计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(8):101−108 [doi: 10.15888/j.cnki.csa.007032]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于国产平台的智能跟踪调试技术①

张大方, 胡先浪, 王立杰

(江苏自动化研究所, 连云港 222006)
通讯作者: 张大方, E-mail: zdfxgd@126.com

摘　要: 随着龙芯、飞腾、申威等国产处理器的诞生, 目前涌现出了一批支持国产处理器的国产操作系统, 如JARI-Works、中标麒麟等. 但是, 与国产软硬平台配套的调试工具发展却相对滞后, 严重制约着国产平台下的软件调试效率. 本文针对国产软硬平台, 设计了一种基于 Eclipse 平台的智能调试方案. 通过在 Eclipse 平台下集成面向国产平台的工具链和智能跟踪调试插件, 自动启动并智能跟踪调试流程, 从而实现调试功能的图形化和智能化, 进而构建出一套界面友好、使用便捷、自主可控的集成开发环境. 通过实验表明, 该调试方案能够有效简化调试流程, 提高国产平台下的软件调试效率.

关键词: Eclipse 平台; 插件; 国产软硬件平台; 智能跟踪调试

引用格式: 张大方,胡先浪,王立杰.基于国产平台的智能跟踪调试技术.计算机系统应用,2019,28(8):101–108. http://www.c-s-a.org.cn/1003-3254/7032.html

Intelligent Tracking and Debugging Technology Based on Domestic Platform

ZHANG Da-Fang, HU Xian-Lang, WANG Li-Jie

(Jiangsu Automation Institute, Lianyungang 222006, China)

Abstract: With the birth of Loognson, Phytium, and SW domestic processor, a number of domestic operating systems supporting domestic processors have emerged, such as JARI-Works, Neokylin, and so on. The development of debugging tools matched with domestic soft and hard platforms is lagging behind, restricting the efficiency of software debugging under the domestic platform. We design an intelligent debugging project aiming at domestic software and hardware platform, integrate tool chains and intelligent tracking and debugging plug-ins for domestic platforms, then we can auto start and intelligently track debugging process, to realize the intelligent debugging in graphic. We build a UI-friendly, easy to use, and self-control integrated development environment. Experiment shows that the project can simplify the debugging flows and improve software debugging efficiency on domestic platform.

Key words: Eclipse platform; plug-in; domestic software and hardware platform; intelligent tracking and debugging

引言

软件调试是软件开发流程中不可或缺的重要环节, 是为了能在开发阶段及时发现程序中的错误并修正. 根据国内外研发人员的经验, 软件的调试时间一般能够占到研发时间总量的一半以上[1]. 正所谓“工欲善其事, 必先利其器”, 集成开发环境便是软件调试的“利器”[2]. “利器”的易用性及功能的全面性不仅仅会影响国产平台下软件调试的效率, 同时也会在一定程度上影响国产软硬件的规模化应用进程.

Eclipse 是一个开源的、可扩展的、用 Java 语言实现的开发平台, 就其本身而言, 它不是一个一整块的程序, 而是一个包含了插件载入器、被数百个甚至更多的插件所包围的小内核[3]. 该内核为插件的加载和运行提供了环境, 每个插件则以结构化的方式在整体中

① 收稿时间: 2019-01-31; 修改时间: 2019-02-27; 采用时间: 2019-03-18; csa 在线出版时间: 2019-08-08

提供服务. 因此, 只需要通过增删插件, 就可以根据需求定制所需的调试环境[4].

CDT(C/C ++ Development Toolkit) 是 Eclipse 平台下一组用来开发、调试 C/C++程序的插件, 使用 CDT 插件可以开发、调试用 C/C++编程语言实现的应用程序[5]. 但是以该插件为代表的传统调试方式尚存在一些不足, 例如:

(1) 对远程调试等常见的调试方式支持有限, 在发起调试时往往需要开发人员手动完成填写 IP 地址和端口号、上传待调试程序、启动调试服务器等工作, 调试准备工作繁琐;

(2) 对多平台编译工具链的支持有限, 当开发环境下存在多个编译工具链时, 需要开发人员手动选择与该工具链对应的调试工具和动态链接库的搜索路径等, 不支持一键式调试;

(3) 对错误定位的支持有限, 开发人员为提高调试效率往往会设置断点并让一部分代码自动执行, 在传统的调试方式下如果这部分代码在自动执行的过程出现异常, 则需要重新设置断点并再次发起调试, 严重迟滞了调试进度;

(4) 对国产软硬件平台的支持有限, 社区开源的 Eclipse 集成开发环境仅对 X86 平台提供支持, 国产系统厂商默认提供的集成开发环境则绑定了各自的软硬平台, 对其它国产平台的支持力度不足, 不具备通用性和易用性.

因此, 针对国产平台飞速发展的现状和 CDT 插件的上述不足之处, 本文面向龙芯、申威等国产处理器以及中标麒麟、深度、JARI-Works 等国产操作系统, 设计开发了一个支持国产平台并提供智能跟踪调试功能的功能插件, 增强了国产平台下的调试功能, 大大提高了调试效率.

1 反向调试技术

反向调试技术[6]是实现程序智能跟踪与调试的基础, 只有具备了能在程序执行的整个历史中来回穿梭的能力, 才有可能精确定位到程序故障发生的位置.

目前的主流调试器 GDB 除了提供传统的单步执行调试的方式以外, 在 GDB 7 版本之后, 也提供了反向调试的功能, 能够通过 reverse-next、reverse-nexti 等指令对程序指令执行历史进行保存与回溯[7]. GDB 的反向调试功能原理如图 1 所示.

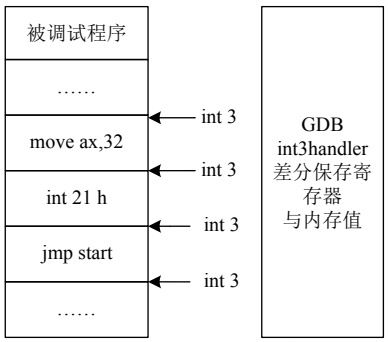

图 1 GDB 反向调试示意图

在图 1 中可以看到 GDB 对每条指令都加入了断点 (int 3), 然后在断点发生时差分保存被调试应用程序的寄存器与内存状态. 由于对每条指令都加入了断点, 并在断点处保存程序的运行状态, 因此可以保证最高的历史调试精度. 但是, GDB 实现反向调试功能的代价就是数据记录将严重影响被调试程序的性能, 极端情况下会使该程序的执行速度下降数个数量级, 同时将导致记录数据占用较大的磁盘空间.

2 Eclipse 插件结构

“万物皆插件”是 Eclipse 平台的基本设计理念. Eclipse 的主要功能便是提供了一套明确定义了接口、类、方法的插件机制, 使各种定制化的插件能够无缝集成到该平台中, 方便了功能组件的研发[3].

插件 (Plug-in) 是该平台中最小的功能单元, 平台下的每项功能都由一个或多个插件完成. 每个插件的行为由具体的代码实现定义, 依赖项和提供的服务则由 MANIFEST.MF、plugin.xml 等插件清单文件定义. 这种架构保证了插件仅在需要的时候被 Eclipse 载入, 可以减少平台启动时间和内存消耗. 图 2 展示了 Eclipse 平台架构的组织形式, 可以看到该平台提供的绝大多数功能 (如 Java 开发、C/C++开发等) 均以外围插件 (如 JDT、CDT 等) 的形式予以实现.

在开发人员启动 Eclipse 之后, 平台运行时系统 (Platform Runtime) 将扫描默认插件的存放目录 plugins 和链接文件的存放目录 links, 在注册表中添加插件的相关信息, 并缓存各个插件的数据信息. 在开发人员使用某个插件提供的功能服务时, 该插件才会被平台运行时系统激活并调入内存, 最后在使用结束的某个合适时机被清理. 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[8].

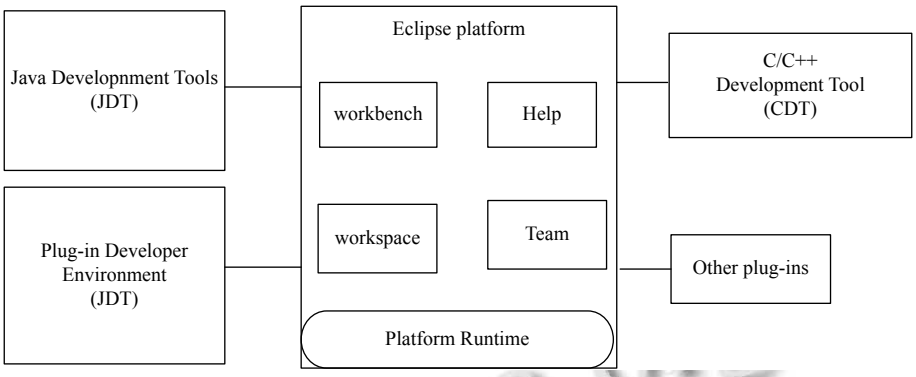

图 2 Eclipse 平台架构

3 智能跟踪调试方案设计

在国产平台上, 由于 GDB 反向调试的架构特定性[9], 以及单指令中断跟踪的方式[10], 导致其移植工作量巨大, 而且性能开销极大, 因此不适合在国产平台进行直接移植使用. 有鉴于此, 本文借鉴 Visual Studio 的 IntelliTrace 功能[11], 提出了基于事件的反向调试的技术方案.

在基于事件的反向调试技术中, 调试器不再维护一个连续执行、连续变化的程序执行历史. 亦即不再提供单步后退、单步前进等功能, 而仅在程序运行过程中对程序的运行状态在关键点进行快照, 提供对快照的管理以辅助开发人员定位程序错误, 开发人员无法对快照进行单步后退或者单步前进以查看单条指令或单行源码执行前后的程序状态, 而只能跳跃式地在离散的快照之间穿梭. 这些快照的管理包括快照的存储、快照的载入、在快照中前进与后退, 以及读取快照存储下来的程序状态, 以确定程序的故障. 在功能上, 这种方式类似于 IntelliTrace 的反向调试, 其总体架构如图 3 所示. 在图 3 中, 插件首先通过通信模块和本地或远程的调试服务器取得联系, 随后实时获取堆栈、系统调用、信号、事件等调试信息, 最后通过 UI 视图和 GDB、CDT 等后台工具插件以图形化的方式向开发人员显示调试信息.

3.1 事件跟踪模块

在 Linux 操作系统下, 应用程序与系统环境之间的交互主要是通过系统调用与信号, 它们均会以超出应用程序指令预期的方式改变程序的行为. 其中信号对应了大部分程序运行时的错误, 包括最常见的段错误. 因此, 记录下相应的事件, 即可对相当部分难以调试的软件故障进行回放调试, 从而协助快速定位与解决故障. 基于事件的反向调试技术支持的事件如表 1 所示.

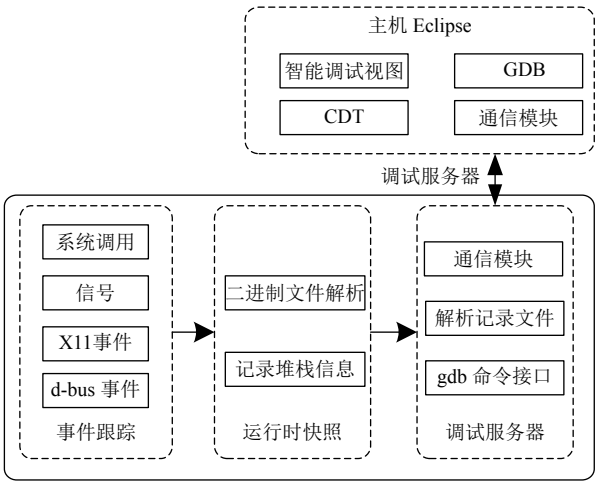

图 3 智能跟踪调试工具总体架构

表 1 支持的事件列表

分类	事件名称
系统调用	文件 (open、close、read、write 等)、网络 (socket、connect 等)、用户标识、进程线程等
信号	SIGINT、SIGQUIT、SIGSEGV 等
X11 事件	鼠标事件、键盘事件、焦点事件等
d-bus 事件	d-bus 方法调用、d-bus 信号接收等

监听程序的系统调用与信号, 可以通过 ptrace 的 PTRACE_SYSCALL 以及 PTRACE_GETSIGINFO 等功能实现. 一旦发生了对应的系统调用与信号, 调试器

即可获取对应的系统调用与信号的信息, 并同时获得被调试程序的状态, 包括寄存器、堆、各个线程的调用栈与线程局部存储等信息, 并将其保存下来. 针对 X11 事件的监控, 由于 XServer 从硬件那里接收到所有输入事件都会通知 XRecord, 因此当需要监控 X 事件时, 调试器只需把对应的代码挂到 XRecord 循环中. 只有系统一有输入事件产生, XRecord 接着就通过事件循环告诉调试器, 调试器再利用实时截获到的输入事件进行处理. 对于 d-bus 事件的监控, 由于所有的消息接口都来源于 libdbus, 调试器只要对 d-bus 收发函数通过 PRELOAD 的方式接管, 即可对被调试程序的 d-bus 事件进行跟踪调试.

3.2 运行时内存快照设计

在发生特定事件, 收到信号和发起系统调用的时候, 只有记录完整的堆栈信息才能对程序进行回溯. 考虑到性能开销, 本文设置了两种记录模式:

1) 转储模式: 每个事件点发生的时候自动暂停所有运行的线程, 记录所有线程的调用栈信息, 同时使用 minidump 方式转储指定的内存数据, 记录完成后自动恢复所有线程 (用户基本感受不到程序被暂停过). 由于每次转储都有进程切换, 并且记录的内存数据跟程序的复杂性和设置的转储粒度相关, 程序越复杂或转储粒度越细致, 记录的内存数据就越多, 会导致被调试程序运行变慢. 但是优点是记录了大量的内存数据, 在回放的时候可以查看内存数据 (具体跟转储粒度有关). 本文转储模式特点如表 2 所示.

2) 快速模式: 每个事件点只记录当前发生事件的线程的调用栈里面每个栈帧的函数的参数、函数内已知局部变量的值、errno 等全局变量、用户态所有寄存器值等, 不转储内存数据. 优点是记录过程没有线程切换, 速度快. 缺点是不能查看应用程序的全局变量的值和当前发生事件点时刻的其它线程的调用栈信息.

表 2 本文转储模式与 coredump 对比

	minidump	coredump
转储大小	可配置线程、线程栈、模块、全局变量等	可配置栈大小和内存类型
转储时机	发生 3.1 节所述的事件时	仅在发生致命错误时 (收到 SIGQUIT 等信号)
转储优势	设置灵活, 数据大小可控	设置不够灵活, 精度与性能无法折中

3.3 调试服务器设计

在转储阶段, 调试器跟踪被调试应用程序记录的跟踪文件是调试器自定义的格式, GDB 无法识别. 此外, 调试器还要保证记录阶段和回放阶段程序映射的内存地址空间是同一个地址范围. 原生的 GDB 无法支持该功能, 因此还需要在 GDB 基础上扩展.

本文依据 GDB Remote Serial Protocol 实现一个 RSP 服务器[12,13], 在 RSP 服务器里面启动记录的进程, 用户就可以用 GDB 连接到该服务器来进行回放调试. 该方法节省了开发调试器的时间, 同时也可以让熟悉 GDB 的用户快速熟悉反向调试的用法, 而且方便与集成开发环境里的插件进行集成.

3.4 可视化调试插件设计

可视化调试插件包含以下 4 个部分: 工程管理模块、远程通信模块、调试调度模块和后台调试器. 工程管理模块负责获取当前待调试工程的具体信息, 包括工具链类型、调试类型等. 如果工程采用的是交叉调试等模式时, 远程通信模块会被调用, 自动建立与目标机之间的通信连接, 并收发本机和目标机之间的通信报文. 通信发起模块对 Eclipse 平台和 CDT 插件提供的 API 进行改写, 根据工程管理模块提供的数据重新配置调试信息, 并调用后台调试器发起调试. 各模块的关系如图 4 所示.

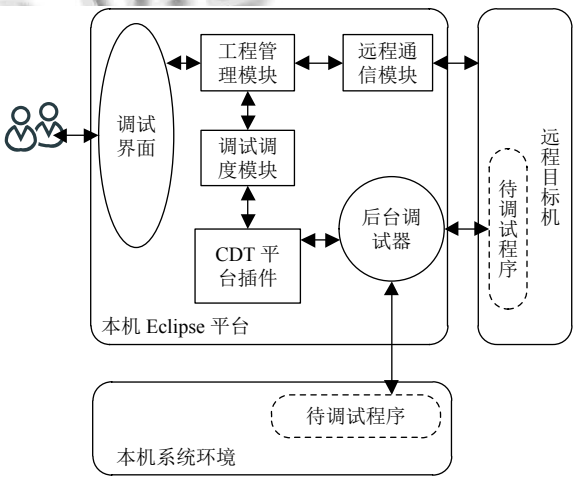

图 4 智能调试插件模块关系图

开发人员首先需要在界面上点击“调试”菜单项或命令按钮以激活智能调试插件并进入调试流程, 此时

插件会自动读取工程的配置信息, 并根据工程的类型(本地或远程) 选择不同的调试操作, 如图 5 所示.

对于本地调试, 插件会自动读取当前的软硬件平台信息, 并选择与之匹配的调试工具. 随后, 插件会再去寻找待调试工具, 同时搜索调试所需的动态库、相关联的工程, 并设置环境变量. 待所有准备工作就绪后, 启动调试. 整体流程如图 6 所示.

远程调试的流程与本地调试基本类似, 但在进行准备工作之前需要与目标机建立网络连接, 并将待调试的程序通过网络传送至指定位置, 随后再进行搜索动态库、配置环境变量等工作. 整体流程如图 7 所示.

通过上述步骤处理, 智能调试插件支持一键进入调试界面, 将手动配置等操作屏蔽掉, 可以显著提高软件的开发效率.

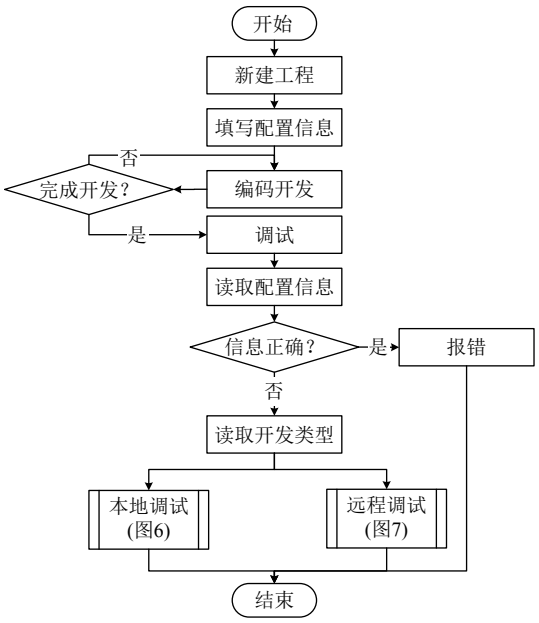

图 5 智能调试插件工作流程图

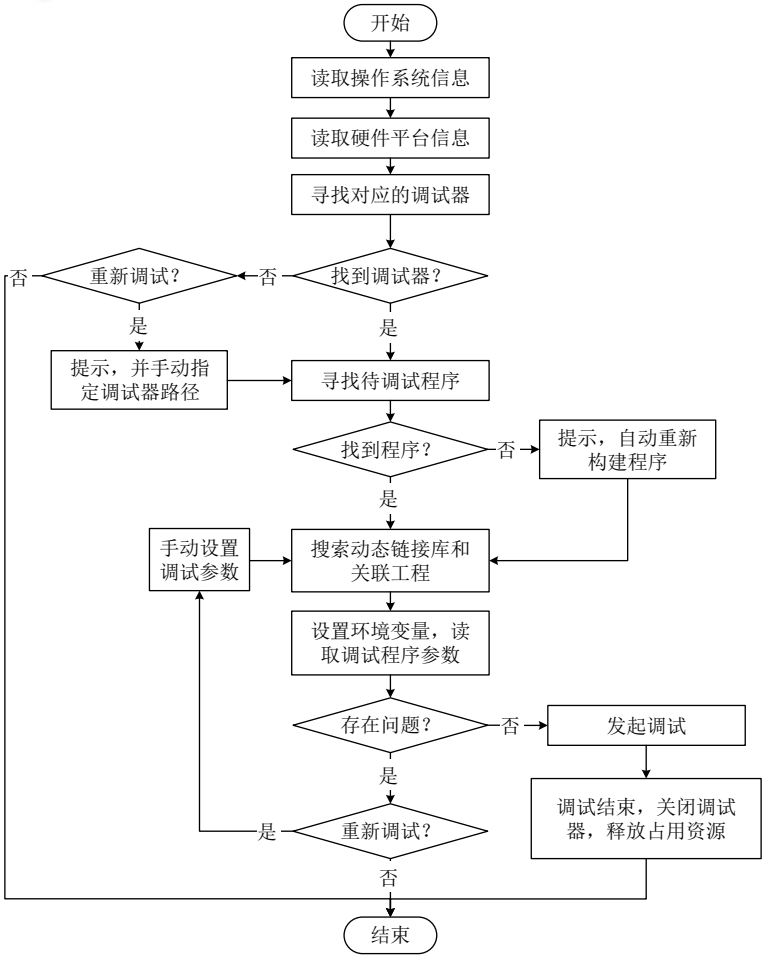

图 6 本地调试流程图

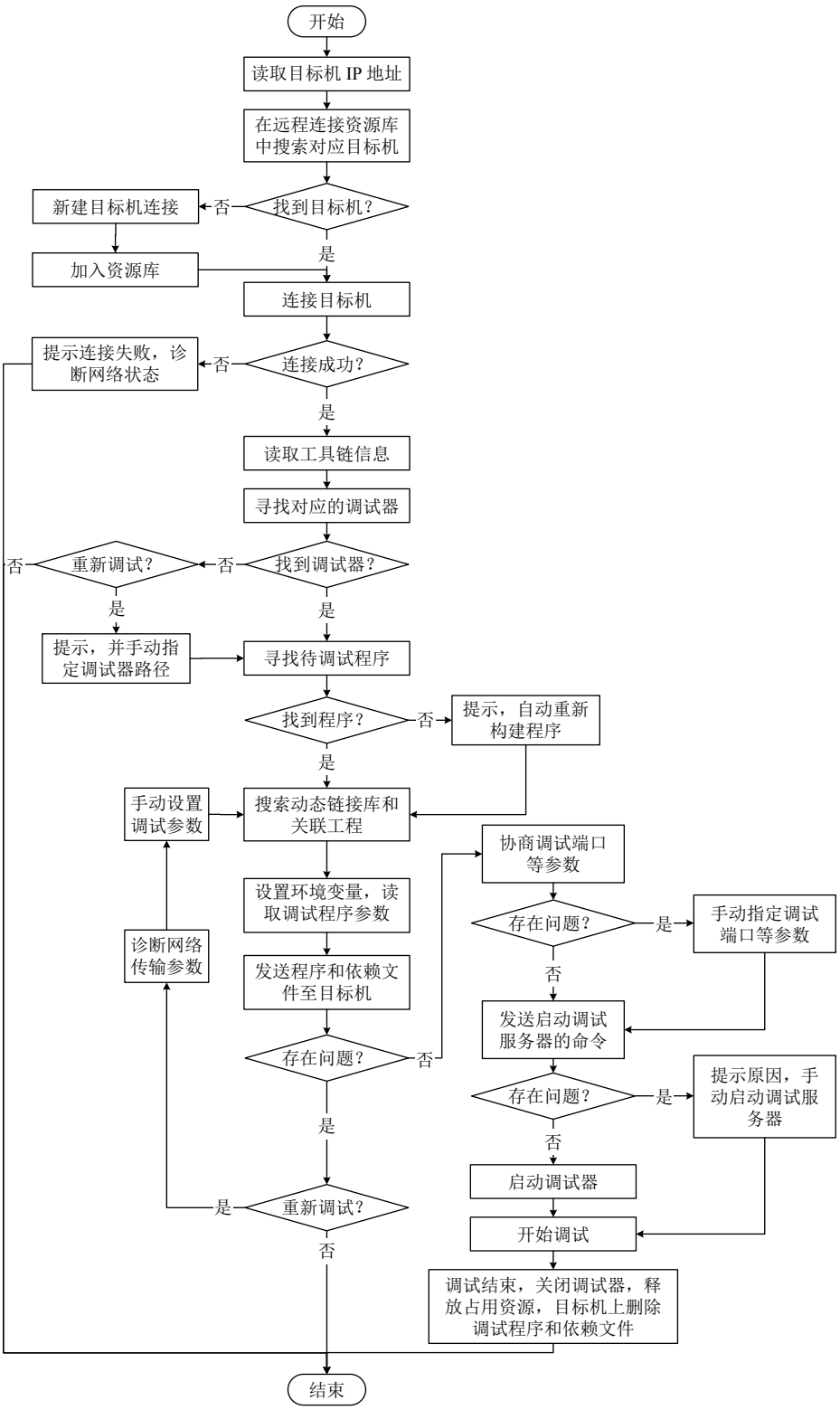

图 7 远程调试流程图

4 试验结果与系统实现

现通过 Eclipse CDT 原始插件和智能调试插件二者操作流程的对比以验证智能调试插件方案的实施效果, 测试环境配置见表 3. 实验结果表明本文设计的智

能调试插件具备方案可行、简便易用等特性.

使用传统的调试插件时, 首先需要在一个对话框中选择调试类型, 如“gdb/mi”、“gdbserver”和“remote gdb/mi”等. 对于“gdb/mi”这样的本地调试, 一般可以正常发起调试, 但如果应用程序用到的动态链接库不在/usr/lib、/usr/lib64、/usr/local/lib 等常见的搜索路径下时, 则需要手动在调试配置项中指定搜索路径. 倘若该工程依赖的多个动态链接库分布在不同位置, 则需要重复该工作若干次, 较为繁琐. 对于“gdbserver”、“remote gdb/mi”等远程调试, 除了本地调试的上述问题之外, 还需要先将生成的二进制程序通过网络发送到目标机, 再在目标机端以指定参数启动 GDB Server, 最后在 CDT 插件提供的界面中填入 IP 地址、端口号等信息并发起调试. 上述过程操作均需开发人员手动操作, 过程较为繁复. 同时, 传统的 CDT 插件对国产平台下的调试功能缺乏支持.

表 3 智能调试插件测试配置

配置名称	参数
主机操作系统	Windows XP、Windows 7、中标麒麟 5.0
主机硬件架构	X86、龙芯 3A3000
Eclipse 版本	4.4
CDT 版本	8.6.0
目标机操作系统	JARI-Works 3.2
目标机硬件架构	龙芯 3A3000
编程语言	C/C++

智能跟踪调试技术和与之对应的功能插件从流程上简化了用户的调试操作, 其为数不多的人工操作步骤中的绝大多数也可以通过简单的鼠标点击完成. 与传统的 CDT 调试相比, 该技术的优点见表 4.

表 4 调试技术对比

对比条目	CDT 调试	智能跟踪调试
操作复杂度	繁琐	简单
自动化程度	低	高
启动调试耗时	8~10 秒	5 秒以下
反向跟踪调试	仅支持 X86	龙芯平台 申威平台 飞腾平台 兼容 X86

在 Eclipse 平台下发起调试只需两步操作: (1) 右键点击工程; (2) 依次选择“Debug As”、“JARI-Works C/C++ Application”菜单项, 如果工程信息与调试插件存储的信息不匹配则会弹出提示窗口, 此时根据具体情形用鼠标点击对应的按钮即可. 对应的界面如图 8 所示, 其中包含了 A、B、C、D 等若干个视图. 其中, 视图 A 为项目管理视图, 通过工程名称的后缀可以看到各工程的类型 (应用工程、动态链接库工程、驱动工程等) 和工具链信息 (本地开发工具链、龙芯系列工具链、飞腾工具链等); 视图 B 为代码编辑视图, 显示当前调试的代码行, 通过 Alt+快捷键的形式可以控制调试方向, 如 F6 为正向单步调试的快捷键, 则 Alt+F6 为反向单步调试, 每个代码行改变的变量在视图 E 中高亮显示, 打印信息则在视图 D 中显示; 视图 C 为远程连接视图, 当前集成开发环境保存的远程连接均在此显示, 本地调试时此视图无用, 远程调试时调试插件会根据工程信息自动建立调试连接, 并在此视图中予以显示.

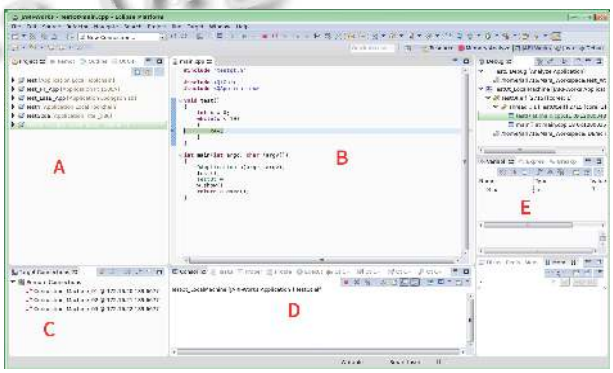

图 8 调试界面示意图

5 结语

本文阐述了反向调试的基本原理和 Eclipse 的插件体系架构, 针对国产平台性能较低的瓶颈, 设计了基于事件的反向调试方案, 通过对程序运行过程中关键点进行快照, 结合高性能的转储模块, 为国产平台提供易用的智能调试解决方案.

通过设计 Eclipse 下基于事件的反向调试插件, 可以方便地进行故障定位, 从而提高调试效率, 但还存在如下不足: (1) 转储时可设置粒度不足, 尤其针对性能较弱的国产处理器平台; (2) 调试视图设计较为简单, 未全面反应调试时程序状态; (3) 反向调试目前只支持 Alt+快捷键的方式操作, 在工具栏中缺少按钮控件, 不能全面覆盖开发人员的使用习惯; (4) 随着国产平台的推陈出新, 本技术覆盖的工具链仍需继续扩充. 这些问题将在后续工作中尝试改进.

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. [doi: 10.3969/j.issn.1672-9722.2012.

07.052]
2 田丹, 李运喜, 胡宁, 等. 基于 Eclipse 的嵌入式软件交叉调试. 现代电子技术, 2015, 38(6): 86–89. [doi: 10.3969/j.issn.1004-373X.2015.06.024]
3 李金萍, 程满玲. Eclipse 的应用发展. 电脑编程技巧与维护, 2016, (16): 31–32, 58. [doi: 10.3969/j.issn.1006-4052.2016.16.013]
4 黄子卿. 基于 OSGI 构建 Eclipse 高标准扩展组件. 电脑编程技巧与维护, 2018, (2): 53–59. [doi: 10.3969/j.issn.1006-4052.2018.02.015]
5 田丹. Eclipse 的 CDT 插件分析. 信息通信, 2018, (1): 156–157. [doi: 10.3969/j.issn.1673-1131.2018.01.070]
6 江山, 王维维, 蒋龙, 等. 一种快速定位 bug 的记录-回放调试系统. 计算机应用与软件, 2016, 33(10): 220–222, 237.
7 GDB and reverse debugging. http://www.gnu.org/software/gdb/news/reversible.html.
8 曹广旭, 孔平. 基于 Eclipse 平台的交叉调试方案. 计算机系统应用, 2015, 24(1): 216–220. [doi: 10.3969/j.issn.1003-3254.2015.01.041]
9 How to do bidirectional or reverse debugging of programs? https://stackoverflow.com/questions/522619/how-to-do-bidirectional-or-reverse-debugging-of-programs.
10 Cui WD, Peinado M. RETracer: Triaging crashes by reverse execution from partial memory dumps. Proceedings of the IEEE/ACM 38th International Conference on Software Engineering. Austin, TX, USA. 2016. 820-831.
11 IntelliTrace for visual studio enterprise (C#, Visual Basic, C++)https://docs.microsoft.com/en-us/visualstudio/debugger/intellitrace?view=vs-2017.
12 盛建忠, 王胜, 张庆文. GDB RSP 协议与 USB 通信在嵌入式调试系统中的应用. 电子与封装, 2013, 13(3): 43–48. [doi: 10.3969/j.issn.1681-1070.2013.03.011]
13 鲍庆国. 嵌入式设备固件分析的关键技术研究[硕士学位论文]. 北京: 北京工业大学, 2016.