What is the difference between a strongly typed language and a statically typed language?

Asked 10 years, 3 months ago Active 24 days ago Viewed 198k times

448

267

Also, does one imply the other?

type-safety strong-typing static-typing

edited Sep 11 '16 at 19:33
Jamal
731 ● 6 ● 22 ● 30

asked Apr 22 '10 at 12:02
JDelage
10.8k ● 21 ● 71 ● 105

9 Note near-duplicate at stackoverflow.com/questions/2351190/… – Norman Ramsey Apr 23 '10 at 5:24

4 Tcl is strongly typed. It only has string :P – nawfal Jul 24 '14 at 2:27

22 @nawfal nah, that's stringly typed ;) – geekonaut Sep 8 '15 at 18:52

8 Answers

Active Oldest Votes

550

> What is the difference between a strongly typed language and a statically typed language?

A statically typed language has a type system that is checked at compile time by the implementation (a compiler or interpreter). The type check rejects some programs, and programs that pass the check usually come with some guarantees; for example, the compiler guarantees not to use integer arithmetic instructions on floating-point numbers.

There is no real agreement on what "strongly typed" means, although the most widely used definition in the professional literature is that in a "strongly typed" language, it is not possible for the programmer to work around the restrictions imposed by the type system.

+300

This term is almost always used to describe statically typed languages.

Static vs dynamic

The opposite of statically typed is "dynamically typed", which means that

1. Values used at run time are classified into types.
2. There are restrictions on how such values can be used.
3. When those restrictions are violated, the violation is reported as a (dynamic) type error.

For example, Lua, a dynamically typed language, has a string type, a number type, and a Boolean type, among others. In Lua every value belongs to *exactly* one type, but this is not a requirement for all dynamically typed languages. In Lua, it is permissible to concatenate two strings, but it is not permissible to concatenate a string and a Boolean.

Strong vs weak

The opposite of "strongly typed" is "weakly typed", which means you can work around the type system. C is notoriously weakly typed because any pointer type is convertible to any other pointer type simply by casting. Pascal was intended to be strongly typed, but an oversight in the design (untagged variant records) introduced a loophole into the type system, so technically it is weakly typed. Examples of truly strongly typed languages include CLU, Standard ML, and Haskell. Standard ML has in fact undergone several revisions to remove loopholes in the type system that were discovered after the language was widely deployed.

What's really going on here?

Overall, it turns out to be not that useful to talk about "strong" and "weak". Whether a type system has a loophole is less important than the exact number and nature of the loopholes, how likely they are to come up in practice, and what are the consequences of exploiting a loophole. In practice, **it's best to avoid the terms "strong" and "weak" altogether**, because

- Amateurs often conflate them with "static" and "dynamic".
- Apparently "weak typing" is used by some persons to talk about the relative prevalance or absence of implicit conversions.
- Professionals can't agree on exactly what the terms mean.
- Overall you are unlikely to inform or enlighten your audience.

The sad truth is that when it comes to type systems, **"strong" and "weak" don't have a universally agreed on technical meaning.** If you want to discuss the relative strength of type systems, it is better to discuss exactly what guarantees are and are not provided. For example, a good question to ask is this: "is every value of a given type (or class) guaranteed to have been created by calling one of that type's constructors?" In C the answer is no. In CLU, F#, and Haskell it is yes. For C++ I am not sure—I would like to know.

By contrast, *static typing* means that programs are *checked before being executed*, and a program might be rejected before it starts. *Dynamic typing* means that the types of *values* are checked *during* execution, and a poorly typed operation might cause the program to halt or otherwise signal an error at run time. A primary reason for static typing is to rule out programs that might have such "dynamic type errors".

> Does one imply the other?

On a pedantic level, no, because the word "strong" doesn't really mean anything. But in practice, people almost always do one of two things:

- They (incorrectly) use "strong" and "weak" to mean "static" and "dynamic", in which case they (incorrectly) are using "strongly typed" and "statically typed" interchangeably.
- They use "strong" and "weak" to compare properties of static type systems. It is very rare to hear someone talk about a "strong" or "weak" dynamic type system. Except for FORTH, which doesn't really have any sort of a type system, I can't think of a dynamically typed language where the type system can be subverted. Sort of by definition, those checks are bulit into the execution engine, and every operation gets checked for sanity before being executed.

Either way, if a person calls a language "strongly typed", that person is very likely to be talking about a statically typed language.

edited Apr 25 at 14:49

answered Apr 23 '10 at 5:17

Norman Ramsey

182k ● 56 ● 335 ● 517

3 @Adam: Evidently not correct enough to be upvoted :) Because Cletus's answer contains so many misconceptions (although I edited out the worst of them), I felt compelled to spell everything out in words of one syllable... – Norman Ramsey Apr 23 '10 at 16:14

1 Well I upvoted you :) Even the word "compile" is not a clear one with today's VMs running dynamic languages. Technically Java and C# are both compiled twice (JIT) and both do some type analysis. A language like Javascript running in .NET vm might be more Typesafe because of the VM. – Adam Gent Apr 24 '10 at 21:06

2 I'm so confused now! Okay, dear great gladiators of the arena, can a poor soul like me go with the following simple understanding? 1.Static:Values are associated with the type during compile time and not run time.2.Dynamic:Values are associated to the type during runtime, hence the type of the value can change during runtime~ so more prone to type casting related issues during runtime. 3. Strong/weak: Forget it! These are not technical terms and just bad nomenclature. It depends on what context one is talking about. Can I carry on my life with this simple understanding? :(– Saurabh Patil Apr 30 '14 at 16:08

"is every value of a given type (or class) guaranteed to have been created by calling one of that type's constructors?" In C the answer is no. Could someone provide an example situation where this occurs in C? I suppose it involves casting pointers to structs? – corazza Aug 31 '14 at 18:26

Strong and Weak Typing: no such classification. – Raúl Feb 24 '15 at 9:57

251

This is often misunderstood so let me clear it up.

Static/Dynamic Typing

Static typing is where the type is bound to the *variable*. Types are checked at compile time.

Dynamic typing is where the type is bound to the *value*. Types are checked at run time.

So in Java for example:

```
String s = "abcd";
```

`s` will "forever" be a `String`. During its life it may point to different `String`s (since `s` is a reference in Java). It may have a `null` value but it will never refer to an `Integer` or a `List`. That's static typing.

In PHP:

```
$s = "abcd";          // $s is a string
$s = 123;             // $s is now an integer
$s = array(1, 2, 3);  // $s is now an array
$s = new DOMDocument; // $s is an instance of the DOMDocument class
```

That's dynamic typing.

Strong/Weak Typing

(Edit alert!)

Strong typing is a phrase with no widely agreed upon meaning. Most programmers who use this term to mean something other than static typing use it to imply that there is a type discipline that is enforced by the compiler. For example, CLU has a strong type system that does not allow client code to create a value of abstract type except by using the constructors provided by the type. C has a somewhat strong type system, but it can be "subverted" to a degree because a program can always cast a value of one pointer type to a value of another pointer type. So for example, in C you can take a value returned by `malloc()` and cheerfully cast it to `FILE*`, and the compiler won't try to stop you—or even warn you that you are doing anything dodgy.

(The original answer said something about a value "not changing type at run time". I have known many language designers and

(The original answer said something about a value "not changing type at run time". I have known many language designers and compiler writers and have not known one that talked about values changing type at run time, except possibly some very advanced research in type systems, where this is known as the "strong update problem".)

Weak typing implies that the compiler does not enforce a typing discpline, or perhaps that enforcement can easily be subverted.

The original of this answer conflated weak typing with **implicit conversion** (sometimes also called "implicit promotion"). For example, in Java:

```
String s = "abc" + 123; // "abc123";
```

This is code is an example of implicit promotion: 123 is implicitly converted to a string before being concatenated with `"abc"`. It can be argued the Java compiler rewrites that code as:

```
String s = "abc" + new Integer(123).toString();
```

Consider a classic PHP "starts with" problem:

```
if (strpos('abcdef', 'abc') == false) {
  // not found
}
```

The error here is that `strpos()` returns the index of the match, being 0. 0 is coerced into boolean `false` and thus the condition is actually true. The solution is to use `===` instead of `==` to avoid implicit conversion.

This example illustrates how a combination of implicit conversion and dynamic typing can lead programmers astray.

Compare that to Ruby:

```
val = "abc" + 123
```

which is a runtime error because in Ruby the *object* 123 is *not* implicitly converted just because it happens to be passed to a `+` method. In Ruby the programmer must make the conversion explicit:

```
val = "abc" + 123.to_s
```

Comparing PHP and Ruby is a good illustration here. Both are dynamically typed languages but PHP has lots of implicit conversions and Ruby (perhaps surprisingly if you're unfamiliar with it) doesn't.

Static/Dynamic vs Strong/Weak

The point here is that the static/dynamic axis is independent of the strong/weak axis. People confuse them probably in part because strong vs weak typing is not only less clearly defined, there is no real consensus on exactly what is meant by strong and weak. For this reason strong/weak typing is far more of a shade of grey rather than black or white.

So to answer your question: another way to look at this that's *mostly* correct is to say that static typing is compile-time type safety and strong typing is runtime type safety.

The reason for this is that variables in a statically typed language have a type that must be declared and can be checked at compile time. A strongly-typed language has values that have a type at run time, and it's difficult for the programmer to subvert the type system without a dynamic check.

But it's important to understand that a language can be Static/Strong, Static/Weak, Dynamic/Strong or Dynamic/Weak.

edited Feb 22 '12 at 23:12

alex
416k ● 184 ● 813 ● 945

answered Apr 22 '10 at 12:10

cletus
557k ● 151 ● 869 ● 925

Instead of saying $s is a integer or string ,would have been better if you say the type is associated with the "abcd" or 1234 not with the variable $s. – Srinivas Reddy Thatiparthy Apr 22 '10 at 12:18

Excellent answer with clear examples. However, I think it doesn't completely address the confusion as to WHY people ask about strong/static as a twin pair of concepts. For example, the OP's wording of "does static typing IMPLY strong typing?" Your answer emphasizes their independence. To continue the clarification of why the strong is often PAIRED with static, Norman Ramsey's previous answer is very good: stackoverflow.com/questions/376611/… – JasDev Apr 22 '10 at 12:37

1 `"abc" + 123` is a *runtime* error, not a compile error in ruby. If it was a compile error, ruby would be statically typed. – sepp2k Apr 22 '10 at 17:35

The weak typing examples need to be improved (see my answer) but otherwise nice formating. – Adam Gent Apr 22 '10 at 21:07

In my opion strong vs weak typgin is this: Strong: "c" + True = runtime error or compile time error. Weak: "c" + True = "b" or "d" because everything is treated as raw bytes. Strong: C#, Ruby, C++ Weak: Assembly, C (due to implicit void pointers) – Jonathan Allen Apr 23 '10 at 5:27

20

Both are poles on two different axis:

- strongly typed vs. weakly typed
- statically typed vs. dynamically typed

Strongly typed means, a will not be automatically converted from one type to another. Weakly typed is the opposite: Perl can use a string like `"123"` in a numeric context, by automatically converting it into the int `123`. A strongly typed language like python will not do this.

Statically typed means, the compiler figures out the type of each variable at compile time. Dynamically typed languages only figure out the types of variables at runtime.

answered Apr 22 '10 at 12:11

Daren Thomas
58.4k ● 38 ● 138 ● 187

7 I have to disagree. A strongly typed language is one that knows what the types are at runtime. A weakly typed language is one that doesn't like Assembly. Your example is on a third axis, "implicit vs explicit conversions". – Jonathan Allen Apr 23 '10 at 5:33

1 Actually normal I agree Jonathan but you don't have to have the types available at runtime to be strongly typed if you do complete static analysis and don't allow casting. (see my edited answer). – Adam Gent Apr 23 '10 at 11:38

1 Python is an example of a dynamically typed and a strongly typed language – MaRoBet Feb 18 '19 at 16:31

14

Strongly typed means that there are restrictions between conversions between types.

Statically typed means that the types are not dynamic - you can not change the type of a variable once it has been created.

edited Jul 8 at 12:06

Pshemo
108k ● 18 ● 159 ● 229

answered Apr 22 '10 at 12:07

Svetlozar Angelov
18.9k ● 6 ● 54 ● 67

To demonstrate this: In a strongly-typed language, you cannot compare "5" == 5 and get it to be true: Strings are not integers. If my memory serves, most modern scripting languages are strongly dynamically typed. Tcl/Tk, however is weakly typed - Everything can be treated as a string. – Little Bobby Tables Apr 22 '10 at 12:14

Bobby, in a weakly typed language "5" == 5 is read as 0x35 == 0x05. Or in other words, everything is treated as raw bytes. – Jonathan Allen Apr 23 '10 at 5:30

I have to disagree with you both. Take Lua; you can compare "5" == 5 and it will return false, however a quick conversion can be done by going "5" + 0. – RCIX Apr 27 '10 at 4:32

12

Data Coercion does not necessarily mean weakly typed because sometimes its syntacical sugar:
The example above of Java being weakly typed because of

```
String s = "abc" + 123;
```

Is not weakly typed example because its really doing:

```
String s = "abc" + new Integer(123).toString()
```

Data coercion is also not weakly typed if you are constructing a new object. Java is a very bad example of weakly typed (and any language that has good reflection will most likely not be weakly typed). Because the runtime of the language always knows what the type is (the exception might be native types).

This is unlike C. C is the one of the best examples of weakly typed. The runtime has no idea if 4 bytes is an integer, a struct, a pointer or a 4 characters.

The runtime of the language really defines whether or not its weakly typed otherwise its really just opinion.

EDIT: *After further thought this is not necessarily true as the runtime does not have to have all the types reified in the runtime system to be a Strongly Typed system. Haskell and ML have such complete static analysis that they can potential ommit type information from the runtime.*

edited Apr 23 '10 at 11:36

answered Apr 22 '10 at 12:23

Adam Gent
42.7k ● 19 ● 138 ● 180

B is probably a better, if a little less well known, example. – Tom Hawtin - tackline Apr 22 '10 at 23:06

Javascript is also rather weakly type but because there are so few types and because you can't really construct new types. – Adam Gent Apr 22 '10 at 23:48

10

Answer is already given above. Trying to differentiate between strong vs week and static vs dynamic concept.

What is Strongly typed VS Weakly typed?

Strongly Typed: Will not be automatically converted from one type to another

In Go or Python like strongly typed languages "2" + 8 will raise a type error, because they don't allow for "type coercion".

Weakly (loosely) Typed: Will be automatically converted to one type to another: Weakly typed languages like JavaScript or Perl won't throw an error and in this case JavaScript will results '28' and perl will result 10.

Perl Example:

```
my $a = "2" + 8;
print $a,"\n";
```

Save it to main.pl and run `perl main.pl` and you will get output 10.

What is Static VS Dynamic type?

In programming, programmer define static typing and dynamic typing with respect to the point at which the variable types are checked. Static typed languages are those in which type checking is done at compile-time, whereas dynamic typed languages are those in which type checking is done at run-time.

- Static: Types checked before run-time
- Dynamic: Types checked on the fly, during execution

What is this means?

In Go it checks typed before run-time (static check). This mean it not only translates and type-checks code it’s executing, but it will scan through all the code and type error would be thrown before the code is even run. For example,

```
package main

import "fmt"

func foo(a int) {
    if (a > 0) {
        fmt.Println("I am feeling lucky (maybe).")
    } else {
        fmt.Println("2" + 8)
    }
}

func main() {
    foo(2)
```

```
}
```

Save this file in main.go and run it, you will get compilation failed message for this.

```
go run main.go
# command-line-arguments
./main.go:9:25: cannot convert "2" (type untyped string) to type int
./main.go:9:25: invalid operation: "2" + 8 (mismatched types string and int)
```

But this case is not valid for Python. For example following block of code will execute for first foo(2) call and will fail for second foo(0) call. It's because Python is dynamically typed, it only translates and type-checks code it’s executing on. The else block never executes for foo(2), so "2" + 8 is never even looked at and for foo(0) call it will try to execute that block and failed.

```
def foo(a):
    if a > 0:
        print 'I am feeling lucky.'
    else:
        print "2" + 8
foo(2)
foo(0)
```

You will see following output

```
python main.py
I am feeling lucky.
Traceback (most recent call last):
  File "pyth.py", line 7, in <module>
    foo(0)
  File "pyth.py", line 5, in foo
    print "2" + 8
TypeError: cannot concatenate 'str' and 'int' objects
```

edited Dec 31 '19 at 16:19

Leśny Rumcajs

1,737 ● 1 ● 12 ● 27

answered Mar 7 '19 at 23:35

Balkrishna

1,827 ● 2 ● 18 ● 28

8

One does not imply the other. For a language to be **statically** typed it means that the types of all variables are known or inferred at compile time.

A **strongly** typed language does not allow you to use one type as another. C is a weakly typed language and is a good example of

A **strongly** typed language does not allow you to use one type as another. C is a weakly typed language and is a good example of what strongly typed languages don't allow. In C you can pass a data element of the wrong type and it will not complain. In strongly typed languages you cannot.

answered Apr 22 '10 at 12:11

Joe Cannatti
4,541 ● 5 ● 33 ● 52

7

Strong typing probably means that variables have a well-defined type and that there are strict rules about combining variables of different types in expressions. For example, if A is an integer and B is a float, then the strict rule about A+B might be that A is cast to a float and the result returned as a float. If A is an integer and B is a string, then the strict rule might be that A+B is not valid.

Static typing probably means that types are assigned at compile time (or its equivalent for non-compiled languages) and cannot change during program execution.

Note that these classifications are not mutually exclusive, indeed I would expect them to occur together frequently. Many strongly-typed languages are also statically-typed.

And note that when I use the word 'probably' it is because there are no universally accepted definitions of these terms. As you will already have seen from the answers so far.

answered Apr 22 '10 at 12:13

High Performance Mark
72.3k ● 7 ● 94 ● 143
