

Sun Microsystems, Inc. - Trail: The Reflection API

Comments on:

Sun Microsystems, Inc (2006). Trail: The Reflection API. In *The Java™ Tutorials*. Available online at <http://java.sun.com/docs/books/tutorial/reflect/index.html>. (Last modified Thursday, 9 November 2006; visited Sunday, 4 March 2007.)

Note: Earlier versions of this trail indicated that the author was Dale Green.

Check Plus or Plus

From my understanding, it is relatively easy to support reflective programming in java because java applications are compiled to bytecode, before being interpreted to object code. In languages such as C that are compiled straight to machine code, how would one go about writing reflective code?

It's not clear that byte code is the only reason that reflection is available in Java (you also need to make that information available to the programmer). In fact, although the byte code is available, it's not really available (you can't introspect the implementation of a method, for example).

It's also not clear what reflection would mean in C, since you don't have objects, per se. But you are correct that you don't typically write reflective code (or anything similar) in C. My feeling is that it's more language design than anything.

Outside of a few obvious situations (such as a debugger or a class search program), it seems that reflections have little utility in solving real world problems that cannot be solved in a better way. For example, it is now possible to do some sort of higher order programming but it would be better to have things such as methods as values instead. What are the motivations behind reflections?

The debugger example and the class search examples are good ones. We'll go over a few others in class. I expect that the central motivation is the obvious one - If we have a tool that helps with A, B, and C, then rather than providing A, B, and C in an ad hoc manner, we provide one central solution that you can use for all three.

There is a Method class and sufficiently many ways to obtain methods that one might argue that Method is a kind of value. It's just hard to create new ones anonymously.

They are problems only if you value compile-time syntax and semantics checking :-). Some people appreciate the freedom that run-time checking provides.

I will admit that I have not explored the reflection API in sufficient depth to say how much of an efficiency issue it is. In some cases, it provides no real barrier. For example, if we want to load a new class at runtime and create a new object in the class, it may take fractionally more time to call `class.forName("FooBar").getConstructor(null).newInstance()` than it does to call `new FooBar()`, once the object is created, everything else will take the same time.

It is also the case that many cases in which we use reflection, we are willing to sacrifice a bit of efficiency in that part of the program.

Check

Does the reflection API have any type checking? Since types can be unknown until run time, it seems like the only way you would know if a type is incorrect is an exception being thrown during run time. This seems like it would make it harder to test a program because of this. If a program would need to make extensive use of the reflection API wouldn't it be easier to use an interpreted language (like say Lisp) instead of Java?

The reflection API primarily has run-time type checking, rather than compile-time type checking. And you are correct, the only way you really know about an error is an exception that you have to catch. Is it harder to program this way? Certainly people have programmed in languages like that (like, say, Lisp, to use your own words) for many years. Why would you use reflection instead of Lisp? Often, because you want to extend an existing Java program. Alternately, perhaps you are told that you must use Java, and you really think better in a dynamic model.

I can see lots of ways that the readings of Java provide an interesting and useful way to deal with objects when you don't know what they are. However, it doesn't seem as powerful as writing generics in C, because in order to do anything you have to first process an object. That said isn't it both easier (more readable) and more efficient (fewer lines of code) to do generics in C?

The purpose of reflection is not simply to do generics. (In fact, I'm not sure that it's that closely related to generics at all, but, well, you can make the connections you want.) If your only goal is a data structure and corresponding methods that work for a variety of types, then Java's generics are a much better solution than reflection.

As it also occurs to me... if you aren't given information about a particular object that's usually because the designer of the object doesn't want you to have to worry about the particulars of it. Setting up this series of methods/functions allows a user to break through that. Is it really a good thing? I thought Java was one of the more anal languages about not trusting the programmer (which I would expect to extend to the user as well).

The Reflection API is one point in which Java clearly trusts the programmer. However, it strikes me that it is designed in such a way that the average programmer would never use it, which puts us in the "ideal" situation in which average programmers have restrictions imposed on them, and advanced programmers do not :-).

If you are writing development tools, you may not know the class of an object until runtime. I am not sure I understand how I would create an object in such a case where the constructor has arguments. In the reading, the creation of objects with or without arguments is discussed, but I don't understand what one would do when the class of the object is known only at runtime? The reading suggests using the newInstance method.

Well, you would determine the types of the parameters. You would then create a vector of values of the correct types. (How? I'd use 0 for numbers, the empty string or "String" for strings, and do this stuff recursively for anything compound.) Finally, you'd call the constructor. Of course, it might not mean much in such cases, but ...

Typically, when you are intending to use reflection, you design the classes so that they have at least one zero-ary constructor.

My reflection pertains to the following section "The Reflection API > Manipulation Objects > Creating Objects > Using Constructors that have Arguments". It seems odd that the newInstance method of Constructor object does not support primitive types to create objects with a constructor that has arguments. I understand the need to create a generic program but I also think that it would be helpful to support at least the primitive data types. Hence, if a constructor has a lot of arguments of primitive data type we would not have to create objects for each of them. That adds to the efficiency of a programmer.

So, how would you implement this? Java only supports heterogenous arrays (that is, arrays that may have multiple types of values as contents) through arrays of objects. That means we must treat the primitive types as objects in calling the constructor. (Otherwise, you essentially need to provide an arbitrary number of pseudo-constructors, one for each combination of parameter types, and it is difficult to tell which are and are not available for each type.)

Note that many language designers criticize Java for having separate primitive types.

Note also that Java 5 and above support automatic boxing and unboxing. Automatic boxing means that the programmer won't need to explicitly create those objects.

This is the first time I have come across the term reflection in Java. From my understanding, reflection is a way to get/modify some information about objects, classes or methods at runtime. The reading warns that a programmer should not use reflection when other tools more natural to the Java programming language would suffice. Can you please give concrete examples of when reflection will be necessary in a program.

I'll go over examples of when reflection is useful in class.

Do other languages we have learned about also have tools similar to Java's reflection?

C does not. One might argue that the Lisp/Scheme model provides a different kind of reflection.

One thing I was wondering is the runtime security. Because the API offers the ability to manipulate the objects, it sounds that it has potential to violate the idea of encapsulation.

You are correct that the reflection API seems to violate encapsulation. I expect that the designers felt that Java would benefit from a type of back door for certain situations, and hoped that people would not abuse it.

Check Minus or Minus Responses

Generally, The Reflection API is a great idea. It makes programmers work much easier in terms of coding and debugging. Specifically, I feel that because of the help from Eclipse (an application of the reflection API), I no longer need to focus on very low level programming techniques, instead I can pay whole attention to the problem.

This is a bit vague. What low-level techniques does it free you from?

From the aspect of using the API to build debuggers, class browsers, and even some GUIs, the reflection API provides an elegant package of classes. Because I have no experience in this kind of programming, I have no criticisms.

One thing we hope you develop at Grinnell is an imagination. What does it mean to have to write a debugger or class browser? How would you write one without the reflection API? Does this technique seem appropriate, or can you envision another technique that might be even more appropriate?