State of Using Rust in Python

sample Rust crate

`Cargo.toml`

```toml
[package]
name = "sum_even"
version = "0.1.0"

[dependencies]
libc = "0.2"

[lib]
crate-type = ["cdylib", "rlib"]
```

`cdylib` crate type creates a dynamically linked library.

src/lib.rs

```rust
extern crate libc;

use libc::{uint32_t, size_t};
use std::slice;

pub fn sum_even(numbers: &[u32]) -> u32 {
    numbers
        .iter()
        .filter(|&v| v % 2 == 0)
        .sum()
}

#[no_mangle]
pub extern fn c_sum_even(n: *const uint32_t, len: size_t) -> uint32_t {
    let numbers = unsafe {
        assert!(!n.is_null());

        slice::from_raw_parts(n, len as usize)
    };
    let sum = sum_even(&numbers);
    sum as uint32_t
}
```

Build the dylib

```
$ cargo build
$ nm target/debug/libsum_even.dylib | grep sum_even
0000000000001150 t __ZN8sum_even8sum_even17h64c50448ecece425E
0000000000000da0 t __ZN8sum_even8sum_even28_$u7b$$u7b$closure$u7d$$u7d$17h7f9a4d0c9f65174bE
00000000000011b0 T _c_sum_even
```

Common Methods

- ctypes
- rust-cpython
- PyO3
- CFFI

ctypes

- Python built-in
- ABI mode
- Global Interpreter Lock is released during C function call

```python
import sys, ctypes
from ctypes import POINTER, c_uint32, c_size_t

prefix = {'win32': ''}.get(sys.platform, 'lib')
extension = {'darwin': '.dylib', 'win32': '.dll'}.get(sys.platform, '.so')
lib = ctypes.cdll.LoadLibrary(prefix + "sum_even" + extension)

lib.c_sum_even.argtypes = (POINTER(c_uint32), c_size_t)
lib.c_sum_even.restype = ctypes.c_uint32

def sum_of_even(numbers):
    buf_type = c_uint32 * len(numbers)
    buf = buf_type(*numbers)
    return lib.c_sum_even(buf, len(numbers))

print(sum_of_even([1, 2, 3, 4, 5, 6]))
```

rust-cpython & PyO3

- Links to libpython
- Featureful, supports classes, functions, inheritance, import existing Python modules and etc.
- Global Interpreter Lock, but you can release it in code
- Build and distribute with setuptools-rust/pyo3-pack
- rust-cpython: macro_rules style macros, compiles on Rust stable release

PyO3 example

```
#![feature(specialization)]

extern crate pyo3;
extern crate sum_even;

use pyo3::prelude::*;

/// This module is a python moudle implemented in Rust.
#[pymodinit]
fn sum_even(_py: Python, m: &PyModule) -> PyResult<()> {

    #[pyfn(m, "sum_even")]
    fn sum_even_py(numbers: Vec<u32>) -> PyResult<u32> {
        Ok(sum_even::sum_even(&numbers))
    }

    Ok(())
}
```

setup.py

```
from setuptools import setup
from setuptools_rust import Binding, RustExtension

setup(
    name='sum_even',
    version='1.0',
    rust_extensions=[
        RustExtension('sum_even.sum_even', 'Cargo.toml', binding=Binding.PyO3)
    ],
    packages=['sum_even'],
    zip_safe=False
)
```

CFFI

- API mode
- does not link to libpython
- release GIL when calling a C function
- Often needs to write high level wrapper in pure Python

milksnake

Universal wheels, support Python 2 & 3 with a single dylib

```
void initXXX(void) {}
void PyInit_XXX(void) {}
```

setup.py

```python
# -*- coding: utf-8 -*-
from setuptools import setup, find_packages


def build_native(spec):
    build = spec.add_external_build(
        cmd=['cargo', 'build', '--release'],
        path='./cabi'
    )

    spec.add_cffi_module(
        module_path='crfsuite._native',
        dylib=lambda: build.find_dylib('pycrfsuite', in_path='target/release'),
        header_filename=lambda: build.find_header('pycrfsuite.h', in_path='include'),
        rtld_flags=['NOW', 'NODELETE']
    )


setup(
    name='crfsuite',
    version='0.2.8',
    url='https://github.com/bosondata/crfsuite-rs',
    description='Python binding for crfsuite',
    packages=find_packages(),
    zip_safe=False,
    platforms='any',
    setup_requires=['milksnake'],
    install_requires=['milksnake'],
    milksnake_tasks=[
        build_native
    ]
)
```

Exposing C-ABIs

```
pub struct Model;

ffi_fn! {
    unsafe fn pycrfsuite_model_open(s: *const c_char) -> Result<*mut Model> {
        let path_cstr = CStr::from_ptr(s);
        let model = crfsuite::Model::from_file(path_cstr.to_str().unwrap())?;
        Ok(Box::into_raw(Box::new(model)) as *mut Model)
    }
}
```

Generate C headers with cbindgen

build.rs

```rust
extern crate cbindgen;

use std::env;

fn main() {
    let crate_dir = env::var("CARGO_MANIFEST_DIR").unwrap();
    let mut config: cbindgen::Config = Default::default();
    config.language = cbindgen::Language::C;
    cbindgen::generate_with_config(&crate_dir, config)
      .unwrap()
      .write_to_file("target/pycrfsuite.h");
}
```

Generated C header example

```c
#ifndef PYCRFSUITE_H_INCLUDED
#define PYCRFSUITE_H_INCLUDED

#include <stdint.h>
#include <stdlib.h>

enum CrfErrorCode {
  CRF_ERROR_CODE_NO_ERROR = 0,
  CRF_ERROR_CODE_PANIC = 1,
  CRF_ERROR_CODE_CRF_ERROR = 2,
};
typedef uint32_t CrfErrorCode;
typedef struct Model Model;

void pycrfsuite_err_clear();
CrfErrorCode pycrfsuite_err_get_last_code();
void pycrfsuite_init();
void pycrfsuite_model_destroy(Model *m);
Model *pycrfsuite_model_open(const char *s);
Model *pycrfsuite_model_from_bytes(const uint8_t *bytes, size_t len);
void pycrfsuite_model_dump(Model *m, int fd);
#endif /* PYCRFSUITE_H_INCLUDED */
```

Catch Rust panic and raise Python exception

```
thread_local! {
    pub static LAST_ERROR: RefCell<Option<ErrorKind>> = RefCell::new(None);
    pub static LAST_BACKTRACE: RefCell<Option<(Option<String>, Backtrace)>> = RefCell::new(None);
}

std::panic::catch_unwind<F: FnOnce() -> R + UnwindSafe, R>(f: F) -> Result<R>;
std::panic::set_hook(hook: Box<Fn(&PanicInfo) + Sync + Send + 'static>);

/// Set panic hook upon initialization
#[no_mangle]
pub unsafe extern "C" fn pycrfsuite_init() {
    set_panic_hook();
}
```

On Python side, call `ffi.init_once(lib.pycrfsuite_init, 'init')` to register it.

```rust
pub unsafe fn set_panic_hook() {
    panic::set_hook(Box::new(|info| {
        let backtrace = Backtrace::new();
        let thread = thread::current();
        let thread = thread.name().unwrap_or("unnamed");

        let msg = match info.payload().downcast_ref::<&str>() {
            Some(s) => *s,
            None => {
                match info.payload().downcast_ref::<String>() {
                    Some(s) => &**s,
                    None => "Box<Any>",
                }
            }
        };

        let panic_info = match info.location() {
            Some(location) => {
                format!("thread '{}' panicked with '{}' at {}:{}",
                                     thread, msg, location.file(),
                                     location.line())
            }
            None => {
                format!("thread '{}' panicked with '{}'", thread, msg)
            }
        };

        LAST_BACKTRACE.with(|e| {
            *e.borrow_mut() = Some((Some(panic_info), backtrace));
        });
    }));
}
```

```python
from ._native import lib, ffi
from .exceptions import exceptions_by_code, CrfSuiteError


def rustcall(func, *args):
    """Calls rust method and does some error handling."""
    lib.pycrfsuite_err_clear()
    rv = func(*args)
    err = lib.pycrfsuite_err_get_last_code()
    if not err:
        return rv
    msg = lib.pycrfsuite_err_get_last_message()
    cls = exceptions_by_code.get(err, CrfSuiteError)
    exc = cls(decode_str(msg))
    raise exc


def decode_str(s, free=False):
    try:
        if s.len == 0:
            return u''
        return ffi.unpack(s.data, s.len).decode('utf-8', 'replace')
    finally:
        if free:
            lib.pycrfsuite_str_free(ffi.addressof(s))
```

There is also a project called shippai trying to simpify error handling in between Rust and Python

Keep Python objects from garbage collected

```python
class Tagger(object):
    def tag(self, xseq):
        attrs_list = ffi.new('AttributeList []', len(xseq))
        keepalive = []
        for i, items in enumerate(xseq):
            attrs = attrs_list[i]
            attrs.len = len(items)
            attr_ptr = ffi.new('Attribute []', len(items))
            keepalive.append(attr_ptr)
            for j, item in enumerate(items):
                attr = attr_ptr[j]
                name, value = _to_attr(item)
                name = ffi.from_buffer(name)
                keepalive.append(name)
                attr.name = name
                attr.value = ffi.cast('double', value)
            attrs.data = attr_ptr

        tags = rustcall(lib.pycrfsuite_tagger_tag, self.tagger, attrs_list, len(xseq))
        ffi_strs = ffi.unpack(tags.data, tags.len)
        labels = [decode_str(s) for s in ffi_strs]
        lib.pycrfsuite_tags_destroy(tags)
        return labels
```

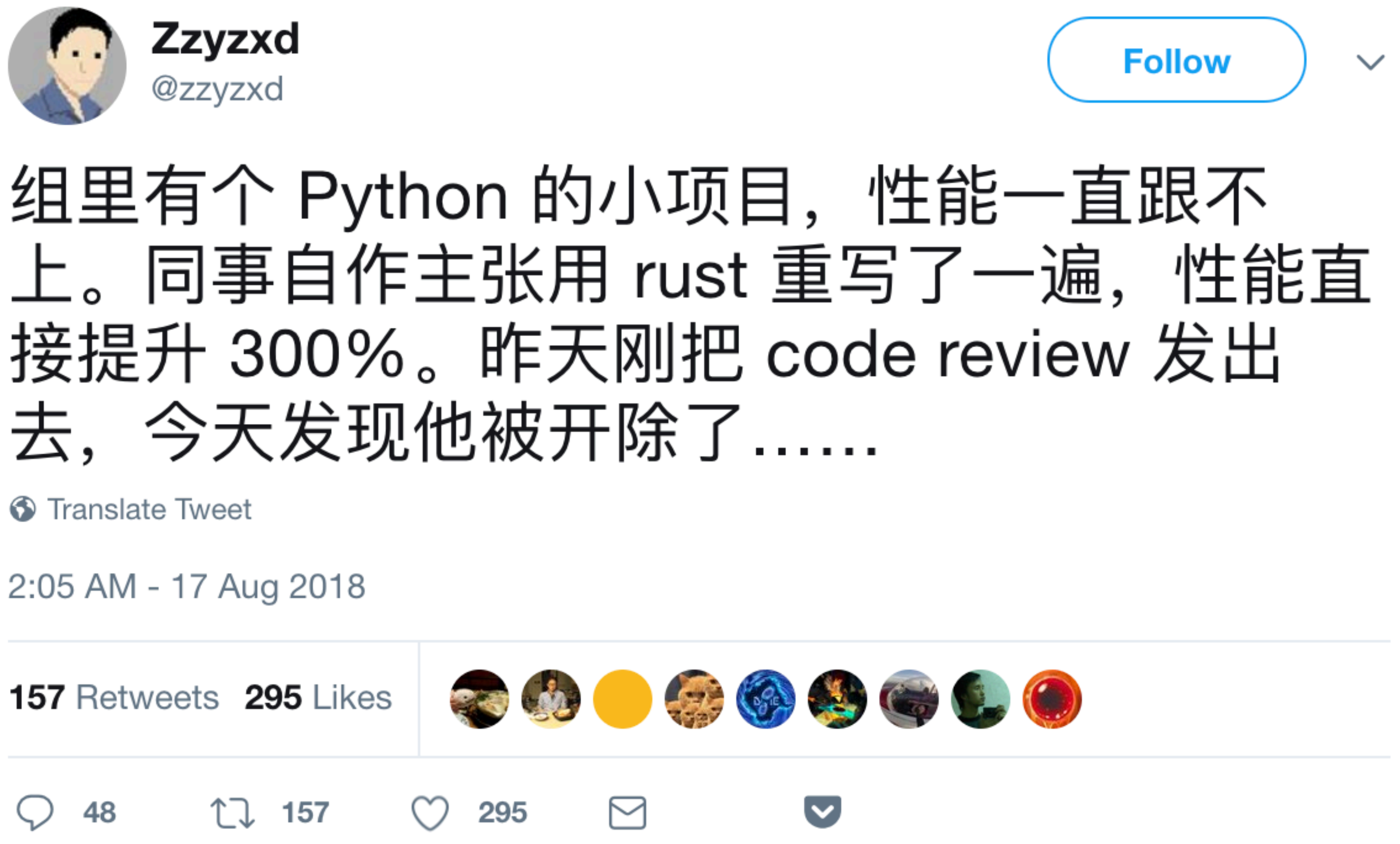
Build manylinux1 wheels

```
#!/bin/bash
set -e -x

ln -s `which cmake28` /usr/bin/cmake
# Install dependencies needed by our wheel
yum -y install gcc libffi-devel

# Install Rust
curl https://sh.rustup.rs -sSf | sh -s -- -y
export PATH=~/.cargo/bin:$PATH

# Build wheels
which linux32 && LINUX32=linux32
$LINUX32 /opt/python/cp27-cp27mu/bin/python setup.py bdist_wheel

# Audit wheels
for wheel in dist/*-linux_*.whl; do
  auditwheel repair $wheel -w dist/
  rm $wheel
done
```

```
crfsuite-0.2.8-py2.py3-none-manylinux1_x86_64.whl
crfsuite-0.2.8-py2.py3-none-manylinux1_i686.whl
```

Should you do it?

Before doing so, don't get yourself fired 😱


Zzyzxd
@zzyzxd

组里有个 Python 的小项目，性能一直跟不上。同事自作主张用 rust 重写了一遍，性能直接提升 300%。昨天刚把 code review 发出去，今天发现他被开除了……

2:05 AM - 17 Aug 2018

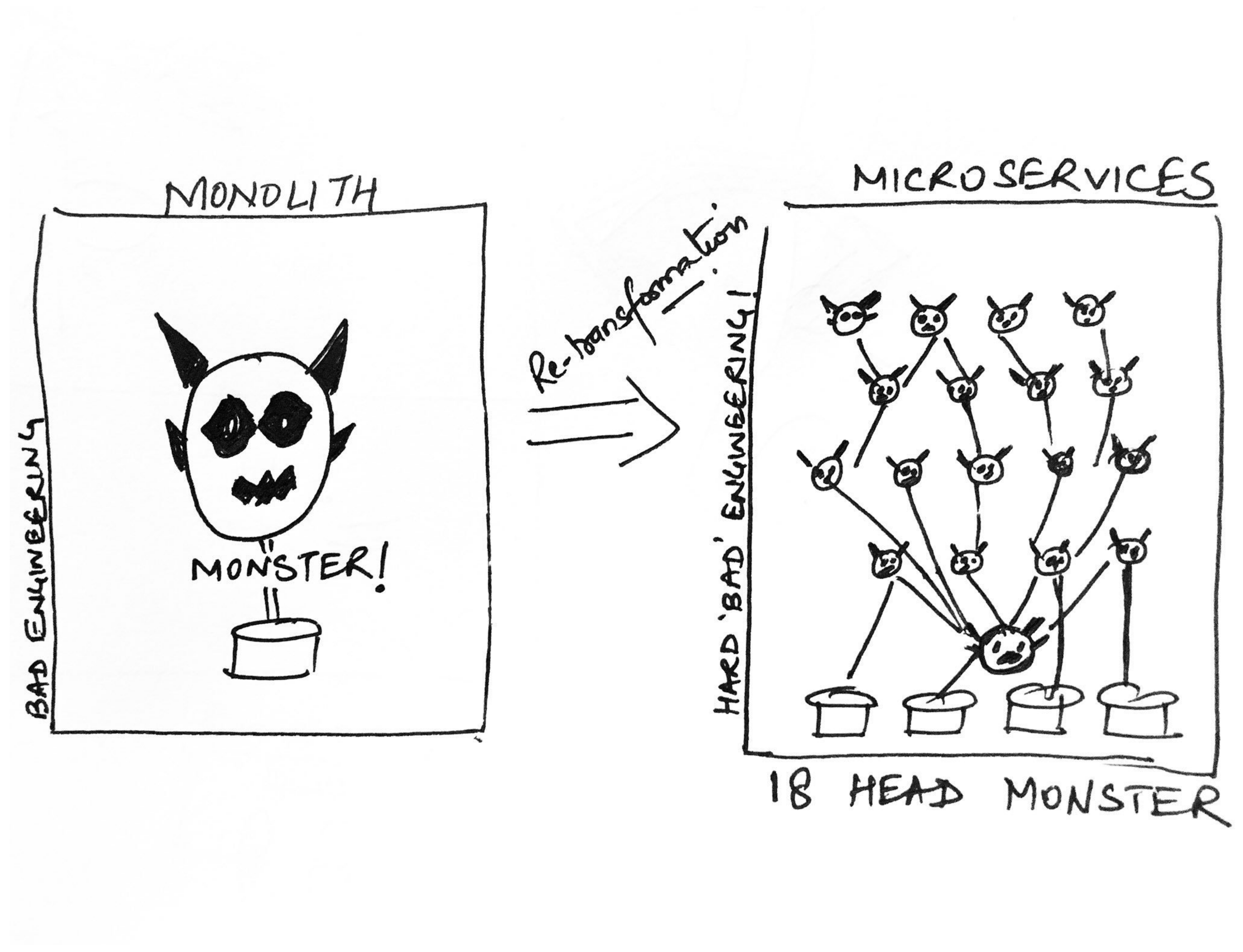
157 Retweets 295 Likes

Measure!

Measure!

And measure again!

How about microservices?



References

1. The Rust FFI Omnibus
2. Evolving Our Rust With Milksnake
3. A dive into packaging native python extensions
4. Oxidizing sourmash: Python and FFI
5. Speed up your Python using Rust

Thanks!