RustConAsia 2019 WorkShop

- Actix介绍
  - Actor模型
    - tick机制
    - 一个Actor指的是一个最基本的计算单元
      - 它能接收一个消息并且基于其执行计算
      - 面向对象
      - 每个actor都有一个固定地址
      - 每个actor都携带一个mailbox
      - 当收到一个消息的时候可以做三件事
        - 创建其他actor
        - 向其他actor发送消息
        - 指定下一条消息的行为 可以定义下条消息来到时的状态
    - 多个Actor对象组成一个计算系统，相互发送消息。完成目标
    - Actor运行于用户线程之上
      - Erlang的轻量级线程
      - Actix则是运行于futures task之上
      - 有助于一次性运行成千上万的actor
    - 容错
      - 如果用户线程崩溃，则会重启
      - supervisor 监视器
    - 天生分布式 基本不会在意消息是发往本地节点还是远程节点
    - 支持的语言
      - Erlang/Elixir
      - Scala akka
      - Ruby Celluloid
      - Rust
        - actix/actix
        - riker-rs/riker
      - Dart Isolates
      - Pony Reference Capabilities
  - Actix中的概念
    - actix特点
      - 异步/同步 actors
      - Actor 在本地/线程上下文中通信
      - 使用 Futures 进行异步消息处理
      - 支持 HTTP1/HTTP2(actix-web)
      - Actor 监控
      - 类型化消息 (No Any type)
    - actor
      - 实现trait Actor的任意类型
      - 包含行为状态
        - Started
        - Running
        - Stopping
        - Stopped
    - message
      - 实现trait Message的任意类型
      - 可作为actor中传递的消息
    - Mailbox
      - Mailbox<A: Actor>
        - 默认会开启一个channel
          - 非标准库的channel
          - 自己实现用于actor通信的channel
        - loop
          - 实现了poll
          - 与futures相似
    - Address
      - pub struct Addr<A: Actor> {
        tx: AddressSender<A>,
        }
      - 实现了send 可以发送异步消息
      - 实现了clone 可以clone多个channel tx
      - 携带一个默认线程安全的mpsc队列 实现了Send和Sync
      - 实现了一个channel 用于actor之间通信
      - 实现了一个message
        - request，发送消息
          ```
          pub struct Request<A, M>
          where
            A: Handler<M>,
            A::Context: ToEnvelope<A, M>,
            M: Message,
          {
            rx: Option<oneshot::Receiver<M::Result>>,
            info: Option<AddressSender<A>, M>,
            timeout: Option<Delay>,
            act: PhantomData<A>,
          }
          ```
        - 每个request是一个Future
          ```
          impl<A, M> Future for Request<A, M>
          where
            A: Handler<M>,
            A::Context: ToEnvelope<A, M>,
            M: Message + Send,
            M::Result: Send,
          ```
    - Handler
      - pub trait Handler<M>
        where
        Self: Actor,
        M: Message,
        {
        /// The type of value that this handler will return.
        type Result: MessageResponse<Self, M>;

        /// This method is called for every message received by this actor.
        fn handle(&mut self, msg: M, ctx: &mut Self::Context) -> Self::Result;
        }
      - 用于处理消息
      - 然后返回一个 实现了trait MessageResponse的MessageResult
        ```
        pub trait MessageResponse<A: Actor, M: Message> {
          fn handle<R: ResponseChannel<M>>(self, ctx: &mut A::Context, tx: Option<R>);
        }
        ```
      - 通过ResponseChannel在actor之间传送消息
      - 需要注意的类型
        - /// A specialized actor future for asynchronous message handling.
          pub type ResponseActFuture<A, I, E> = Box<ActorFuture<Item = I, Error = E, Actor = A>>;
        - /// A specialized future for asynchronous message handling.
          pub type ResponseFuture<I, E> = Box<Future<Item = I, Error = E>>;
        - /// Helper type for representing different type of message responses
          pub struct Response<I, E> {
          item: ResponseTypeItem<I, E>,
          }
    - Context
      - Actor执行所需的上下文信息
        ```
        pub fn run(self, act: A) -> Addr<A> {
            let fut = self.into_future(act);
            let addr = fut.address();
            actix_rt::spawn(fut);
            addr
        }
        ```
        - run方法，为Actor准备好一个执行环境
        - 每个actor可以转换为一个future
        - actix-rt
          - 在actix-net中的actor运行时库
          - 基于tokio和future
          - 实现spawn方法，将future task放到tokio的运行时中执行计算
      - 种类
        - 异步上下文
        - 同步上下文
    - Arbiters
      - 为actor提供异步计算的运行环境
      - 异步任务调度工具
        - 一个Arbiter控制一个带有一个事件池的线程。当Arbiter产生任务时（通过Arbiter :: spawn，Context <Actor> :: run_later或类似的构造），Arbiter将任务排队，以便在该任务队列上执行。当你想到Arbiter时，你可以想到“单线程事件循环”。
      - SyncArbiter
        - 能够在OS线程池上启动Actor的多个实例。
        - 只能托管单一类型的Actor
        - 对应SyncContext
    - Superviser trait Supervised: Actor fn restarting
    - System System::new
      - 为Actor创建一个Arbiters
      - actor.start方法会让actor在System Arbiters的线程内部运行
- Actix-Web介绍
  - Actix-Web 0.7 State
  - Actix-Web 1.0 进行了大规模重构
    - 组件化
    - 架构没变
    - State变为了Data
    - 将常用的模块归类到了web模块中
    - 引入了service
    - 差异
- Diesel介绍
  - http://diesel.rs/guides/getting-started/
- Actix-Web实现一个Restful API
  - ToDo List 功能介绍
  - 代码V1.0实现
  - 代码重构
  - 打包docker

Trial Version XMind:ZEN Trial Version