Full Stack Type Safety

Szymon Pyżalski

Egnyte Inc.

Europython 2020

Outline

Premise

Typing basics

Our typical stack

Annotations and ORM

Enforcing the contract

Summary

Our goal

Our goal

- Catch typing errors ASAP (not later than in CI)

Our goal

- Catch typing errors ASAP (not later than in CI)
- Catch typing errors that span layers of stack

Problems

Problems

- Type annotation system in Python is new and immature

Problems

- Type annotation system in Python is new and immature
- Various layers of stack feature different typing paradigms

Problems

- Type annotation system in Python is new and immature
- Various layers of stack feature different typing paradigms
- We tend to test layers in separation

Weak vs strong

Weak vs strong

Weak typing A value can be misinterpreted unless we care about the type by ourselves.

Weak vs strong

Weak typing A value can be misinterpreted unless we care about the type by ourselves.

Strong typing We are protected from misinterpretations by the type system.

Weak typing

```
#include <stdio.h>
short int fun(int* x) {
        short int y = *(short int*)x;
        return y + 1;
}

int main(int argc, char** argv) {
        int a = -10;
        int b = 777777;
        printf("%u\n", a);      // prints: 4294967286
        printf("%d\n", fun(&b)); // prints: -8654
}
```

Static vs dynamic

Static vs dynamic

Static typing The types of objects can be determined during compile time.

Static vs dynamic

Static typing The types of objects can be determined during compile time.

Dynamic typing The types of objects are determined during runtime.

Dynamic typing in python

```
def sum(xs, init):
    result = init
    for x in xs:
        result += x
    return result


print(sum([1, 2, 3], 0)) # prints 6
print(sum({'a': 'b', 'c': 'd'}, 'Keys: ')) # prints: Keys: ac
```

Static typing with inference

```
package main

import "fmt";

func fact(n int) int {
        result := 1
        for i := 1; i <=n; i++ {
                result *= i
        }
        return result
}

func main() {
        x := 10
        y := fact(5)
        fmt.Println(x)
        fmt.Println(y)
}
```

Strict vs loose

Strict vs loose

Strict typing Type conversions must be explicit. Type mismatch exceptions.

Strict vs loose

Strict typing Type conversions must be explicit. Type mismatch exceptions.

Loose typing Type conversions can be implicit.

Stricter than Python

```haskell
import Data.String.Utils (join)

list2Str :: [[Char]] -> [Char]
list2Str xs = if xs then "No elements" else (join "," xs)

main = do
        putStrLn $ list2Str []
        putStrLn 10 -- Error
```

Looser than Python

```
1 + ’a’                    // ’1a’
{} + 2                     // 0
’abc’ + [’d’, ’e’, ’f’]    // "abcd,e,f"
{} + ’z’                   // NaN
{} + {}                    // NaN
{} + []                    // 0
[] + {}                    // "[object Object]"
```

Duck vs ???

Duck vs ???

Duck typing ~~Interfaces~~ Protocols are implemented implicitly. Object is compatible with a protocol if it implements required methods.

Duck vs ???

Duck typing ~~Interfaces~~ Protocols are implemented implicitly. Object is compatible with a protocol if it implements required methods.

??? Classes must inherit from a class in order to be compaticle, or at least be marked as implementing the protocol.

Duck vs platonic

Duck typing ~~Interfaces~~ Protocols are implemented implicitly. Object is compatible with a protocol if it implements required methods.

Platonic typing Classes must inherit from a class in order to be compaticle, or at lease be marked as implementing the protocol.

Structural vs nominal

Structural typing ~~Interfaces~~ Protocols are implemented implicitly. Object is compatible with a protocol if it implements required methods.

Nominal typing Classes must inherit from a class in order to be compaticle, or at lease be marked as implementing the protocol.

The pythonish language

They say	We say

The pythonish language

They say	We say
throw	raise

The pythonish language

They say	We say
throw	raise
array	list

The pythonish language

They say	We say
throw	raise
array	list
list	deque

The pythonish language

They say	We say
throw	raise
array	list
list	deque
blatant abuse of exceptions	StopIteration

The pythonish language

They say	We say
throw	raise
array	list
list	deque
blatant abuse of exceptions	StopIteration
interfaces	protocols

The pythonish language

They say	We say
throw	raise
array	list
list	deque
blatant abuse of exceptions	StopIteration
interfaces	protocols

Static but duck-typed

```
package main
import "fmt"

type Duck interface {
        swim(x int, y int)
        quack() string
}

type Mallard struct {
        x, y int
}

func (m *Mallard) swim(x, y int) {
        m.x += x
        m.y += y
}

func (m Mallard) quack() string {
        return "Quack quaaaack"
}

func swimThenQuack(d Duck) {
        d.swim(1, 1)
        fmt.Println(d.quack())
}

func main() {
        donald := Mallard{x: 0, y: 0}
        swimThenQuack(&donald)
        fmt.Println(donald)
}
```

Typing models

- Strong vs weak typing
- Static vs dynamic typing
- Strict vs loose typing
- Structural vs nominal typing

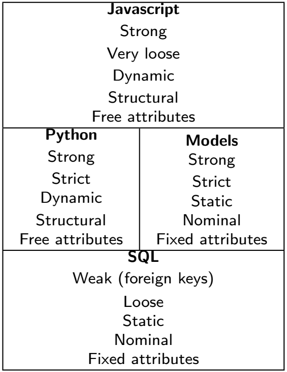
Typing models

- Strong vs weak typing
- Static vs dynamic typing
- Strict vs loose typing
- Structural vs nominal typing
- Free vs fixed attributes

Our typical stack

Javascript
Strong
Very loose
Dynamic
Structural
Free attributes
Python
Strong
Strict
Dynamic
Structural
Free attributes
SQL
Weak (foreign keys)
Loose
Static
Nominal
Fixed attributes

Our typical stack



Javascript
Strong
Very loose
Dynamic
Structural
Free attributes

Python
Strong
Strict
Dynamic
Structural
Free attributes

Models
Strong
Strict
Static
Nominal
Fixed attributes

SQL
Weak (foreign keys)
Loose
Static
Nominal
Fixed attributes

Weakness of SQL foreign keys

```
UPDATE books set author_id = (
        SELECT id FROM publishers
        WHERE name="Chilton Books"
);
```

ORM improving type safety

```
b = Book.objects.get(id=1)
b.author = Publisher.objects.get(name=’Chilton Books’)
```

mypy enters the game

<table>
<tr><td colspan="3"><b>Javascript</b>
Strong
Very loose
Dynamic
Structural
Free attributes</td></tr>
<tr><td><b>mypy</b>
Strong
String
Static
Preference for nominal
Fixed attributes</td><td><b>Python</b>
Strong
Strict
Dynamic
Structural
Free attributes</td><td><b>Models</b>
Strong
Strict
Static
Nominal
Fixed attributes</td></tr>
<tr><td colspan="3"><b>SQL</b>
Weak (foreign keys)
Loose
Static
Nominal
Fixed attributes</td></tr>
</table>

Demo 1

Django and mypy working together

mypy and Django pros and cons

mypy and Django pros and cons

- Pro: Recognizes the relationship between column types and python types

mypy and Django pros and cons

- Pro: Recognizes the relationship between column types and python types
- Pro: Recognizes the idea of null

mypy and Django pros and cons

- Pro: Recognizes the relationship between column types and python types
- Pro: Recognizes the idea of null
- Con: Can't handle problems with incomplete data

mypy and Django pros and cons

- Pro: Recognizes the relationship between column types and python types
- Pro: Recognizes the idea of null
- Con: Can't handle problems with incomplete data
- Con: Requires a mypy plugin

Considering the JSON

<table>
<tr><td colspan="3"><b>Javascript</b>
Strong
Very loose
Dynamic
Structural
Free attributes</td><td rowspan="2"><b>JSON</b>
No typing above primitives</td></tr>
<tr><td><b>mypy</b>
Strong
String
Static
Preference for nominal
Fixed attributes</td><td><b>Python</b>
Strong
Strict
Dynamic
Structural
Free attributes</td><td><b>Models</b>
Strong
Strict
Static
Nominal
Fixed attributes</td></tr>
<tr><td colspan="4"><b>SQL</b>
Weak (foreign keys)
Loose
Static
Nominal
Fixed attributes</td></tr>
</table>

One solution

<table>
<tr><td colspan="3"><b>Typescript</b>
Strong
Strict
Static
Structural
Fixed attributes</td><td rowspan="2"><b>JSON</b>
No typing above primitives</td><td rowspan="2"><b>OpenAPI3</b>
Schema
Tests
Code generation</td></tr>
<tr><td><b>mypy</b>
Strong
String
Static
Preference for nominal
Fixed attributes</td><td><b>Python</b>
Strong
Strict
Dynamic
Structural
Free attributes</td><td><b>Models</b>
Strong
Strict
Static
Nominal
Fixed attributes</td></tr>
<tr><td colspan="5"><b>SQL</b>
Weak (foreign keys)
Loose
Static
Nominal
Fixed attributes</td></tr>
</table>

Demo 2

Enforcing the contract

Takeaways

Takeaways

- There are tools for code safety enforcement in a Python stack that are worth consideration

Takeaways

- There are tools for code safety enforcement in a Python stack that are worth consideration
- They are not yet perfect and we can't expect to catch all errors

Future can bring

Future can bring

- Support for more patterns in type annotations without plugins

Future can bring

- Support for more patterns in type annotations without plugins
- Tools based on code annotations instead of descriptors (strawberry-graphql, pydantic,)

Tools used

- django-stubs A distribution of code annotations for django complete with a mypy plugin
- spectacular A schema generator for django-rest-framework
- openapi-generator Code generator that can create boilerplate code for several languages/frameworks based on OpenAPI3.