

Dynamic type languages versus static type languages

Asked 16 years, 3 months ago Modified 4 years, 3 months ago

Viewed 131k times



What are the advantages and limitations of dynamic type languages compared to static type languages?

208



See also: [whats with the love of dynamic languages](#) (a far more argumentative thread...)



programming-languages

dynamic-languages

type-systems



Share

Improve this question

Follow

edited May 23, 2017 at 12:18



Community Bot

1 ● 1

asked Sep 24, 2008 at 4:05



CVS

3,061 ● 8 ● 28 ● 22

10 This question is too subjective. – [Jason Dagit](#) Sep 24, 2008 at 4:07

7 I wouldn't call it subjective, but flame bait. But there are some objective facts regarding it. – [Vinko Vrsalovic](#) Sep 24, 2008 at 4:16

- 2 Agreed: too subjective. It's interesting to compare and contrast the two approaches, but it teeters dangerously on the brink of forum apocalypse. – [Daniel Spiewak](#) Sep 24, 2008 at 5:23
-
- 17 Dynamic languages are great for rapid development of demo/throwaway applications because if you make a typo who cares, the webpage still loads you might just have a couple of data elements wrong here or there. I cannot imagine any other situation where the ability to mistype your variables without getting a compiler error is seen as an "advantage". – [Alex R](#) Mar 20, 2010 at 4:32
-
- 4 Such an error would typically bring JavaScript to a screeching halt, which I consider a very good thing. At the very least it would throw errors which I also find valuable. For some reason it's always a guy from a static typing paradigm that wants to bury his javascript errors with empty try/catch statements. It's been something of a phenomenon in my experience. What is that? Regardless, it's not like we don't get feedback when we run our code. – [Erik Reppen](#) Feb 14, 2013 at 14:57
-

9 Answers

Sorted by:

Highest score (default)



141



The ability of the interpreter to deduce type and type conversions makes development time faster, but it also can provoke runtime failures which you just cannot get in a statically typed language where you catch them at compile time. But which one's better (or even if that's always true) is hotly discussed in the community these days (and since a long time).



A good take on the issue is from [Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages](#) by Erik Meijer and Peter Drayton at Microsoft:

Advocates of static typing argue that the advantages of static typing include earlier detection of programming mistakes (e.g. preventing adding an integer to a boolean), better documentation in the form of type signatures (e.g. incorporating number and types of arguments when resolving names), more opportunities for compiler optimizations (e.g. replacing virtual calls by direct calls when the exact type of the receiver is known statically), increased runtime efficiency (e.g. not all values need to carry a dynamic type), and a better design time developer experience (e.g. knowing the type of the receiver, the IDE can present a drop-down menu of all applicable members). Static typing fanatics try to make us believe that “well-typed programs cannot go wrong”. While this certainly sounds impressive, it is a rather vacuous statement. Static type checking is a compile-time abstraction of the runtime behavior of your program, and hence it is necessarily only partially sound and incomplete. This means that programs can still go wrong because of properties that are not tracked by the type-checker, and that there are programs that while

they cannot go wrong cannot be type-checked. The impulse for making static typing less partial and more complete causes type systems to become overly complicated and exotic as witnessed by concepts such as “phantom types” [11] and “wobbly types” [10]. This is like trying to run a marathon with a ball and chain tied to your leg and triumphantly shouting that you nearly made it even though you bailed out after the first mile.

Advocates of dynamically typed languages argue that static typing is too rigid, and that the softness of dynamically languages makes them ideally suited for prototyping systems with changing or unknown requirements, or that interact with other systems that change unpredictably (data and application integration). Of course, dynamically typed languages are indispensable for dealing with truly dynamic program behavior such as method interception, dynamic loading, mobile code, runtime reflection, etc. In the mother of all papers on scripting [16], John Ousterhout argues that statically typed systems programming languages make code less reusable, more verbose, not more safe, and less expressive than dynamically typed scripting languages. This argument is parroted literally by many proponents of dynamically typed scripting languages. We argue that this is a fallacy and falls into the same category as arguing that the

essence of declarative programming is eliminating assignment. Or as John Hughes says [8], it is a logical impossibility to make a language more powerful by omitting features. Defending the fact that delaying all type-checking to runtime is a good thing, is playing ostrich tactics with the fact that errors should be caught as early in the development process as possible.

Share Improve this answer

Follow

edited Jan 4, 2018 at 15:18



Bacon Bits

32k ● 6 ● 61 ● 73

answered Sep 24, 2008 at 4:10



Vinko Vrsalovic

340k ● 55 ● 340 ● 373

-
- 37 "method interception, dynamic loading, mobile code, runtime reflection" can all be done in Java, just for the record.
– [Will Hartung](#) Sep 24, 2008 at 6:11
-
- 13 Phantom types are not "overly complicated". – [J D](#) Apr 19, 2010 at 18:01
-
- 12 The link to the Meijer paper is broken as of 5/16/2010.
– [jchadhowell](#) May 17, 2010 at 3:38
-
- 5 @jchadhowell, you can find that here
research.microsoft.com/en-us/um/people/emeijer/Papers/...
– [Srinivas Reddy Thatiparthi](#) Mar 14, 2011 at 7:29
-
- 4 @VinkoVrsalovic Static languages with type inference and polymorphism are quite good for rapid prototyping. They offer the same comfort as dynamic language and the safety of static languages. – [Edgar Klerks](#) Aug 1, 2013 at 9:03
-



119



Static type systems seek to eliminate certain errors statically, inspecting the program without running it and attempting to prove soundness in certain respects. Some type systems are able to catch more errors than others. For example, C# can eliminate null pointer exceptions when used properly, whereas Java has no such power. Twelf has a type system which actually [guarantees that proofs will terminate](#), "solving" the [halting problem](#).

However, no type system is perfect. In order to eliminate a particular class of errors, they must also reject certain perfectly valid programs which violate the rules. This is why Twelf doesn't really solve the halting problem, it just avoids it by throwing out a large number of perfectly valid

proofs which happen to terminate in odd ways. Likewise, Java's type system rejects Clojure's `PersistentVector` implementation due to its use of heterogeneous arrays. It works at runtime, but the type system cannot verify it.

For that reason, most type systems provide "escapes", ways to override the static checker. For most languages, these take the form of casting, though some (like C# and Haskell) have entire modes which are marked as "unsafe".

Subjectively, I like static typing. Implemented properly (hint: *not* Java), a static type system can be a huge help in weeding out errors before they crash the production system. Dynamically typed languages tend to require more unit testing, which is tedious at the best of times. Also, statically typed languages can have certain features which are either impossible or unsafe in dynamic type systems ([implicit conversions](#) spring to mind). It's all a question of requirements and subjective taste. I would no more build the next Eclipse in Ruby than I would attempt to write a backup script in Assembly or patch a kernel using Java.

Oh, and people who say that "x typing is 10 times more productive than y typing" are simply blowing smoke. Dynamic typing may "feel" faster in many cases, but it loses ground once you actually try to make your fancy application *run*. Likewise, static typing may seem like it's the perfect safety net, but one look at some of the more complicated generic type definitions in Java sends most

developers scurrying for eye blinders. Even with type systems and productivity, there is no silver bullet.

Final note: don't worry about performance when comparing static with dynamic typing. Modern JITs like V8 and TraceMonkey are coming dangerously-close to static language performance. Also, the fact that Java actually compiles down to an inherently dynamic intermediate language should be a hint that for most cases, dynamic typing isn't the huge performance-killer that some people make it out to be.

Share Improve this answer

Follow

edited Mar 29, 2015 at 22:08



[Benjamin Hodgson](#)

44.5k ● 17 ● 112 ● 166

answered Sep 24, 2008 at 5:19



[Daniel Spiewak](#)

55.1k ● 14 ● 111 ● 120

5 About the performance. In common cases it won't make that much of a difference, but in high tension maths and such, there is a real difference. Test's have proven a call of a funcion, in the case of ipy vs C#, differs with a thousand cycles. Just because the former has to be sure a method exists. – [Dykan](#) Sep 2, 2009 at 7:15

29 can u elaborate on the point "C# can eliminate null pointer exceptions when used properly, whereas Java has no such power." ? An example or citation would be highly appreciated. – [Srinivas Reddy Thatiparthi](#) Mar 14, 2011 at 7:08

13 "some of the more complicated generic type definitions in Java sends most developers scurrying for eye blinders" - if

this is your worst case example, you obviously haven't used C++ ;-)) – [bacar](#) Jan 30, 2012 at 1:45

- 3 " Also, the fact that Java actually compiles down to an inherently dynamic intermediate language should be a hint that for most cases, dynamic typing isn't the huge performance-killer that some people make it out to be." -- when your example of a language with "good performance" is Java, you might want to reconsider.
– [Yakk - Adam Nevraumont](#) Jun 3, 2014 at 22:45
-

"Java compiles down to an inherently dynamic intermediate" - that misses the point. The static checks have been performed in advance and therefore no additional run time checks covering up for those are required as the compiler chooses instructions like `dadd` because he knows in advance that the operands are `double` s. – [Michael Beer](#) May 30, 2018 at 1:17



44

Well, both are very, very very very misunderstood and also two completely different things. **that aren't mutually exclusive.**



Static types are a restriction of the grammar of the language. Statically typed languages strictly could be said to not be context free. The simple truth is that it becomes inconvenient to express a language sanely in context free grammars that doesn't treat all its data simply as bit vectors. Static type systems are part of the grammar of the language if any, they simply restrict it more than a context free grammar could, grammatical checks thus happen in two passes over the source really. Static types correspond to the mathematical notion of type theory, type theory in mathematics simply restricts the legality of

some expressions. Like, I can't say `3 + [4, 7]` in maths, this is because of the type theory of it.

Static types are thus not a way to 'prevent errors' from a theoretical perspective, they are a limitation of the grammar. Indeed, provided that `+`, `3` and intervals have the usual set theoretical definitions, if we remove the type system `3 + [4, 7]` has a pretty well defined result that's a set. 'runtime type errors' theoretically do not exist, the type system's practical use is to prevent operations that *to human beings* would make no sense. Operations are still just the shifting and manipulation of bits of course.

The catch to this is that a type system can't decide if such operations are going to occur or not if it would be allowed to run. As in, exactly partition the set of all possible programs in those that are going to have a 'type error', and those that aren't. It can do only two things:

- 1: prove that type errors are going to occur in a program
- 2: prove that they aren't going to occur in a program

This might seem like I'm contradicting myself. But what a C or Java type checker does is it rejects a program as 'ungrammatical', or as it calls it 'type error' if it *can't* succeed at 2. It can't prove they aren't going to occur, that doesn't mean that they aren't going to occur, it just means it can't prove it. It might very well be that a program which will not have a type error is rejected simply because it can't be proven by the compiler. A simple example being `if(1) a = 3; else a = "string";`, surely since it's always true, the else-branch will never be executed in the

program, and no type error shall occur. But it can't prove these cases in a general way, so it's rejected. This is the major weakness of a lot of statically typed languages, in protecting you against yourself, you're necessarily also protected in cases you don't need it.

But, contrary to popular believe, there are also statically typed languages that work by principle 1. They simply reject all programs of which they can prove it's going to cause a type error, and pass all programs of which they can't. So it's possible they allow programs which have type errors in them, a good example being Typed Racket, it's hybrid between dynamic and static typing. And some would argue that you get the best of both worlds in this system.

Another advantage of static typing is that types are known at compile time, and thus the compiler can use this. If we in Java do `"string" + "string"` or `3 + 3`, both `+` tokens in text in the end represent a completely different operation and datum, the compiler knows which to choose from the types alone.

Now, I'm going to make a very controversial statement here but bear with me: **'dynamic typing' does not exist.**

Sounds very controversial, but it's true, dynamically typed languages are from a theoretical perspective *untyped*. They are just statically typed languages with only one type. Or simply put, they are languages that are indeed grammatically generated by a context free grammar in practice.

Why don't they have types? Because every operation is defined and allowed on every operand, what's a 'runtime type error' exactly? It's from a theoretical example purely a *side-effect*. If doing `print("string")` which prints a string is an operation, then so is `length(3)`, the former has the side effect of writing `string` to the standard output, the latter simply `error: function 'length' expects array as argument.`, that's it. There is from a theoretical perspective no such thing as a dynamically typed language. They are *untyped*

All right, the obvious advantage of 'dynamically typed' language is expressive power, a type system is nothing but a limitation of expressive power. And in general, languages with a type system indeed would have a defined result for all those operations that are not allowed if the type system was just ignored, the results would just not make sense to humans. Many languages lose their Turing completeness after applying a type system.

The obvious disadvantage is the fact that operations can occur which would produce results which are nonsensical to humans. To guard against this, dynamically typed languages typically redefine those operations, rather than producing that nonsensical result they redefine it to having the side effect of writing out an error, and possibly halting the program altogether. This is not an 'error' at all, in fact, the language specification usually implies this, this is as much behaviour of the language as printing a string from a theoretical perspective. Type systems thus force the programmer to reason about the flow of the code to

make sure that this doesn't happen. Or indeed, reason so that it *does* happen can also be handy in some points for debugging, showing that it's not an 'error' at all but a well defined property of the language. In effect, the single remnant of 'dynamic typing' that most languages have is guarding against a division by zero. This is what dynamic typing is, there are no types, there are no more types than that zero is a different type than all the other numbers. What people call a 'type' is just another property of a datum, like the length of an array, or the first character of a string. And many dynamically typed languages also allow you to write out things like `"error: the first character of this string should be a 'z'".`

Another thing is that dynamically typed languages have the type available at runtime and usually can check it and deal with it and decide from it. Of course, in theory it's no different than accessing the first char of an array and seeing what it is. In fact, you can make your own dynamic C, just use only one type like long long int and use the first 8 bits of it to store your 'type' in and write functions accordingly that check for it and perform float or integer addition. You have a statically typed language with one type, or a dynamic language.

In practise this all shows, statically typed languages are generally used in the context of writing commercial software, whereas dynamically typed languages tend to be used in the context of solving some problems and automating some tasks. Writing code in statically typed languages simply takes long and is cumbersome

because you can't do things which you know are going to turn out okay but the type system still protects you against yourself for errors you don't make. Many coders don't even realize that they do this because it's in their system but when you code in static languages, you often work around the fact that the type system won't let you do things that can't go wrong, because it can't prove it won't go wrong.

As I noted, 'statically typed' in general means case 2, guilty until proven innocent. But some languages, which do not derive their type system from type theory at all use rule 1: Innocent until proven guilty, which might be the ideal hybrid. So, maybe Typed Racket is for you.

Also, well, for a more absurd and extreme example, I'm currently implementing a language where 'types' are truly the first character of an array, they are data, data of the 'type', 'type', which is itself a type and datum, the only datum which has itself as a type. Types are not finite or bounded statically but new types may be generated based on runtime information.

[Share](#) [Improve this answer](#)

[Follow](#)

[edited Aug 26, 2020 at 22:33](#)



[Derek Mahar](#)

28.3k ● 46 ● 127 ● 181

[answered Jul 6, 2010 at 11:54](#)



[Zorf](#)

6,444 ● 2 ● 31 ● 26

1 "Many languages lose their Turing completeness after applying a type system." does not apply to usual programming languages, right? from what I read, regular languages are not turing-complete – [Răzvan Flavius Panda](#) Oct 17, 2012 at 12:09

1 @RăzvanPanda: Lajla was probably referring to variations on the [Typed lambda calculus](#) or some of the programming languages they use on theorem provers. Many of those can only express programs that are guaranteed to halt and therefore are not Turing complete. Practical functional programming languages based on these type systems go around this limitation by extending the core calculus with recursive types. – [hugomg](#) Apr 4, 2013 at 23:06 ✎

1 "Sounds very controversial, but it's true, dynamically typed languages are from a theoretical perspective untyped." -- ...and, in an instant, I know you have no idea what you're talking about. Dynamic typing just means that the types belong to the values, not the identifiers. It makes programs harder to prove, but not necessarily impossible. Inlining and parametric polymorphism has already lead to the development of link-time optimization; which solves the same type of problem that compiling optimal dynamically typed languages have: which is knowing all possible inputs and outputs. – [DeftlyHacked](#) Jan 16, 2017 at 14:00



24



Perhaps the single biggest "benefit" of dynamic typing is the shallower learning curve. There is no type system to learn and no non-trivial syntax for corner cases such as type constraints. That makes dynamic typing accessible to a lot more people and feasible for many people for whom sophisticated static type systems are out of reach. Consequently, dynamic typing has caught on in the



contexts of education (e.g. Scheme/Python at MIT) and domain-specific languages for non-programmers (e.g. [Mathematica](#)). Dynamic languages have also caught on in niches where they have little or no competition (e.g. Javascript).

The most concise dynamically-typed languages (e.g. Perl, APL, J, K, [Mathematica](#)) are domain specific and can be significantly more concise than the most concise general-purpose statically-typed languages (e.g. [OCaml](#)) in the niches they were designed for.

The main disadvantages of dynamic typing are:

- Run-time type errors.
- Can be very difficult or even practically impossible to achieve the same level of correctness and requires vastly more testing.
- No compiler-verified documentation.
- Poor performance (usually at run-time but sometimes at compile time instead, e.g. Stalin Scheme) and unpredictable performance due to dependence upon sophisticated optimizations.

Personally, I grew up on dynamic languages but wouldn't touch them with a 40' pole as a professional unless there were no other viable options.

Share Improve this answer

answered Apr 19, 2010 at 18:09

Follow



J D

48.6k ● 14 ● 174 ● 277

2 I would say lower barrier to entry but mastery is no less of a learning curve. – [Erik Reppen](#) Jun 16, 2013 at 20:37

Isn't the learning curve less because you don't have a type system to learn? – [J D](#) Jul 11, 2018 at 21:50

There's still a type system. You can make reasonable guesses about what happens when you add a bool and a string but it often helps a lot to know some actual details of how types are coerced in a dynamically typed language. That's what a lot of the strict-only folks don't get. We actually learn this stuff. – [Erik Reppen](#) Aug 8, 2018 at 20:00

@ErikReppen: We're using different definitions of "type system". I was referring to not having to learn a static type system, e.g. algebraic data types, generics. The "types" you are referring to are just data. The fact that some functions reject some data at run-time is universal. – [J D](#) Aug 8, 2018 at 23:18 ✎



From Artima's [Typing: Strong vs. Weak, Static vs. Dynamic](#) article:

12



strong typing prevents mixing operations between mismatched types. In order to mix types, you must use an explicit conversion

weak typing means that you can mix types without an explicit conversion

In the Pascal Costanza's paper, [Dynamic vs. Static Typing — A Pattern-Based Analysis](#) (PDF), he claims that in some cases, static typing is more error-prone than dynamic typing. Some statically typed languages force you to manually emulate dynamic typing in order to do "The Right Thing". It's discussed at [Lambda the Ultimate](#).

Share Improve this answer

edited Jun 20, 2020 at 9:12

Follow



Community Bot

1 • 1

answered Sep 24, 2008 at 4:10




Mark Cidade

99.8k • 33 • 229 • 237

1 "static typing is more error-prone than dynamic typing" - Yes, yes, and DOUBLE yes! I've had a lot of experience in both types of languages, and in every case the dynamic language "just works" while the static requires 2x the debugging time (See C++ and Delphi). This is frequently due to type issues, especially passing data between modules and functions with crazy types. Even though there are all sorts of theoretical bugs dynamic languages can supposedly cause, in practice, it's VERY rare for me to run into a bug caused by type coercion unless you are a poor programmer abusing dynamic types. – [dallin](#) Mar 25, 2013 at 21:07

8 I read a draft Costanza paper a few years ago. Everywhere he had written "static" he really meant specifically "Java". I gave him dozens of counter examples in languages like OCaml that disproved his claims but he went ahead and published it anyway. By the looks of that paper, he's still publishing the same old nonsense. For example, in that paper he claims "C# is generally a bad copy of Java". That

has no place in a scientific paper... – [J D](#) Dec 1, 2013 at 14:08

@dallin my experience is the entire opposite: having to program a lot in C, C++, Java, Python, Perl and the likes, I would never start anything bigger than a small tweak program in a dynamically typed language unless forced. In Python, I still shiver when thinking about a WSGI project : the callbacks I had to overeritr were passed in references of objects, and the code seemed to worked fine, when it crashed because it turned out that sometimes it's not objects but some elemental types being passed. A language which makes it this easy to create buggy stuff like that is straight out dangerous. – [Michael Beer](#) May 30, 2018 at 1:42 

@MichaelBeer You could also say a language like C/C++ that lets you directly manage memory is straight out dangerous! I've certainly wrestled with memory errors for hours. Huge Java projects are no picnic either. In any language, you have to understand the dangers of the language and good practices. The absolute worst projects I've ever worked on were team PHP projects with little structure, but I've also worked on projects with dynamic languages that were a dream when they used a good framework and good programming practices. – [dallin](#) May 30, 2018 at 16:59

@dallin Agreed, every language got its pitfalls. But the flaws I referred to are inherent to any dynamically typed language, the possibility to manipulate memory directly is no inherent property of statically typed languages. You can imagine dynamically typed languages that let you manipulate mem directly. I agree that C++ is a straight desaster, with the inventor of the language himself believes not a single person in this planet being able to know all parts of the language. However, this can not be blamed on C++ being statically typed, but a monster that had been growing since 30 years... – [Michael Beer](#) May 30, 2018 at 19:05



3



It depends on context. There a lot benefits that are appropriate to dynamic typed system as well as for strong typed. I'm of opinion that the flow of dynamic types language is faster. The dynamic languages are not constrained with class attributes and compiler thinking of what is going on in code. You have some kinda freedom. Furthermore, the dynamic language usually is more expressive and result in less code which is good. Despite of this, it's more error prone which is also questionable and depends more on unit test covering. It's easy prototype with dynamic lang but maintenance may become nightmare.

The main gain over static typed system is IDE support and surely static analyzer of code. You become more confident of code after every code change. The maintenance is peace of cake with such tools.

Share Improve this answer

answered Nov 30, 2009 at 21:47

Follow



AndrewG

1,429 ● 2 ● 16 ● 21



0



There are lots of different things about static and dynamic languages. For me, the main difference is that in dynamic languages the variables don't have fixed types; instead, the types are tied to values. Because of this, the exact code that gets executed is undetermined until runtime.



In early or naïve implementations this is a huge performance drag, but modern JITs get tantalizingly close to the best you can get with optimizing static compilers. (in some fringe cases, even better than that).

Share Improve this answer

answered Sep 24, 2008 at 4:34

Follow



Javier

62.4k ● 9 ● 81 ● 126



0



It is all about the right tool for the job. Neither is better 100% of the time. Both systems were created by man and have flaws. Sorry, but we suck and making perfect stuff.



I like dynamic typing because it gets out of my way, but yes runtime errors can creep up that I didn't plan for.

Where as static typing may fix the aforementioned errors, but drive a novice(in typed languages) programmer crazy trying to cast between a constant char and a string.

Share Improve this answer

answered Sep 24, 2008 at 5:47

Follow



J.J.

4,892 ● 1 ● 26 ● 29



-3



Static Typing: The languages such as Java and Scala are static typed.

The variables have to be defined and initialized before they are used in a code.



for ex. `int x; x = 10;`



`System.out.println(x);`

Dynamic Typing: Perl is an dynamic typed language.

Variables need not be initialized before they are used in code.

`y=10;` use this variable in the later part of code

Share Improve this answer

answered Jun 27, 2013 at 7:12

Follow



Prakhyat

1,019 ● 9 ● 18

5 This has nothing to do with the type system. – [Thiago Negri](#)
Nov 20, 2013 at 14:27



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.