When should weak types be discouraged?

Asked 15 years, 9 months ago Modified 12 years, 9 months ago

Viewed 2k times

4

When should weak types be discouraged? Are weak types discouraged in big projects? If the left side is strongly typed like the following would that be an exception to the rule?

```
int i = 5
string sz = i
sz = sz + "1"
i  = sz
```

Does any languages support similar syntax to the above? Tell me more about pros and cons to weak types and situations related.

weak-typing

Share

Improve this question

Follow

edited Feb 28, 2009 at 17:20

Eddie

54.4k ● 22 ● 128 ● 146

asked Feb 28, 2009 at 7:06

user34537

This question doesn't make any sense: the terms "weak typing" and "strong typing" don't have any defined meaning. Unless you define them, it is impossible to answer this question. – Jörg W Mittag Feb 28, 2009 at 19:50

2 I think the example the OP gave is sufficient to explain the difference between weak and strong typing. – Jason Baker Mar 8, 2009 at 1:31

7 Answers

Sorted by: Highest score (default)

17

I think you are confusing "weak typing" with "dynamic typing".

The term "weak typing" means "not strongly typed", which means that the value of a memory location is allowed to vary from what it's type indicates it should be.

C is an example of a weakly typed language. It allows code like this to be written:

```
typedef struct
{
    int x;
    int y;

} FooBar;

FooBar foo;
char * pStr = &foo;
pStr[0] = 'H';
pStr[1] = 'i';
pStr[2] = '\0';
```

That is, it allows a FooBar instance to be treated as if it was an array of characters.

In a strongly typed language, that would not be allowed. Either a compiler error would be generated, or a run time exception would be thrown, but never, at any time, would a FooBar memory address contain data that was not a valid FooBar.

C#, Java, Lisp, Java Script, and Ruby are examples of languages where this type of thing would not be allowed. They are strongly typed.

Some of those languages are "statically typed", which means that variable types are assigned at compile time, and some are "dynamically typed", which means that variable types are not known until runtime. "Static vs Dynamic" and "Weak vs Strong" are orthogonal issues. For example, Lisp is a "strong dynamically typed" language, whereas "C" is a "weak statically typed language".

Also, as others have pointed out, there is a distinction between "inferred types" and types specified by the programmer. The "var" keyword in C# is an example of type inference. However, it's still a statically typed construct because the compiler infers the type of a variable at compile time, rather than at runtime.

So, what your question really is asking is:

> What are the relative merits and drawbacks of static typing, dynamic typing, weak typing, stong typing, inferred static types, and user specified static types.

I provide answers to all of these below:

Static typing

Static typing has 3 primary benefits:

1. Better tooling support
2. A Reduced likely hood of certain types of bugs
3. Performance

The user experience and accuracy of things like intellisence, and refactoring is improved greatly in a statically typed language because of the extra information that the static types provide. If you type "a." in a code editor and "a" has a static type then the compiler knows everything that could legally come after the "." and can thus show you an accurate completion list. It's possible to support some scenarios in a dynamically typed language, but they are much more limited.

Also, in a program without compiler errors a refactoring tool can identify every place a particular method, variable, or type is used. It's not possible to do that in a dynamically typed language.

The second benefit is somewhat controversial. Proponents of statically typed languages like to make that claim. Opponents of statically typed languages, however, contend that the bugs they catch are trivial, and that they would get caught by testing anyways. But, you do get notification of things like misspelled variable or method names up front, which can be helpful.

Statically typed languages also enable better "data flow analysis", which when combined with things like Microsoft's SAL (or similar tools) can help find potential security problems.

Finally, with static typing, compilers can do a lot more optimization, and so can produce faster code.

Drawbacks:

The main drawback for static typing is that it restricts the things you can do. You can write programs in dynamically typed languages that you can't write in statically typed languages. Ruby on Rails is a good example of this.

Dynamic Typing

The big advantage of dynamic typing is that it's much more powerful than static typing. You can do a lot of really cool stuff with it.

Another one is that it requires less typing. You don't have to specify types all over the place.

Drawbacks:

Dynamic typing has 2 main draw backs:

1. You don't get as much "hand holding" from the compiler or IDE
2. It's not suitable for critical performance scenarios. For example, no one writes OS Kernels in Ruby.

Strong typing:

The biggest benefit of strong typing is security. Enforcing strong typing usually requires some type of runtime support. If a program can proove type safety then a lot of security issues, such as buffer overuns, just go away.

Weak typing:

The big drawback of strong typing, and the big benefit of weak typing, is performance.

When you can access memory any way you like, you can write faster code. For example a database can swap objects out to disk just by writing out their raw bytes, and not needing to resort to things like "ISerializable" interfaces. A video game can throw away all the data associated with one level by just running a single free on a large buffer, rather than running destructors for many small objects.

Being able to do those things requires weak typing.

Type inference

Type inference allows a lot of the benefits of static typing without requiring as much typing.

User specified types

Some people just don't like type inference because they like to be explicit. This is more of a style thing.

Share Improve this answer Follow

edited Mar 11, 2009 at 15:52

answered Mar 8, 2009 at 2:44

Scott Wisniewski
25k ● 8 ● 62 ● 90

++, but javascript in the same category with C# and java? – dkretz Mar 8, 2009 at 3:32

1 Javascript is dynamically typed, while C# and Java are (mostly) statically typed. However, all 3 are strongly typed. – Scott Wisniewski Mar 8, 2009 at 5:05

Technically C# allows weak typing but you have to very specifically request it using the "unsafe" keyword in combination with changing your project options... – Qwertie Mar 11, 2009 at 15:55

What you call "hand holding from the compiler" I call eliminating 3/4 of my mistakes before ever running the program :) – Qwertie Mar 11, 2009 at 15:58

"The big advantage of dynamic typing is that it's much more powerful than static typing. You can do a lot of really cool stuff with it." I was thinking about that... does dynamic typing actually add expressiveness? yes it can make polymorphism

easier, but that's about it. – Iraimbilanja Mar 11, 2009 at 16:02

3

Weak typing is an attempt at language simplification. While this is a worthy goal, weak typing is a poor solution.

Weak typing such as is used in COM Variants was an early attempt to solve this problem, but it is fraught with peril and frankly causes more trouble than it's worth. Even Visual Basic programmers, who will put up with all sorts of rubbish, correctly pegged this as a bad idea and backronymed Microsoft's ETC (Extended Type Conversion) to Evil Type Cast.

Do not confuse inferred typing with weak typing. Inferred typing is strong typing inferred from context at compile time. A good example is the `var` keyword, used in C# to declare a variable suitable to receive the value of a LINQ expression.

By contrast, weak typing is inferred each and every time an expression is evaluated. This is illustrated in the question's sample code. Another example would be use of untyped pointers in C. Very handy yet begging for trouble.

Inferred typing addresses the same issue as weak typing, *without* introducing the problems associated with weak typing. It is therefore a preferred alternative whenever the host language makes it available.

Share Improve this answer Follow

edited Mar 8, 2009 at 1:24

answered Feb 28, 2009 at 9:35

Peter Wone
18.7k ● 14 ● 94 ● 147

1 +1 for 'rubbish' and 'backronymed' (hadn't seen that one before) in the same sentence. Solid answer. – mechanical_meat Feb 28, 2009 at 9:52

-1 for confusion of type inference with dynamic typing. See herbsutter.wordpress.com/2008/06/20/… – Pete Kirkham Feb 28, 2009 at 10:30

LINQ doesn't use any dynamic typing. In fact, LINQ doesn't have anything to do with typing at all, it's just a) an API with some interesting properties and b) syntactic sugar for an internal DSL to simplify usage of said API. – Jörg W Mittag Feb 28, 2009 at 19:53

Wrong word, ok. I watched a video with Anders going on about dynamic typing and I can't get the word out of my head. – Peter Wone Mar 1, 2009 at 7:35

2

They should almost always be discouraged. The only type of code that I can think of where it would be required is low-level code that requires some pointer voodoo.

And to answer your question, C supports code like that (except of course for not having a string type), and that sounds like something PHP or Perl would have (but I could be totally wrong on that).

answered Mar 8, 2009 at 1:34

Jason Baker
198k • 138 • 382 • 520

2

"

> When should weak types be discouraged? Are weak types discouraged in big projects? If the left side is strongly typed like the following would that be an exception to the rule?
>
> ```
> int i = 5    string sz = i    sz = sz + "1"
> i  = sz
> ```
>
> Does any languages support similar syntax to the above? Tell me more about pros and cons to weak types and situations related.

"

Perhaps you could program your own library to do that.

In C++ you can use something called an "operator overload", which means that you can declare a variable of one type to be initialized as a variable of another type. That is what makes the statement:

```
[std::string str = "Hello World";][1]
```

specifically you would define a function (where the variable's type is T and B is the type you want to set it as)

work, even though any text between quotes is interpreted as an array of chars.

```
T& T::operator= ( const B s );
```

Please note that this is a class's member function Also note that you will probably want to have some sort of function that reverses this manipulation if you want to use it liberally - something like

```
B& T::operator= ( const T s);
```

C++ is powerful enough to allow you to make an object generally weakly typed, but if you want to treat it purely weakly typed, you will want to make just a single variable type that can be used as any primitive, and use only functions that take a pointer to void. Believe me, it is a lot easier to use strongly typed programming when it is available.

I personally prefer strongly typed, because I don't need to worry about the errors that come when I don't know what a variable is meant to do. For example, if I wanted to write a function to talk to a person - and that function used the person's height, weight, name, number of children, etc. - but you gave me a color, I would get an error because you can't really determine most of these things for a color using an algorithm that is very simple.

As far as the pros of weakly typed, you might want to get used to loosely typed programming if you are programming something to be run within a program(i.e. a web browser or a UNIX shell). JavaScript and Shell Script are weakly typed.

I would suggest that a programming language like assembly language is one of the only harware-level weakly typed languages, but the flavor of Assembly language I've seen attaches a type to each variable depending on the allocated size, i.e. word, dword, qword.

I hope I gave you a good explanation and did not put any words in your mouth.

Share Improve this answer

Follow

answered Mar 6, 2012 at 2:10

Jesse

17 ● 1

1

Weak types are by their very nature less robust than strong types, because you don't tell the machine exactly what to do - instead the machine has to figure out what you meant. This often works quite adequately, but in general it is not clear what the result should be. What is, for example, a string multiplied by float?

Share Improve this answer

Follow

answered Feb 28, 2009 at 7:31

Joonas Pulakka

36.6k ● 29 ● 108 ● 171

2 It's obviously a strat. ROCK ON! \m/ – Dan Lew Feb 28, 2009 at 8:06

One thing you should keep in mind is that machine language is weakly typed. (You can't get much more specific about what you want the computer to do than machine code). It's statically typed (the instructions indicate operand types), but because no enforcement is made, the typing is weak.
– Scott Wisniewski Mar 11, 2009 at 15:51

1

> Does any languages support similar syntax to the above?

Perl allows you to treat some numbers and strings interchangeably. For example, "5" + "1" will give you 6. The problem with this sort of thing in general is that it can be hard to avoid ambiguity: should "5" + 1 be "51" or "6"? Perl gets around this by having a separate operator for string concatenation, and reserving + for numeric addition.

Other languages would have to sort out whether you mean to do a concatenation or an addition, and (if relevant) what type or representation the result will be.

Share Improve this answer

Follow

answered Mar 8, 2009 at 3:39

MandyK

1,221 ● 10 ● 11

I did ASP/VBScript coding and work with legacy code without "option strict" which allows weak typing.

It was a hell in many times, especially in the hands of less experienced programmers. We got all stupid errors takes ages to diagnose.

One of the stupid examples was like this:

```
'Config
    Dim pass
    pass = "asdasd"


If NOT pass = Request("p") Then
Response.Write "login failed"
REsponse.End()
End If
```

So far so good but if the user changes pass to an integer password, guess what it won't work anymore because int pass != string pass (from querystring). I thought it supposed to work but it didn't *I can't remember the exact piece of code*.

I hate weak typing, instead of stupid debugging session I can spend extra seconds for typing exact type of a variable.

Simply put, in my experience especially in the big projects and especially with unexperienced developers it's just trouble.

Share Improve this answer

Follow

answered Feb 28, 2009 at 10:50

dr. evil

27.2k ● 37 ● 134 ● 202

so the problem was pass was being NOT'd instead of pass == Request("p")? in the question examples i have implicit conversion, would this still happen in my situation? unless i misunderstood i am sure it wouldnt. – user34537 Feb 28, 2009 at 18:19

the problem is ASP tried to cast one of the inputs as integer and other one as string then compare, obviously failed. As developer I was expecting it to always cast it to string.
– dr. evil Mar 1, 2009 at 7:11