Ruby Debugger for Ruby 1.9

Zhang Chiyuan
pluskid@gmail.com

March 31, 2008

Abstract

Most experienced programmers admit that debug is boring yet taking up most of our developing time. Using TDD (Test-driven development) can reduce the pain dramatically, but we still need to debug sometimes.

The goal of this project is to improve the debugging experience of the Ruby language. More specifically, the goal is either port the ruby-debug gem to Ruby 1.9 or implement a Rubinius like full speed debugger for Ruby 1.9.

Contents

1 The Project

Being very dynamic and flexible is the biggest feature of Ruby. It generally make our life better. However, it is really a double-edged sword. It is very hard to find out what's wrong in the code when something strange happened. For example, there are dozens of ways to define a method for an object, you never know when your class is accidentally monkey patched. So debugging in dynamic languages are especially important.

There are general two ways to debug in Ruby:

- Run the ruby interpreter inside gdb. This is usually used to debug the core Ruby code or some c extensions. gdb itself is a very powerful and extendable debugger with many third party tools. For example, attaching to a running process is easy in gdb, scripts can be written to enable evaluating Ruby code directly in the gdb console[1, 2].

- Run a debugger on the Ruby script level. Ruby comes with a debugger (debug.rb), but it's pretty minimal, yet slow on non-trivial applications[3]. A good alternative is the ruby-debug[4] gem, which is a faster implementation of the standard debug.rb using a native extension with a new hook Ruby C API. Moreover, it has many nice features like supporting multiple IDE frontends and remote debugging, etc.

The focus of this project is the latter: debugging on the Ruby level, or more specifically, Ruby 1.9.

Debugging in Ruby is a little bit weird because there's very few debugging support built in with the interpreter. There's even no primitives for accessing stack frames. One of the reason why `debug.rb` is so slow is creating and saving frame information along with a *binding* to access variables. Things are going to get even weirder since Ruby 1.9. There may be mixed levels of debugging.

Despite the lack of built in support, there're several decent debugger available:

- **ruby-debug**, as I mentioned earlier, this is a faster implementation of the standard debug.rb using a native extension with a new hook Ruby C API.

- **Cylon**, the debugger of Ruby In Steel[5], claimed to be the fastest and the most functional debugger available. However, this is a commercial one.

- **Rubinius debugger**, Rubinius[7] comes with a instruction replacement debugger. By this way, no overhead is added to normal statements.

The problem is: **none** of them works with Ruby 1.9. It is *necessary* for Ruby 1.9 to have a decent debugger. While try to re-invent the wheel to make a brand-new debugger from scratch totally violates the DRY (Don't Repeat Yourself) rule of Ruby community, I noticed that some progress[6] has been made towards porting ruby-debug to Ruby 1.9.

So I would like to propose this project: to port ruby-debug to Ruby 1.9 and investigate more ideas like instruction replacement.

Both the language and the implementation have changed dramatically since Ruby 1.9. A totally different VM is used. Furthermore, the internal structure of the new VM (YARV) seems to be still evolving and the access to that hasn't been officially exposed to the degree that would be needed for a straightforward port of ruby-debug. So porting to Ruby 1.9 isn't a easy task like simply replacing all `File.exists?` with `File.exist?`.

If everything goes well, I would also like to try out some new ideas. For example, the instruction replacement debugging. The Rubinius's VM is totally different from that of MRI or YARV. I *guess* it has plenty of support for debugging in the VM so that when instruction replacement is used to yield the debugger, the other normal code can run with full speed[8]. But we need to collect frame information in Ruby, that won't be *full speed* any longer.

However, this is still interesting. Even not *full speed*, avoiding the trace hook overhead is still a big step. This is not only about the speed of the debugger. Debugger will affect the original code, sometimes bugs will disappear as soon as you start the debugger (we generally call them Heisenbug). So making minimal change is sometimes required.

Besides the idea of instruction replacement, there are various cool things that come up in my mind when I write this proposal:

- **Dependency backtracking**. For example, you can track back where the variable bas been modified. CodeSurfer[9] can do this kind of things, though it is a static analyzing tool. Dynamic analyzing is also possible as is done in WHYLINE[10] for the Alice[11] language.

- **Backward debugging**. You might be familiar with the GDB command 'next' (or simply 'n'), but have you considered a 'prev' command? It is not impossible, ODB[12] implemented this for Java, so why not Ruby? In fact, there's already some sort of support in the ruby-debug's Emacs front-end.

- **Visualized debugging**, A concrete example is DDD (Data Display Debugger). For example, you can display the content and relationship of each item of a linked list from the debugger. It is very useful to debug complex structures. This should be considered in the front-end part.

  Another example is BlueJ[13] for Java. This is related but not necessarily for debugging. BlueJ is used for teaching: objects are visible shapes (UML, in fact) on the screen; calling a method is simply right clicking on that object and selecting the method.

- etc.

The 3-month time is very limited for me to try out all those ideas. However, I'm considering this as the first step into the community. The will definitely be further developments and maintenance after the Google Summer of Code, so on the other hand, the time is not limited.

2 Benefits to the Community

As I mentioned earlier, debugging is a very common task for programmers. A good debugger is especially important for programmers of a dynamic language like Ruby.

So almost all projects of the community can benefit from a better debugger. Ruby 1.9 is coming out, becoming stable and the next big thing! So a decent debugger is *needed*.

3 Milestones

With respect to the time schedule of Google Summer of Code, I would like to set two big milestones for this summer: porting ruby-debug to Ruby 1.9 and adding instruction replacement mechanism. I don't know much of how ruby-debug works right now and I was setting many *vague* milestones that can never be measured (like "understanding the ruby-debug code") in my original proposal. Fortunately, Rocky Bernstein gave me many suggestions and even an overview of what need to be done to finish the work. I set those milestones on top of his description:

- Porting ruby-debug to Ruby 1.9:
  - Remote debugging support in Ruby 1.9.
  - Write a `Kernel#binding_n` for Ruby 1.9. This is an important reason that makes ruby-debug faster than `debug.rb` in Ruby 1.8.
  - Invoking debugger from the outside, like rdebug.
  - Ideas on the front-ends. Something can be learned from user interface of the Cylon debugger of Ruby In Steel[5].
- Instruction replacement mechanism.
  - Saving the original instruction somewhere and restore later.
  - Disassembly of instructions at the current stopping point:
    - setting a breakpoint inside the current file/class/method.
    - and then in general (any file/class/method).
  - Adding the rest of ruby-debug on top of instruction replacement.

4 Timeline

Based on the milestones above and my own time schedule, I created the timeline before. In general, I don't have such a long 3-month summer vacation like those students study in the USA. I'll take my final examination during June 16 and July 2. However, I don't have any summer classes, and I'm not planning any traveling, so I can work full time for this project in my summer vacation after my final examination. I would also like to start before May 26 (students should start coding at this day on the GSoC timeline).

4.1 Remote debugging

April 19 ⇒ May 11

I should get familiar with the coding conventions and other stuffs before getting to write codes. I should port the remote debugging feature of ruby-debug to Ruby 1.9 at the of this period.

4.2 `Kernel#binding_n`

May 12 ⇒ May 28

`Kernel#binding_n` for Ruby 1.9 should be implemented during this period. This necessary for porting ruby-debug to Ruby 1.9.

4.3 rdebug

May 29 ⇒ June 11

Invoking the debugger from the outside should be supported at the end of this period.

4.4 Off for final examination

June 12 ⇒ July 2

My final examination starts on June 16. I'll start to prepare for it several days before that. I'll try to devote more time to my courses instead of this project in this period.

4.5 Porting

July 3 ⇒ July 16

Since the parts that are most specific the the internal Ruby implementation are ported to Ruby 1.9. I can start to port the other parts. This is a relative easy task if the code of ruby-debug is well abstracted.

Note that Google's mid-term evaluation of this project is on July 7. We should have most parts of ruby-debug ported to Ruby 1.9 before that day. However, at the end of this period (instead of July 7), we should have a working version of ruby-debug with all features of the Ruby 1.8 version supported.

4.6 Instruction replacement

July 17 ⇒ July 29

After porting ruby-debug to Ruby 1.9. We can have a look at instruction replacement mechanism now. During this period, I should implement a way to replace the instruction (or bytecode) at a particular position with the specific one for debugging, saving the original instruction and restore it when coming back from the debugger.

4.7 Breakpoint for current file/class/method

July 30 ⇒ August 7

Setting a breakpoint inside the current file/class/method should be implemented during this period.

4.8 Breakpoint for any file/class/method

August 8 ⇒ August 17

Setting a breakpoint in any file/class/method should be implemented during this period.

4.9 Google "pencil down" day

August 18 ⇒ August 21

The "pencil down" day on the Google Summer of Code timeline is August 18. We should check the status of the project and prepare materials for Google to evaluate the project. At this point, we should have a ruby-debug for Ruby 1.9 and the core parts of an instruction replacement style debugger.

4.10 Instruction replacement style debugger

August 22 ⇒ September 5

On top of the instruction replacement debugger core, adding the rest facility to make it a real debugger.

4.11 Front-end brainstorm and other

September 6 ⇒ ?

Brainstorm on improving the front-end support, other cool ideas, further developments and maintaining.

5 About Me

I've been using a general description of myself in my original proposal. Until Rocky Bernstein asked me some very specific questions, I realized that I need to provide more information on *why I am suitable for this project*.

5.1 Eligibility

I'm a junior student of Zhejiang University in China. I'm major in Computer Science and Technology. I've read the FAQ and I'm eligible to participate as a student in Google Summer of Code.

5.2 Passion

I love open source. I admire the idea of sharing source code with each other. I first got to know Linux about three years ago. Then I learned about Emacs, Mozilla, GNU and various open source projects. I've been very active in the local community of open source and Linux.

I have been using various open source programs. I love them because I can just patch the code to solve the problem myself. On the other hand, when I have problems, the community is always willing to help me. I've report bugs of various tools that I used. I myself also have some open source goodies. Here are some examples:

- **RMMSeg**[14]: A Ruby implementation of the maximum-matching Chinese word segmentation algorithm.
- **YASnippet**[15]: I call it Yet Another Snippet extension for Emacs because there are so many of them. Originally I borrowed the code of snippet.el and wrote smart-snippet[16]. But I finally design and write YASnippet from scratch. I'm quite confident about YASnippet. If you are a Emacser, do have a try! :D
- **kid-scheme**[17]: A scheme interpreter written in Ruby. This is not for real world production. In fact, I write it in less than two days. But I learned a lot from that, this is really a rite of passage for everyone who learns scheme.

I care very much about the community. Many users of my open source goodies describe me as very responsive.

I got to know Ruby about one year ago. I like it very much. But as I learned more about it, I know there are still many improvements needed (e.g. Ruby is usually considered as a very slow language). I would like to make things better through this project.

5.3 Ability

I'm quite familiar with Ruby programming. However, it seems this there won't be much Ruby code in this project. Instead, most of the code may be written in C. Fortunately, I'm familiar with the C programming language as well. I've learned C and C++ in our courses. Though I'd prefer writing code in a dynamic language, writing some medium sized program in C/C++ isn't impossible for me. I have written several course projects (with one of my roommate) of 4000 to 5000 lines of C/C++ code (and most of them were A+).

I've also tried to write an extension in C for my Ruby Chinese word segmentation implementation – RMMSeg[14]. Though finally I discard the result, I've learned a lot in the process:

- I learned how to write a C extension for Ruby.
- I experienced that *premature optimization is the root of all evil*[18].

- I learned how to debug Ruby in gdb.
- I understand the implementation of `Hash` in Ruby 1.8, found a bug[19] of `Fixnum` overflow in calculating hash code and sent a patch to the Ruby community. The patch was accepted with some modifications to make it better.

Finally, I'll mention my blog[20] (mainly Chinese) that I'm maintaining since last year. And my resume[21] is also available online if you would like to have a look. I always use the nickname 'pluskid' in mailinglist or IRC channels.

References

[1] Jamis Buck, *Inspecting a live Ruby process*, http://weblog.jamisbuck.org/2006/9/22/inspecting-a-live-ruby-process

[2] Mauricio Fernandez, *Inspecting a live Ruby process, easier if you cheat*, http://eigenclass.org/hiki.rb?ruby+live+process+introspection

[3] brian, *Ruby Debug Basics*, http://brian.maybeyoureinsane.net/blog/2007/05/07/ruby-debug-basics-screencast/

[4] *ruby-debug homepage*, http://rubyforge.org/projects/ruby-debug/

[5] *Ruby In Steel homepage*, http://www.sapphiresteel.com/

[6] *svn repository of debug-1.9*, http://rocky-hacks.rubyforge.org/svn/debug-1.9/trunk/

[7] *Rubinius homepage*, http://rubini.us/

[8] Werner Schuster, *Inside the full speed Rubinius debugger*, http://www.infoq.com/news/2008/01/rubinius-full-speed-debugger

[9] *CodeSurfer homepage*, http://www.grammatech.com/products/codesurfer/overview.html

[10] *WHYLINE homepage*, http://www.cs.cmu.edu/ NatProg/whyline.html

[11] *Alice homepage*, http://www.alice.org/

[12] *ODB homepage*, http://www.lambdacs.com/debugger/debugger.html

[13] *BlueJ homepage*, http://www.bluej.org/

[14] *RMMSeg homepage*, http://rmmseg.rubyforge.org/

[15] *YASnippet homepage*, http://yasnippet.googlecode.com/

[16] *smart-snippet homepage*, http://smart-snippet.googlecode.com/

[17] *kid-scheme homepage*, http://kid-scheme.googlecode.com/

[18] *Premature Optimization*, http://c2.com/cgi/wiki?PrematureOptimization

[19] Zhang Chiyuan, *Fixnum Overflow in Ruby's Hash Implementation*, http://pluskid.lifegoo.com/?p=286

[20] *My Blog*, http://pluskid.lifegoo.com/

[21] *My Resume*, http://pluskid.lifegoo.com/upload/personal/resume.pdf