QCon+ 案例研习社

Rust 在 PingCAP 的应用实践

张文博

目录

Rust 语言的现状

- 特征
  - 可靠性
  - 生产力

- 效率
  - 学习效率
  - 运行效率
  - 开发效率

Rust 语言的现状

使用现状

1. AWS 从 2017 年开始就用 Rust 实现了无服务器计算平台：AWS Lambda 和 AWS Fargate，用 Rust 重写了 Bottlerocket OS 和 AWS Nitro 系统，这两个是弹性计算云（EC2）的重要服务
2. Cloudflare 是 Rust 的重度用户，DNS、无服务计算、网络包监控等基础设施都与 Rust 密不可分
3. Dropbox 的底层存储服务完全由 Rust 重写，达到了数万 PB 的规模
4. Google 除了在安卓系统的部分模块中使用 Rust 外，还在最新的操作系统 Fuchsia 中重度使用了 Rust

Rust 语言的现状

使用现状

5. Facebook 使用 Rust 来增强自己的网页端、移动端和 API 服务的性能，同时还写了 Hack 编程语言的虚拟机
6. Microsoft 使用 Rust 为 Azure 平台提供一些组件，其中包括 IoT 的核心服务
7. GitHub 和 npmjs.com，使用 Rust 提供高达每天 13 亿次的 npm 包下载
8. Rust 目前已经成为全世界区块链平台的首选开发语言
9. TiKV：国内开源分布式数据库

Rust 语言的现状

2024 Roadmap

- **降低学习门槛**
  - 更精确分析，更少繁琐
  - 更容易、更直接地表达代码的意图
  - 改进异步支持
  - 让 dyn Trait 更有用
- **扩展生态系统**
- **Rust 项目扩展**

Rust 在 TiKV 项目上的实践

如何快速上手 Rust 项目

- **掌握基础概念**
  - 变量在内存中的位置
  - 类型的定义
  - 并发编程方式
  - 泛型
- **为开源项目添砖加瓦**
  - 从简单的功能上手
  - 向编译器学习
  - 向 Code Reviewer 学习

Rust 在 TiKV 项目上的实践

Rust 的类型系统的特点

- **强类型，静态类型**
  - [ConfigValue]()
- **对多态的支持**
  - 参数多态
  - 特设多态
  - 子类型多态
- **支持类型推导**

```rust
#[derive(Clone, PartialEq)]
pub enum ConfigValue {
    Duration(u64),
    Size(u64),
    U64(u64),
    F64(f64),
    I32(i32),
    U32(u32),
    Usize(usize),
    Bool(bool),
    String(String),
    BlobRunMode(String),
    IOPriority(String),
    Module(ConfigChange),
    Skip,
    None,
}
```

```rust
macro_rules! impl_from {
    ($from: ty, $to: tt) => {
        impl From<$from> for ConfigValue {
            fn from(r: $from) -> ConfigValue {
                ConfigValue::$to(r)
            }
        }
    };
}
impl_from!(u64, U64);
impl_from!(f64, F64);
impl_from!(i32, I32);
impl_from!(u32, U32);
impl_from!(usize, Usize);
impl_from!(bool, Bool);
impl_from!(String, String);
impl_from!(ConfigChange, Module);
```

Rust 在 TiKV 项目上的实践

TiKV 在线配置更新

- ConfigValue
- Rust 过程宏

```rust
/// There are four type of fields inside derived OnlineConfig struct:
/// 1. `#[online_config(skip)]` field, these fields will not return
/// by `diff` method and have not effect of `update` method
/// 2. `#[online_config(hidden)]` field, these fields have the same effect of
/// `#[online_config(skip)]` field, in addition, these fields will not appear
/// at the output of serializing `Self::Encoder`
/// 3. `#[online_config(submodule)]` field, these fields represent the
/// submodule, and should also derive `OnlineConfig`
/// 4. normal fields, the type of these fields should be implment
/// `Into` and `From` for `ConfigValue`
pub trait OnlineConfig<'a> {
    type Encoder: serde::Serialize;
    /// Compare to other config, return the difference
    fn diff(&self, _: &Self) -> ConfigChange;
    /// Update config with difference returned by `diff`
    fn update(&mut self, _: ConfigChange);
    /// Get encoder that can be serialize with `serde::Serializer`
    /// with the disappear of `#[online_config(hidden)]` field
    fn get_encoder(&'a self) -> Self::Encoder;
    /// Get all fields and their type of the config
    fn typed(&self) -> ConfigChange;
}
```

Rust 在 TiKV 项目上的实践

TiKV 在线配置更新

- [ConfigValue]()
- Rust 过程宏

```rust
fn get_config_attrs(attrs: &[Attribute]) -> Result<(bool, bool, bool)> {
    let (mut skip, mut hidden, mut submodule) = (false, false, false);
    for attr in attrs {
        if !is_attr("online_config", attr) {
            continue;
        }
        match attr.parse_args::<Ident>()? {
            name if name == "skip" => skip = true,
            name if name == "hidden" => hidden = true,
            name if name == "submodule" => submodule = true,
            name => {
                return Err(Error::new(
                    name.span(),
                    "expect #[online_config(skip)], #[online_config(hidden)] or #[online_config(submodule)]",
                ));
            }
        }
    }
    Ok((skip, hidden, submodule))
}
```

QCon+ 案例研习社

Rust 在 TiKV 项目上的实践

TiKV 在线配置更新

- ConfigValue
- Rust 过程宏

```rust
fn generate_token(ast: DeriveInput) -> std::result::Result<TokenStream, Error> {
    let name = &ast.ident;
    check_generics(&ast.generics, name.span())?;

    let crate_name = Ident::new("online_config", Span::call_site());
    let encoder_name = Ident::new(
        {
            // Avoid naming conflict
            let mut hasher = DefaultHasher::new();
            format!("{}", &name).hash(&mut hasher);
            format!("{}_encoder_{:x}", name, hasher.finish()).as_str()
        },
        Span::call_site(),
    );
    let encoder_lt = Lifetime::new("'lt", Span::call_site());

    let fields = get_struct_fields(ast.data, name.span())?;
    let update_fn = update(&fields, &crate_name)?;
    let diff_fn = diff(&fields, &crate_name)?;
    let get_encoder_fn = get_encoder(&encoder_name, &encoder_lt);
    let typed_fn = typed(&fields, &crate_name)?;
```

```rust
let encoder_struct = encoder(
    name,
    &crate_name,
    &encoder_name,
    &encoder_lt,
    ast.attrs,
    fields,
)?;

Ok(quote! {
    impl<#encoder_lt> #crate_name::OnlineConfig<#encoder_lt> for #name {
        type Encoder = #encoder_name<#encoder_lt>;
        #update_fn
        #diff_fn
        #get_encoder_fn
        #typed_fn
    }
    #encoder_struct
})
```

Rust 在 TiKV 项目上的实践

TiKV 在线配置更新

- ConfigValue
- Rust 过程宏

```rust
fn update(fields: &Punctuated<Field, Comma>, crate_name: &Ident) -> Result<TokenStream> {
    let incoming = Ident::new("incoming", Span::call_site());
    let mut update_fields = Vec::with_capacity(fields.len());
    for field in fields {
        let (skip, hidden, submodule) = get_config_attrs(&field.attrs)?;
        if skip || hidden || field.ident.is_none() {
            continue;
        }
        let name = field.ident.as_ref().unwrap();
        let name_lit = LitStr::new(&format!("{}", name), name.span());
        let f = if submodule {
            quote! {
                if let Some(#crate_name::ConfigValue::Module(v)) = #incoming.remove(#name_lit) {
                    #crate_name::OnlineConfig::update(&mut self.#name, v);
                }
            }
        } else if is_option_type(&field.ty) {
            quote! {
                if let Some(v) = #incoming.remove(#name_lit) {
                    if #crate_name::ConfigValue::None == v {
                        self.#name = None;
                    } else {
                        self.#name = Some(v.into());
                    }
                }
            }
```

```rust
        } else {
            quote! {
                if let Some(v) = #incoming.remove(#name_lit) {
                    self.#name = v.into();
                }
            }
        };
        update_fields.push(f);
    }
    Ok(quote! {
        fn update(&mut self, mut #incoming: #crate_name::ConfigChange) {
            #(#update_fields)*
        }
    })
```

Rust 在 TiKV 项目上的实践

TiKV 在线配置更新

- 使用

```rust
#[derive(Clone, Debug, Serialize, Deserialize, PartialEq, OnlineConfig)]
#[serde(default)]
#[serde(rename_all = "kebab-case")]
pub struct Config {
    pub max_batch_size: Option<usize>,
    pub pool_size: usize,
    #[online_config(skip)]
    pub reschedule_duration: ReadableDuration,
    #[online_config(skip)]
    pub low_priority_pool_size: usize,
}
```

```rust
impl ConfigManager for RaftstoreConfigManager {
    fn dispatch(
        &mut self,
        change: ConfigChange,
    ) -> std::result::Result<(), Box<dyn std::error::Error>> {
        {
            let change = change.clone();
            self.config
                .update(move |cfg: &mut Config| cfg.update(change));
        }
```

Rust 在 TiKV 项目上的实践

Rust 的类型系统的特点

- **参数多态**
  - 泛型数据结构
  - 泛型函数

```rust
pub fn send_extra_message<T: Transport>(
    &self,
    msg: ExtraMessage,
    trans: &mut T,
    to: &metapb::Peer,
) {
    let mut send_msg = self.prepare_raft_message();
    let ty = msg.get_type();
```

```rust
pub struct PollContext<EK, ER, T>
where
    EK: KvEngine,
    ER: RaftEngine,
{
    pub cfg: Config,
    pub store: metapb::Store,
    pub pd_scheduler: Scheduler<PdTask<EK, ER>>,
    pub consistency_check_scheduler: Scheduler<ConsistencyCheckTask<EK::Snapshot>>,
    pub split_check_scheduler: Scheduler<SplitCheckTask>,
```

Rust 在 TiKV 项目上的实践

Rust 的类型系统的特点

- **特设多态**
  - 类似于面向对象语言中的运算符重载
- **子类型多态**
  - 把子类型当成父类型使用
- **Trait**
  - “基本” trait，相当于 C 语言中在一个 struct 里定义一组函数指针
  - 在实现 trait 的时候，也可以用泛型参数来实现 trait，需要对泛型参数做一定的限制
  - 带关联类型的 trait
  - 支持泛型的 trait

Rust 在 TiKV 项目上的实践

Trait 在 TiKV 中的典型应用

- [engine_traits]

```rust
/// A TiKV key-value store
pub trait KvEngine:
    Peekable
    + SyncMutable
    + Iterable
    + WriteBatchExt
    + DBOptionsExt
    + CFNamesExt
    + CFOptionsExt
    + ImportExt
    + SstExt
    + CompactExt
    + RangePropertiesExt
    + MvccPropertiesExt
    + TtlPropertiesExt
    + TablePropertiesExt
    + PerfContextExt
    + MiscExt
    + Send
    + Sync
    + Clone
    + Debug
    + Unpin
    + 'static
{
    /// A consistent read-only snapshot of the database
    type Snapshot: Snapshot;

    /// Create a snapshot
    fn snapshot(&self) -> Self::Snapshot;

    /// Syncs any writes to disk
    fn sync(&self) -> Result<()>;
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- **Rust 的并发处理**
  - OS threads
  - Event driven
  - The actor module

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- Thread pool
  - yatp

```rust
pub struct ThreadPool<T: TaskCell + Send> {
    remote: Remote<T>,
    threads: Mutex<Vec<JoinHandle<()>>>,
}

impl<T: TaskCell + Send> ThreadPool<T> {
    /// Spawns the task into the thread pool.
    ///
    /// If the pool is shutdown, it becomes no-op.
    pub fn spawn(&self, t: impl WithExtras<T>) {
        self.remote.spawn(t);
    }

    /// Scale workers of the thread pool, the adjustable range is `min_thread_count`
    /// to `max_thread_count`, if this value exceeds `max_thread_count` or zero, it
    /// will be adjusted to `max_thread_count`, if this value is between zero and
    /// `min_thread_count`, it will be adjusted to `min_thread_count`.
    ///
    /// Notice:
    /// 1. The effect of scaling may be delayed, eg:
    ///     - Threads run long-term tasks, resulting in inability to scale down in
    ///       time, until they have no task to process and can sleep;
    ///     - Scaling up relies on spawning tasks to the thread pool to unpark new
    ///       threads, so the delay depends on the time interval between scale and
    ///       spawn;
    /// 2. If lots of tasks have been spawned before start, there may be more than
    ///     `core_thread_count` (max up to `max_thread_count`) threads running at a
    ///     moment until they can sleep (no tasks to run), then scheduled based on
    ///     `core_thread_count`;
    pub fn scale_workers(&self, new_thread_count: usize) {
        self.remote.scale_workers(new_thread_count);
    }
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- Thread pool
  - [yatp]

```rust
pub(crate) struct TaskInjector<T>(InjectorInner<T>);

enum InjectorInner<T> {
    SingleLevel(single_level::TaskInjector<T>),
    Multilevel(multilevel::TaskInjector<T>),
}

impl<T: TaskCell + Send> TaskInjector<T> {
    /// Pushes a task to the queue.
    pub fn push(&self, task_cell: T) {
        match &self.0 {
            InjectorInner::SingleLevel(q) => q.push(task_cell),
            InjectorInner::Multilevel(q) => q.push(task_cell),
        }
    }
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- Thread pool
  - [yatp]

```rust
/// The local queue of a task queue.
pub(crate) struct LocalQueue<T>(LocalQueueInner<T>);

enum LocalQueueInner<T> {
    SingleLevel(single_level::LocalQueue<T>),
    Multilevel(multilevel::LocalQueue<T>),
}

impl<T: TaskCell + Send> LocalQueue<T> {
    /// Pushes a task to the local queue.
    pub fn push(&mut self, task_cell: T) {
        match &mut self.0 {
            LocalQueueInner::SingleLevel(q) => q.push(task_cell),
            LocalQueueInner::Multilevel(q) => q.push(task_cell),
        }
    }

    /// Gets a task cell from the queue. Returns `None` if there is no task cell
    /// available.
    pub fn pop(&mut self) -> Option<Pop<T>> {
        match &mut self.0 {
            LocalQueueInner::SingleLevel(q) => q.pop(),
            LocalQueueInner::Multilevel(q) => q.pop(),
        }
    }
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- Thread pool
  - [yatp]

```rust
/// The core of queues.
///
/// Every thread pool instance should have one and only `QueueCore`. It's
/// saved in an `Arc` and shared between all worker threads and remote handles.
pub(crate) struct QueueCore<T> {
    global_queue: TaskInjector<T>,
    active_workers: AtomicUsize,
    config: SchedConfig,
}

impl<T> QueueCore<T> {
    pub fn new(global_queue: TaskInjector<T>, config: SchedConfig) -> QueueCore<T> {
        QueueCore {
            global_queue,
            active_workers: AtomicUsize::new(config.max_thread_count << WORKER_COUNT_SHIFT),
            config,
        }
    }
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- Thread pool
  - [yatp]

```rust
/// Ensures there are enough workers to handle pending tasks.
///
/// If the method is going to wake up any threads, source is used to trace who triggers
/// the action.
pub fn ensure_workers(&self, source: usize) {
    let cnt = self.active_workers.load(Ordering::SeqCst);
    if (cnt >> WORKER_COUNT_SHIFT) >= self.config.core_thread_count.load(Ordering::SeqCst)
        || is_shutdown(cnt)
    {
        return;
    }

    let addr = self as *const QueueCore<T> as usize;
    let mut unparked_once = false;

    unsafe {
        parking_lot_core::unpark_filter(
            addr,
            |p: ParkToken| {
                if !unparked_once && p.0 <= self.config.core_thread_count.load(Ordering::SeqCst)
                {
                    unparked_once = true;
                    FilterOp::Unpark
                } else {
                    FilterOp::Skip
                }
            },
            |_| UnparkToken(source),
        );
    }
}
```

QCon+ 案例研习社

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- Thread pool
  - [yatp]()

```rust
/// Marks current thread as woken up states.
pub fn mark_woken(&self) {
    let mut cnt = self.active_workers.load(Ordering::SeqCst);
    loop {
        match self.active_workers.compare_exchange_weak(
            cnt,
            cnt + WORKER_COUNT_BASE,
            Ordering::SeqCst,
            Ordering::SeqCst,
        ) {
            Ok(_) => return,
            Err(n) => cnt = n,
        }
    }
}
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- Thread pool
  - yatp

```rust
/// In the model of yatp, any piece of logic aiming to be executed in a thread
/// pool is called Task. There can be different definitions of Task. Some people
/// may choose `Future` as Task, some may just want callbacks, or even Actor
/// messages. But no matter what a Task is, there should be some role know how
/// to execute it. The role is call `Runner`.
///
/// The life cycle of a Runner is:
/// ```text
///   start
///     |
///     | <--- resume
///     |        |
///   handle -> pause
///     |
///    end
/// ```
///
/// Generally users should use the provided future thread pool or callback
/// thread pool instead. This is only for advance customization.
pub trait Runner {
    /// The local spawn that can be accepted to spawn tasks.
    type TaskCell;
```

```rust
pub fn run(mut self) {
    self.runner.start(&mut self.local);
    while !self.local.core().is_shutdown() {
        let task = match self.pop() {
            Some(t) => t,
            None => continue,
        };
        self.runner.handle(&mut self.local, task.task_cell);
    }
    self.runner.end(&mut self.local);

    // Drain all futures in the queue
    while self.local.pop().is_some() {}
}
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- The actor model
  - batch-system

```rust
/// A more high level mailbox.
pub struct Mailbox<Owner, Scheduler>
where
    Owner: Fsm,
    Scheduler: FsmScheduler<Fsm = Owner>,
{
    mailbox: BasicMailbox<Owner>,
    scheduler: Scheduler,
}

impl<Owner, Scheduler> Mailbox<Owner, Scheduler>
where
    Owner: Fsm,
    Scheduler: FsmScheduler<Fsm = Owner>,
{
    pub fn new(mailbox: BasicMailbox<Owner>, scheduler: Scheduler) -> Mailbox<Owner, Scheduler> {
        Mailbox { mailbox, scheduler }
    }

    /// Force sending a message despite channel capacity limit.
    #[inline]
    pub fn force_send(&self, msg: Owner::Message) -> Result<(), SendError<Owner::Message>> {
        self.mailbox.force_send(msg, &self.scheduler)
    }

    /// Try to send a message.
    #[inline]
    pub fn try_send(&self, msg: Owner::Message) -> Result<(), TrySendError<Owner::Message>> {
        self.mailbox.try_send(msg, &self.scheduler)
    }
}
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- The actor model
  - batch-system

```rust
/// Router route messages to its target mailbox.
///
/// Every fsm has a mailbox, hence it's necessary to have an address book
/// that can deliver messages to specified fsm, which is exact router.
///
/// In our abstract model, every batch system has two different kind of
/// fsms. First is normal fsm, which does the common work like peers in a
/// raftstore model or apply delegate in apply model. Second is control fsm,
/// which does some work that requires a global view of resources or creates
/// missing fsm for specified address. Normal fsm and control fsm can have
/// different scheduler, but this is not required.
pub struct Router<N: Fsm, C: Fsm, Ns, Cs> {
    normals: Arc<Mutex<NormalMailMap<N>>>,
    caches: Cell<LruCache<u64, BasicMailbox<N>>>,
    pub(super) control_box: BasicMailbox<C>,
    // TODO: These two schedulers should be unified as single one. However
    // it's not possible to write FsmScheduler<Fsm=C> + FsmScheduler<Fsm=N>
    // for now.
    pub(crate) normal_scheduler: Ns,
    pub(crate) control_scheduler: Cs,

    // Count of Mailboxes that is not destroyed.
    // Added when a Mailbox created, and subtracted it when a Mailbox destroyed.
    state_cnt: Arc<AtomicUsize>,
    // Indicates the router is shutdown down or not.
    shutdown: Arc<AtomicBool>,
}
```

Rust 在 TiKV 项目上的实践

TiKV 的并发处理

- The actor model
  - batch-system

```rust
pub trait PollHandler<N, C>: Send + 'static {
    /// This function is called at the very beginning of every round.
    fn begin<F>(&mut self, _batch_size: usize, update_cfg: F)
    where
        for<'a> F: FnOnce(&'a Config);

    /// This function is called when handling readiness for control FSM.
    ///
    /// If returned value is Some, then it represents a length of channel. This
    /// function will only be called for the same fsm after channel's lengh is
    /// larger than the value. If it returns None, then this function will
    /// still be called for the same FSM in the next loop unless the FSM is
    /// stopped.
    fn handle_control(&mut self, control: &mut C) -> Option<usize>;

    /// This function is called when handling readiness for normal FSM.
    ///
    /// The returned value is handled in the same way as `handle_control`.
    fn handle_normal(&mut self, normal: &mut impl DerefMut<Target = N>) -> HandleResult;

    /// This function is called after `handle_normal` is called for all fsm and before calling
    /// `end`. The function is expected to run lightweight work.
    fn light_end(&mut self, _batch: &mut [Option<impl DerefMut<Target = N>>]) {}

    /// This function is called at the end of every round.
    fn end(&mut self, batch: &mut [Option<impl DerefMut<Target = N>>]);

    /// This function is called when batch system is going to sleep.
    fn pause(&mut self) {}
```

总结

重点回顾

1. 通过自定义类型 + 过程宏，降低代码的冗余度
2. 通过 trait 抽象 engine 引擎接口，实现支持多种 engine
3. 自定义线程调度策略及 actor 模型

Learn by contributing.

THANKS

QCon+ 案例研习社