QCon+ 案例研习社

QEngine：支付宝实时行情 SDK

一次 Rust 移动开发之旅 / 徐仙明

蚂蚁集团 ANT GROUP

目录

1 QEngine 是什么

2 为什么用 Rust

3 Rust 如何解决我们的问题

4 项目构建

5 挑战畅想

QEngine 是什么

支付宝股票实时行情数据 SDK

~ 5000 亿
日交易额

> 1 亿
股民

183 个
证券类 App

90%
App 交易客户数占比

5000+
沪深个股总数

>100 亿
日 PV

QEngine：支付宝股票实时行情数据 SDK

实时数据展示 | 技术指标计算

研习社

QEngine 业务特点

稳定

崩溃率 < 1/10 万, 并发安全

准确

无效数据过滤

实时

全链路 < 500ms， >5 次数据更新 /s

多端一致

JSAPI 等 H5 场景，抹平平台差异

技术指标分析

指标要求动态配置参数

架构层面： 推拉结合 / 智能缓存 / 规则校验 / 动态开关 / 巡检

目录

1 QEngine 是什么

2 为什么要用 Rust

3 Rust 如何解决我们的问题

4 项目构建

5 挑战畅想

为什么要用 Rust

QEngine 业务特点

稳定

崩溃率 < 1/10 万, 并发安全

推拉结合，解决高可用问题

Rust + Tokio 解决并发问题

准确

无效数据过滤

规则校验

数据校正

实时

全链路 < 500ms， >5 次数据更新 /s

RTS 直连服务器，减少中间商

端到端 –> 75ms

多端一致

JSAPI 等 H5 场景，抹平平台差异

核心逻辑一套代码

封装平台层 API

技术指标分析

指标要求动态配置参数

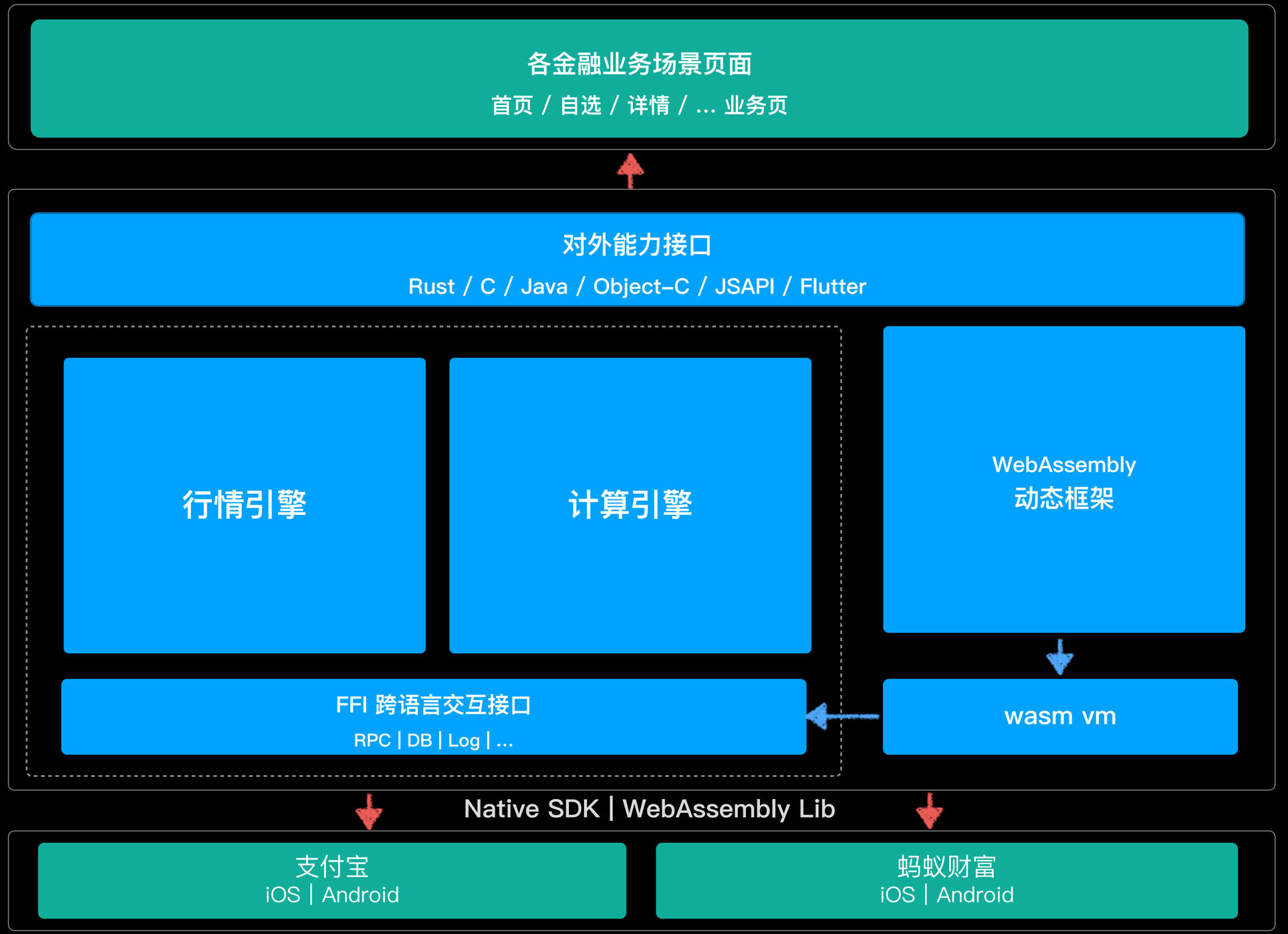
WebAssembly 动态下发计算模块

架构层面： 推拉结合 / 智能缓存 / 规则校验 / 动态开关 / 巡检

语言层面： 多端一套代码、有效处理并发、减少人为错误

QEngine 架构：

跨端、稳定、灵活



目录

1 QEngine 是什么

2 为什么要用 Rust

3 Rust 如何解决我们的问题

4 项目构建

5 挑战畅想

Rust 如何解决我们的问题

实际案例

双端一致性

- 一份代码、多端适用
- LOC：Android 27K / iOS 36K → Rust 20K
- 双端开发效率：80% ↑
- 包大小：30% ↓

架构收益

- 模块面向 API 设计
- 高内聚低耦合
- TDD 驱动
- PC 上可测试，覆盖率有保障

安全稳定

- **NPE** Case Study

并发

- 并发 Case Study

Rust 如何解决我们的问题

NPE Case Study

sum.java

```java
public static int sumOfOddNumber(List<Integer> list) {
    int sum = 0;
    for (Integer i : list) {
        if (i % 2 != 0)    // if (i.intValue() % 2 != 0)
            sum += i;      //    sum += i.intValue();
    }
    return sum;
}
```

sum.rs

```rust
/// sum of all even numbers in a vector
fn sumOfOddNumber(list: Vec<Option<i32>>) -> i32 {
    let mut sum = 0;
    for item in list {
        if let Some(i) = item {
            if i % 2 == 0 {
                sum += i;
            }
        }
    }
    sum
}
```

Rust 为什么没有 NPE？因为 Rust 没有 null！

Rust 如何解决我们的问题

并发 Case Study

concurrent.java

```java
public static void execute(Runnable r, ScheduleType type) {
    ThreadPoolExecutor executor = acquireExecutor(type);
    if (executor != null) {
        executor.execute(r);
    } else {
        Logger.warn("获取框架后台线程池失败，任务取消");
    }
}
```

concurrent.rs

```rust
async fn rpc(req: Request) -> Response {
    // implement stub !!!
    RpcClient.get(req).await?
}

let response = rpc(request).await?;
```

架构图 Credit: Tokio internals: Understanding Rust's asynchronous I/O framework from the bottom up

目录

1 QEngine 是什么

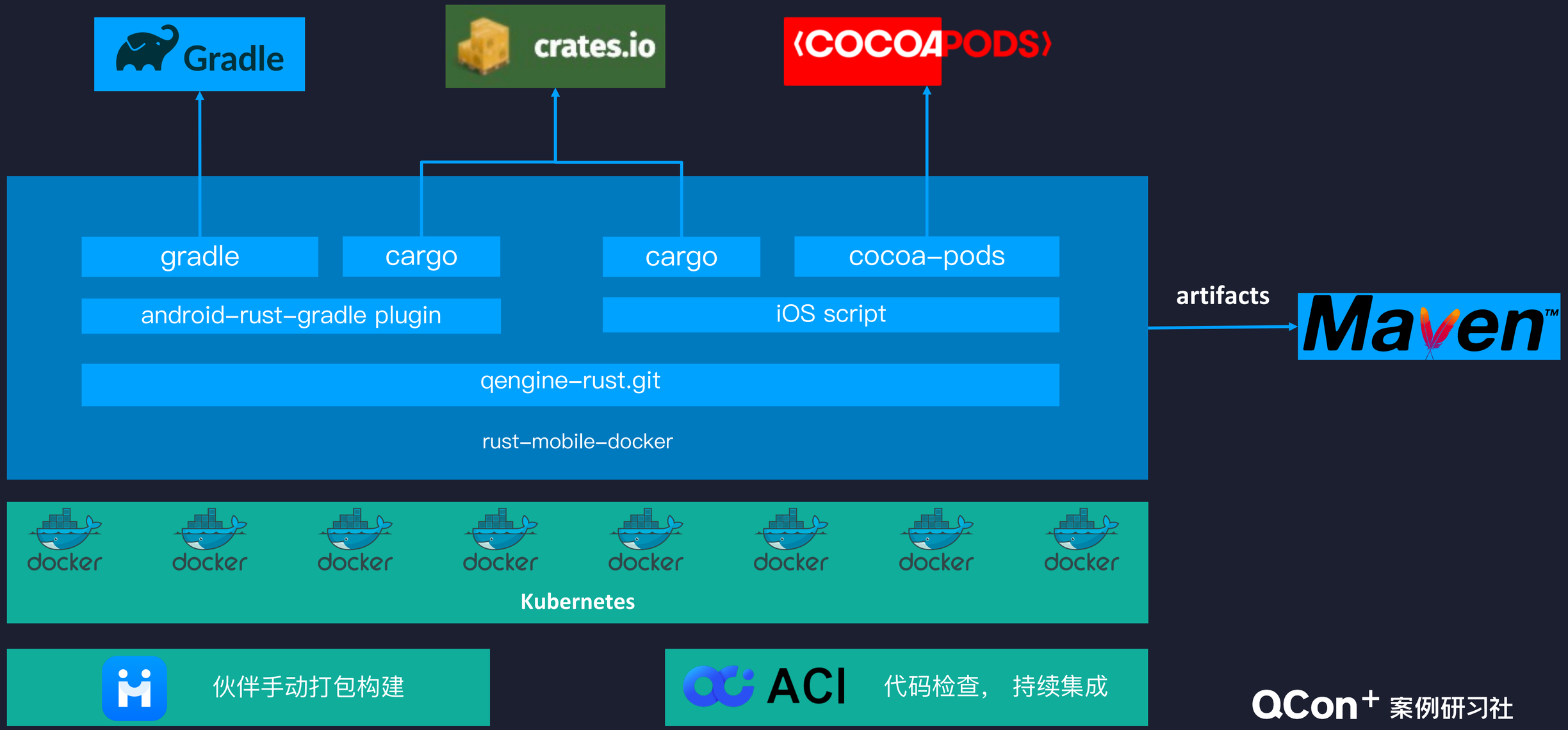
2 为什么要用 Rust

3 Rust 如何解决我们的问题

4 项目构建

5 挑战畅想

项目构建


Gradle
crates.io
COCOAPODS
gradle
cargo
cargo
cocoa–pods
android–rust–gradle plugin
iOS script
qengine–rust.git
rust–mobile–docker
artifacts
Maven
docker
docker
docker
docker
docker
docker
docker
docker
Kubernetes
伙伴手动打包构建
ACI
代码检查， 持续集成

目录

1 QEngine 是什么

2 为什么要用 Rust

3 Rust 如何解决我们的问题

4 项目构建

5 挑战畅想

挑战

我们踩过的坑

工具链不成熟

- 构建：自定义脚本连接
- FFI：封装
- mvp：最小依赖

集成调试困难

- 单测：TDD 驱动，提高单测覆盖率
- 集成：Hook 平台日志 API，关键节点落日志

concurrent.rs

```rust
#[cfg(target_os = "macos")]
pub fn init_logger() {
    log::set_logger(&MACOS_LOGGER);
}

#[cfg(target_os = "android")]
pub fn init_logger() {
    log::set_logger(&*ANDROID_LOGGER);
}

#[cfg(target_os = "ios")]
pub fn init_logger() {
    log::set_logger(&*IOS_LOGGER);
}

#[cfg(target_os = "macos")]
/// Mac 上的日志直接输出到console
struct MacosLogger;
impl log::Log for MacosLogger {
    fn enabled(&self, _: &Metadata) -> bool { true }
    fn log(&self, _: &log::Record) {
        if self.enabled(record.metadata()) {
            println!("{} - {}", record.level(), record.args());
        }
    }
    fn flush(&self) {}
}

static MACOS_LOGGER: MacosLogger = MacosLogger;
```

畅想

社区共建

完善移动开发生态

- 完善构建工具链
- 提供标准的 FFI 类库，抹平平台差异
- IDE（Android Studio / Xcode）支持
- 包管理

移动框架和类库

- 提供类 Flutter / Xamarin 的跨平台 UI 框架
- RPC / 缓存 / 安全等各种跨平台类库

小步快跑，快速迭代

THANKS

QCon+ 案例研习社