机进程视图.

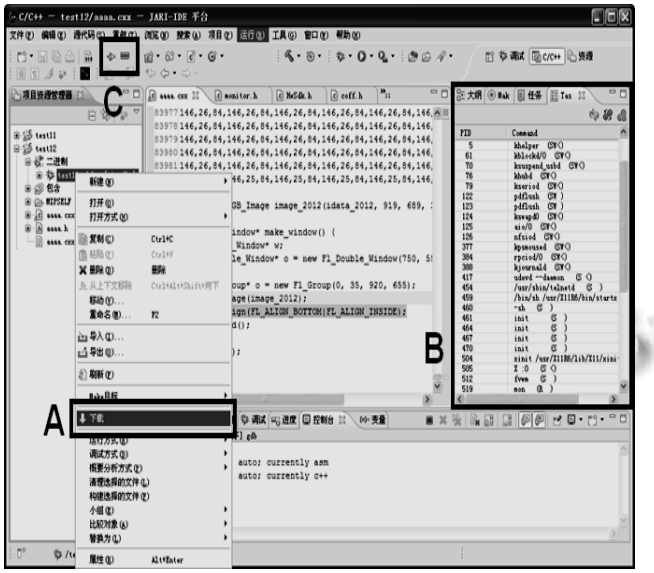

图 6　交叉调试运行界面

5　结语

文中阐述了 Eclipse 下的插件开发机制, 分析了 CDT 交叉调试插件在 Linux 系统下使用存在的不足并给出了解决问题的方法, 通过系统的架构设计、运用应用程序框架封装技术等, 实现了对 CDT 插件的调试功能扩展, 提高了嵌入式软件的调试开发效率.

通过对 Eclipse 和 CDT 的扩展, 简化了交叉调试的手段, 但仍然存在一些不足, 如目前调试插件只针对单核 CPU 调试, 对于多核 CPU 的支持还不够, 目标机进程视图的设计较简单, 显示的目标机信息不够全面, 这些问题需要在下一步的工作中继续改进.

参考文献

1 李志丹.嵌入式软件调试方法研究.计算机与数字工程, 2012,40(7):157–159.

2 Wikipedia. Eclipse 集成开发环境. [2014-03-10]. http://zh.wikipedia.org/wiki/Eclipse.

3 IBM. Eclipse CDT(C/C++ Development Tooling). http://www.eclipse.org/cdt. 2014.

4 刘晓升.嵌入式技术基础与实践.北京:清华大学出版社, 2011.

5 夏安祥,史浩山,阮园,等.一种可重定向的交叉调试器实现方法.计算机应用研究,2011,28(10):3735–3738.

6 OSGi Alliance. The OSGi Architecture. [2014-02]. http://www.osgi.org/Technology/WhatIsOSGi.

7 刘芳,臧威.基于 Eclipse 平台的嵌入式交叉调试环境 MRTOS.电子科技,2013,26(5):18–21.

8 Wikipedia. Network File System. [2014-03]. http://en.Wikipedia.org/wiki/Network_File_System.