CROSS-PLATFORM RUST ON THE CLIENT-SIDE

By wangcong @ ByteDance

Github

Geekbang. InfoQ
极客邦科技

极客邦科技
会议推荐2019

5月

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6月

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7月

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10月

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11月

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12月

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

AGENDA

Why cross-platform and why Rust?

Our experience using Rust

Building with Rust

Client-side programming *is* hard

- Performance requirements
  - Ever-growing requirements with fixed resources
  - Need to achieve more with less battery consumption
- Implementation complexity
  - Almost monolithic architecture
  - Low-level code comes with less safety guaranteed, traditionally
- Security considerations

And, each platform is different

We have to repeat our work and make new mistakes

Going cross-platform brings us

One codebase to design, implement and review

One server-facing client

One place to implement security polices

CONCERNS

We care about user experience when doing cross-platform

- Performance
  - The app should run fast
  - The app should not be battery-hungry
- Look and feel
  - We need platform-native appearance & interactions

How far do we go cross-platform?

- Only for some components
  - Works but not enough
- All code, but with tradeoff
  - No system UI components
  - System UI components with tedious and high-cost bindings

We don't sacrifice UX, especially not in frequently-used apps like Feishu

Non-UI code in cross-platform, system languages. Optimized for performance

UI code uses platform-specific APIs

WHY RUST?

We want a systems programming language with safety guaranteed, best effort is not enough

C++ is great, but your team only lives in peace when everyone is a guru

---

While in Rust, the compiler acts as the guru, always watching your back

Guaranteed safety, that means memory safety and fearless concurrency

We can focus on the business logic, spend less time hunting for bugs

---

We have the bravery and confidence to achieve more

Other things we love about Rust

Minimal runtime that easily embeds into other languages

Performance with zero-overhead abstractions

High-level language with modern features

Scalable to large codebases, with low maintenance if done correctly

OUR EXPERIENCE USING RUST

We introduced Rust into Feishu at the beginning of 2017

We knew what to be scared of, especially in the chaotic world of systems

Unsafe Rust code

C/C++ components

The rules of Rust kept us free

Free to aggresively introduce parallelism with nothing to worry about

---

Free to call battle-tested C++ code, finding thread-safety issues on the C++ side, where mutable shared state was involved

Rust ecosystem is still young and issues do arise

When we started, HTTPS requests failed on iOS & Android due to TLS-related problems

---

After long-term production use, we found that the compression library we used could not handle `NonCompressed` DEFLATE messages

But Rust is growing quickly, and sufficient for general usage

Problems solved quickly, easy to contribute, and easy to get help

---

New libraries & tools are emerging with the qualities of Rust

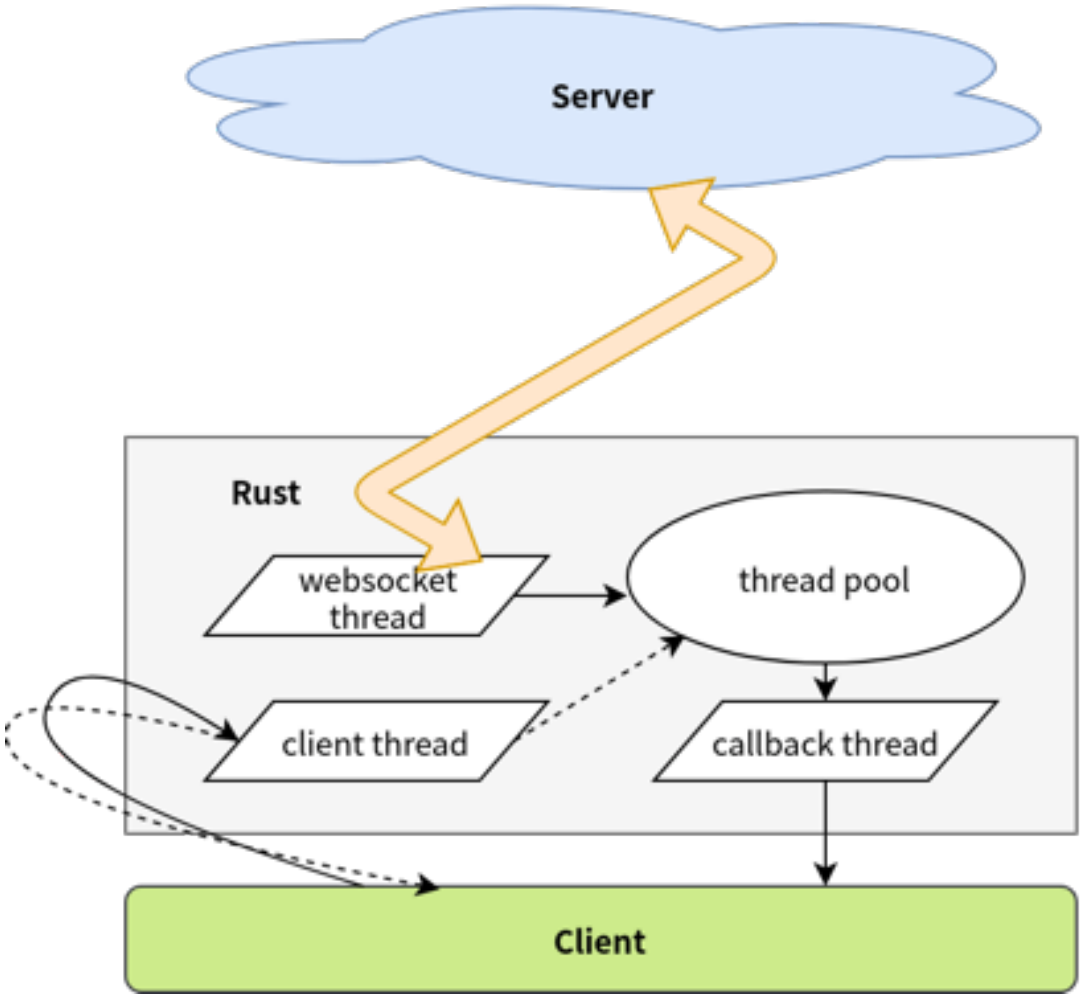
Rust is still in flux and major changes still happen

We have to do refactorings like migrating error handling from `error-chain` to `failure`, instead of focusing on our users

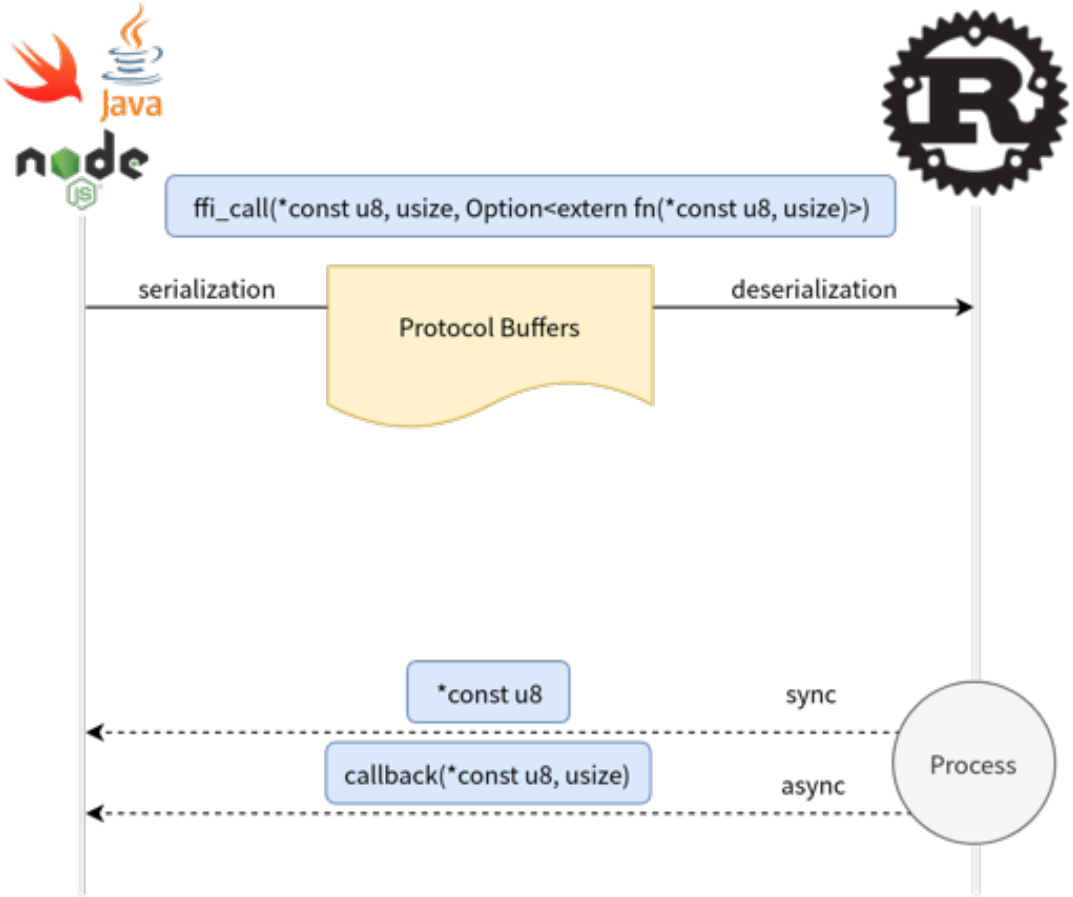
We get to play with all the treasures of C/C++.

Our Architecture

THREAD INTERACTION



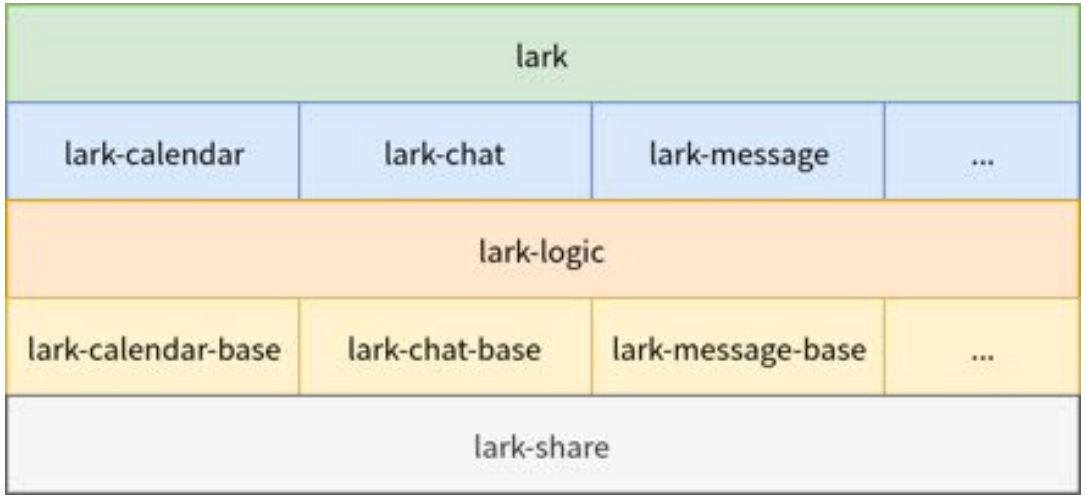
FFI


Java
node
ffi_call(*const u8, usize, Option<extern fn(*const u8, usize)>)
serialization
Protocol Buffers
deserialization
*const u8
sync
callback(*const u8, usize)
async
Process

FFI is unsafe by nature and brings inherent costs

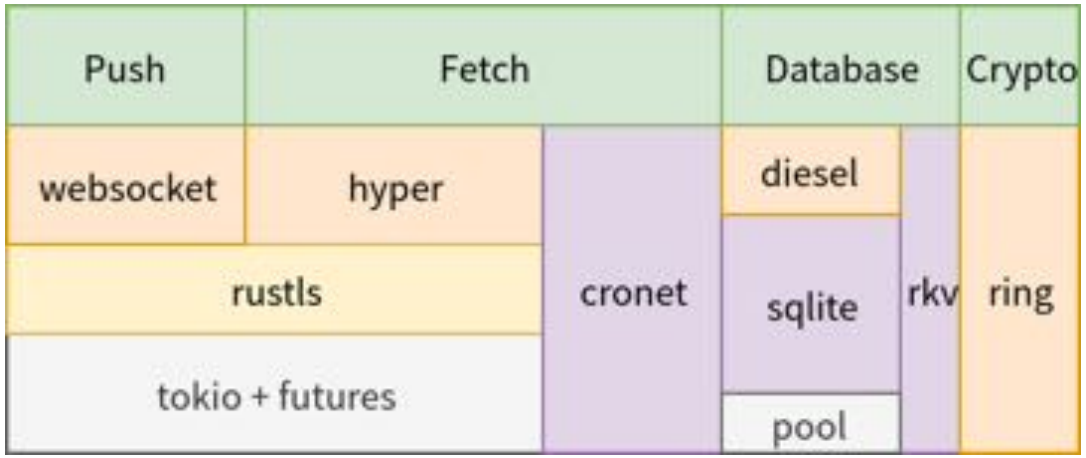
- Protocol Buffers
  - With enough abstraction, only a small number of FFI functions is kept and seldom change
  - Not suitable for large data sets or high-frequency calls
  - `Cap’n Proto` and `FlatBuffers` help reduce overhead
- Raw data types
  - When it's expected or performance demands it

MODULE DESIGN



- Every crate can be independely built, run, tested and depended on
- Managed by Cargo
- Build speed
  - Crates at the bottom should not change often
  - Excessively large crates become a bottleneck and reduce parallelism

BUILDING WITH RUST



Storage-related features built with Rust

- Migration support
- Resource usage and performance monitor
- Encryption support
- And more ...

Network-related features built with Rust

- Hedged requests and route selection
- TLS1.3 and QUIC support
- Persistent connections with adaptive heartbeat
- DNS over HTTPS
- Network diagnostics: ping, traceroute, reachability comparisons
- And more...

Challenges we have faced

- Code footprint
  - Static dispatch of generics
  - Generated code size
- Rust uses a different `LLVM` build than the one Apple ships
  - No direct bitcode compatibility

QUESTIONS?

Challenges we have faced

- Code footprint
  - Static dispatch of generics
  - Generated code size
- Rust uses a different `LLVM` build than the one Apple ships
  - No direct bitcode compatibility

THANKS

Photos from unsplash.com