Metaprogramming - self explanatory code - tutorials, articles, books [closed]

Asked 10 years, 5 months ago Active 3 years, 7 months ago Viewed 5k times

22

19

Closed. This question does not meet Stack Overflow guidelines. It is not currently accepting answers.

Want to improve this question? Update the question so it's on-topic for Stack Overflow.
Closed 3 years ago.

Improve this question

I am looking into improving my programming skils (actually I try to do my best to suck less each year, as our Jeff Atwood put it), so I was thinking into reading stuff about metaprogramming and self explanatory code.

I am looking for something like an idiot's guide to this (free books for download, online resources). Also I want more than your average wiki page and also something language agnostic or preferably with Java examples.

Do you know of such resources that will allow to efficiently put all of it into practice (I know experience has a lot to say in all of this but i kind of want to build experience avoiding the flow bad decisions - experience - good decisions)?

EDIT:

Something of the likes of this example from the Pragmatic Programmer:

...implement a mini-language to control a simple drawing package... The language consists of single-letter commands. Some commands are followed by a single number. For example, the following input would draw a rectangle:

```
P 2 # select pen 2
D # pen down
W 2 # draw west 2cm
N 1 # then north 1
E 2 # then east 2
S 1 # then back south
U # pen up
```

Thank you!

java metaprogramming coding-style

edited Apr 2 '10 at 7:54
Darius Bacon
14.1k 5 48 52

asked Apr 2 '10 at 7:40
elena
221 2 4

Metaprogramming — like C++ templates (which is far from self-explanatory code), or something else? – kennytm Apr 2 '10 at 7:44

@KennyTM: see my edited quetion. Thanks! – elena Apr 2 '10 at 7:49

@stakx: "In what ways would the given example (a simple drawing language) be considered as metaprogramming". It is an interpreter which is metaprogramming by definition. – J D Apr 12 '17 at 15:53

1 @JonHarrop: I may have misunderstood the purpose of the example code back then. I was looking at the actual

understanding why I was thinking of preprocessors. I delete my above comments, they were pretty confusing. – stakx - no longer contributing Apr 12 '17 at 19:03

7 Answers

Active Oldest Votes

39

Welcome to the wonderful world of meta-programming :) Meta programming relates actually to many things. I will try to list what comes to my mind:

- **Macro**. The ability to extend the syntax and semantics of a programming language was explored first under the terminology *macro*. Several languages have constructions which resemble to macro, but the piece of choice is of course **Lisp**. If you are interested in meta-programming, understanding Lisp and the macro system (and the homoiconic nature of the languge where code and data have the same representation) is definitively a must. If you want a Lisp dialect that runs on the JVM, go for **Clojure**. A few resources:
  - Clojure mini language
  - Beating the Averages (why Lisp is a secret weapon)

  There is otherwise plenty of resource about Lisp.
- **DSL**. The ability to extend one language syntax and semantics is now rebranded under the term "DSL". The easiest way to create a DSL is with the interpreter pattern. Then come internal DSL with fluent interface and external DSL (as per Fowler's terminology). Here is a nice video I watched recently:
  - DSL: what, why, how

  The other answers already pointed to resources in this area.
- **Reflection**. Meta-programming is also inseparable form reflection. The ability to reflect on the program structure at run-time is immensely powerful. It's important then to understand what introspection, intercession and reification are. IMHO, reflection permits two broad categories of things: 1. the manipulation of data whose structure is not known at compile time (the structure of the data is then provided at run-time and the program stills works reflectively). 2. powerful programming patterns such as dynamic proxy, factories, etc. **Smalltalk** is the piece of choice to explore reflection, where *everything* is reflective. But I think **Ruby** is also a good candidate for that, with a community that leverage meta programming (but I don't know much about Ruby myself).
  - Smalltalk: a reflective language
  - Magritte: a meta driven approach to empower developpers and end-users

  There is also a rich literature on reflection.
- **Annotations**. Annotations could be seen as a subset of the reflective capabilities of a language, but I think it deserves its own category. I already answered once what annotations are and how they can be used. Annotations are meta-data that can be processed at compile-time or at run-time. **Java** has good support for it with the annotation processor tool, the Pluggable Annotation Processing API, and the mirror API.
- **Byte-code or AST transformation**. This can be done at compile-time or at run-time. This is somehow are low-level approach but can also be considered a form of meta-programming (In a sense, it's the same as macro for non-homoiconic language.)
  - DSL with Groovy (There is an example at the end that shows how you can plug your own AST transformation with annotations).

Conclusion: Meta-programming is the ability for a program to reason about itself or to modify itself. Just like meta stack overflow is the place to ask question about stack overflow itself. Meta-programming is not one specific technique, but rather an ensemble of concepts and techniques.

Several things fall under the umbrella of meta-programming. From your question, you seem more interested in the macro/DSL part. But everything is ultimately related, so the other aspects of meta-programming are also definitively worth looking at.

edited May 23 '17 at 12:10
Community ♦
1 1

answered Apr 2 '10 at 12:14
ewernli
35.6k 3 79 116

4

I've mentioned C++ template metaprogramming in my comment above. Let me therefore provide a brief example using C++ template meta-programming. I'm aware that you tagged your question with `java`, yet this may be insightful. I hope you will be able to understand the C++ code.

Demonstration by example:

Consider the following recursive function, which generates the Fibonacci series (0, 1, 1, 2, 3, 5, 8, 13, ...):

```
unsigned int fib(unsigned int n)
{
    return n >= 2 ? fib(n-2) + fib(n-1) : n;
}
```

To get an item from the Fibonacci series, you call this function -- e.g. `fib(5)` --, and it will compute the value and return it to you. Nothing special so far.

But now, in C++ you can re-write this code using templates (somewhat similar to generics in Java) so that the Fibonacci series won't be generated at *run-time*, but during *compile-time*:

```
// fib(n) := fib(n-2) + fib(n-1)
template <unsigned int n>
struct fib                     // <-- this is the generic version fib<n>
{
    static const unsigned int value = fib<n-2>::value + fib<n-1>::value;
};

// fib(0) := 0
template <>
struct fib<0>                  // <-- this overrides the generic fib<n> for n = 0
{
    static const unsigned int value = 0;
};

// fib(1) := 1
template <>
struct fib<1>                  // <-- this overrides the generic fib<n> for n = 1
{
    static const unsigned int value = 1;
};
```

To get an item from the Fibonacci series using this template, simply retrieve the constant value -- e.g. `fib<5>::value`.

Conclusion ("What does this have to do with meta-programming?"):

In the template example, **it is the C++ compiler that generates the Fibonacci series at compile-time, not your program while it runs.** (This is obvious from the fact that in the first example, you call a function, while in the template example, you retrieve a constant value.) You get your Fibonacci numbers without writing a function that computes them! Instead of programming that function, you have programmed the compiler to do something for you that it wasn't explicitly designed for... which is quite remarkable.

This is therefore one form of meta-programming:

> *Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data, **or that do part of the work at compile time that would otherwise be done at runtime**.*
>
> -- Definition from the Wikipedia article on metaprogramming, emphasis added by me.

(Note also the side-effects in the above template example: As you make the compiler pre-compute your Fibonacci numbers, they need to be stored somewhere. The size of your program's binary will increase proportionally to the highest `n` that's used in expressions containing the term `fib<n>::value`. On the upside, you save computation time at run-time.)

edited Apr 2 '10 at 12:00 answered Apr 2 '10 at 9:42

stakx - no longer contributing
74.4k 17 147 239

3

From your example, it seems you are talking about domain specific languages (DSLs), specifically, Internal DSLs.

Here is a large list of books about DSL in general (about DSLs like SQL).

Martin Fowler has a book that is a work in progress and is currently online.

Ayende wrote a book about DSLs in boo.

Update: (following comments)

Metaprogramming is about creating programs that control other programs (or their data), sometimes using a DSL. In this respect, batch files and shell scripts can be considered to be metaprogramming as they invoke and control other programs.

The example you have shows a DSL that may be used by a metaprogram to control a painting program.

edited Apr 2 '10 at 8:22 answered Apr 2 '10 at 7:55

Oded
450k 84 817 959

Is there a difference about DSL and metaprogramming? I mean are they two different beasts or one derived from the other? (new concepts for me :)) – elena Apr 2 '10 at 8:10

@elena - Metaprogramming is about programs that manipulate other programs or their data. Sometimes a metaprogram uses a DSL. – Oded Apr 2 '10 at 8:19

1

Tcl started out as a way of making domain-specific languages that didn't suck as they grew in complexity to the point where they needed to get generic programming capabilities. Moreover, it remains very easy to add in your own commands precisely because that's still an important use-case for the language.

If you're wanting an implementation integrated with Java, Jacl is an implementation *of* Tcl *in* Java which provides scriptability focussed towards DSLs and also access to access any Java object.

(Metaprogramming is writing programs that write programs. Some languages do it far more than others. To pick up on a few specific cases, Lisp is the classic example of a language that does a lot of metaprogramming; C++ tends to relegate it to templates rather that permitting it at runtime; scripting languages all tend to find metaprogramming easier because their implementations are written to be more flexible that way, though that's just a matter of degree..)

answered Apr 2 '10 at 8:12

Donal Fellows

0

Well, in the Java ecosystem, i think the simplest way to implement a mini-language is to use scripting languages, like Groovy or Ruby (yes, i know, Ruby is not a native citizen of the java ecosystem). Both offer rather good DSL specification mechanism, that will allow you to do that with far more simplicity than the Java language would :

- Writing DSL in Groovy
- Creating Ruby DSL

There are however pure Java laternatives, but I think they'll be a little harder to implement.

answered Apr 2 '10 at 7:57
Riduidel
20.7k 11 70 154

JRuby is, however, a native citizen of the Java ecosystem. – JUST MY correct OPINION Apr 2 '10 at 8:12

Yeah, of course, but Ruby per se was created as an independant language, with its own interpreter/execution environment, providing more integration ot native resources than JRuby is able to. – Riduidel Apr 2 '10 at 9:19

0

You can have a look at the eclipse modeling project, they've got support for meta-models.

answered Apr 5 '10 at 13:47
LB40
10.8k 15 65 103

0

There's a course on Pluralsight about Metaprogramming which might be a good entry point https://app.pluralsight.com/library/courses/understanding-metaprogramming/table-of-contents

answered Jan 11 '17 at 16:44
Oscar Fraxedas
3,531 2 22 29