Rust Web框架比较

@suoyuan
Westar实验室

大纲

- 背景
- Rust常见Web框架
- 主流Web框架的比较
- Rocket分析

背景

Web框架

- Web框架
  - 接受请求，提供校验能力
  - 从底层获取数据
  - 按照特定格式结果
- 其他语言现状
  - Laravel（PHP）
  - Spring MVC（JAVA）
  - Gin/Beego（Golang）

Rust常见Web框架

Rust Web框架

- Rust Web框架的难点
  - 全局状态
  - 生命周期
  - 编译慢

Rust常见Web框架

- Actix-web、Gotham、Iron
- Rocket、warp、Tower-web
- hyper、tiny-http

Rust主流Web框架比较

Hyper的优点

- Client/Server module
- High Concurrency

Hyper的缺点

- 应用侧的功能相对其他框架少
- 通过match block实现路由

Hyper的路由实现

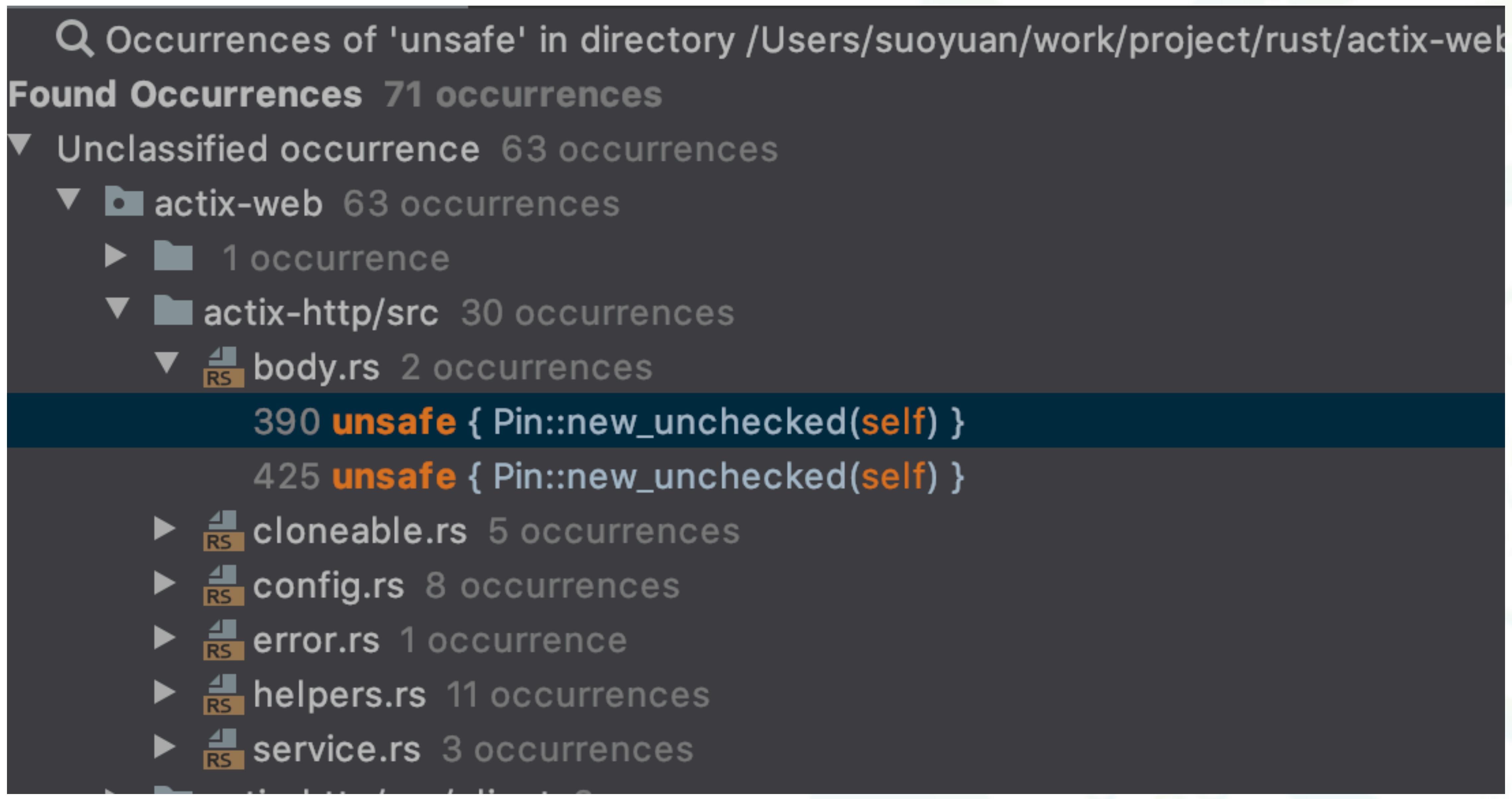
```
async fn response_examples(
    req: Request<Body>,client: Client<HttpConnector>
) -> Result<Response<Body>> {
    match (req.method(), req.uri().path()) {
        (&Method::GET, "/index.html") => {
            Ok(Response::new(INDEX.into()))
        },
       (&Method::GET, "/json_api") => {
            api_get_response().await
        }
        _ => {
   Ok(Response::builder().status(StatusCode::NOT_FOUND)
                .body(NOTFOUND.into())
                .unwrap())
        }
    }
}
```

Actix-web

- Actor model
- Blazingly Fast
- Feature Rich

Actix-web

- Actix-web 缺点
  - 大量unsafe
  - 文档实例不全



```
Occurrences of 'unsafe' in directory /Users/suoyuan/work/project/rust/actix-web
Found Occurrences  71 occurrences
▼ Unclassified occurrence  63 occurrences
  ▼ actix-web  63 occurrences
    ▶  1 occurrence
    ▼ actix-http/src  30 occurrences
      ▼ body.rs  2 occurrences
          390 unsafe { Pin::new_unchecked(self) }
          425 unsafe { Pin::new_unchecked(self) }
      ▶ cloneable.rs  5 occurrences
      ▶ config.rs  8 occurrences
      ▶ error.rs  1 occurrence
      ▶ helpers.rs  11 occurrences
      ▶ service.rs  3 occurrences
```

Rocket

Rocket

- Easy to Use
- Boilerplate Free
- Extensible

Rust主流Web框架比较

框架	Rocket	Actix-web	Hyper
Github Stars	8.4K	5.9K	5.5K
Https support	Yes	Yes	Yes
Http2 support	Yes	Yes	Yes
Websocket	No	Yes	No
base framework	hyper	tokio	tokio
async	支持	支持	支持
周边支持	好	一般	一般

Rocket

Rocket设计原则

- Security, correctness, and developer experience are paramount.
- All request handling information should be typed and self-contained
- Decisions should not be forced

Rocket RequestGuards

- RequestGuards是代表任意验证策略的类型，验证策略通过FromRequest实现
- RequestGuards没有数量限制
- 可以自定义Guards

```
#[get("/<param>")]
fn index(param: isize, a: A, b: B, c: C) -> ... { ... }
```

Rocket Responder定义

```
pub trait Responder {
    /// Returns `Ok` if a `Response` could be generated successfully. Otherwise,
    /// returns an `Err` with a failing `Status`.
    ///
    /// The `request` parameter is the `Request` that this `Responder` is
    /// responding to.
    ///
    /// When using Rocket's code generation, if an `Ok(Response)` is returned,
    /// the response will be written out to the client. If an `Err(Status)` is
    /// returned, the error catcher for the given status is retrieved and called
    /// to generate a final error response, which is then written out to the
    /// client.
    fn respond_to(self, request: &Request) -> response::Result;
}
```

rocket native support : str 、String、 [u8] 、 File、 Option、 Status

Rocket 自定义的Responder

```
impl Responder for Person {
    fn respond_to(self, _: &Request) -> response::Result {
        Response::build()
             .sized_body(Cursor::new(format!("{}:{}", self.name, self.age)))
             .raw_header("X-Person-Name", self.name)
            .header(ContentType::new("application", "x-person"))
            .ok()
      }
 }
#[get("/person")]
 fn person() -> Person { Person { name: "a".to_string(), age: 20 } }
```

Actix-web responder定义

```
pub trait Responder {
    /// The associated error which can be returned.
    type Error: Into<Error>;

    /// The future response value.
    type Future: Future<Output = Result<Response, Self::Error>>;

    /// Convert itself to `AsyncResult` or `Error`.
    fn respond_to(self, req: &HttpRequest) -> Self::Future;


    fn with_status(self, status: StatusCode) -> CustomResponder<Self>
        where
            Self: Sized,
    {
        CustomResponder::new(self).with_status(status)
    }


    fn with_header<K, V>(self, key: K, value: V) -> CustomResponder<Self>
        … …
}
```

Rocket State

```
use rocket::State;
use rocket::response::content;

struct HitCount(AtomicUsize);

#[get("/")]
fn index(hit_count: State<HitCount>) -> content::Html<String> {
    hit_count.0.fetch_add(1, Ordering::Relaxed);
    let msg = "Your visit has been recorded!";
    let count = format!("Visits: {}", count(hit_count));
    content::Html(format!("{}<br /><br />{}", msg, count))
}

#[get("/count")]
fn count(hit_count: State<HitCount>) -> String {
    hit_count.0.load(Ordering::Relaxed).to_string()
}

fn rocket() -> rocket::Rocket {
    rocket::ignite()
        .mount("/", routes![index, count])
        .manage(HitCount(AtomicUsize::new(0)))
}
```

Rocket Fairing 功能

- on_attach
- on_launch
- on_request
- on_response

Rocket Fairing 实例

```
#[derive(Default)]
struct Counter {
    get: AtomicUsize,
    post: AtomicUsize,
}
impl Fairing for Counter {

    fn on_request(&self, request: &mut Request, _: &Data) {
   … …}

    fn on_response(&self, request: Request, response: mut Response)
       {… …}
}
fn rocket() -> rocket::Rocket {
    rocket::ignite()
        .mount("/", routes![… , …])
        .attach(Counter::default())
           … …
}
```

Rocket Fairing

- 挂载在Request的生命周期
- 与其他框架中间件差异：
  - 不能终止或者直接响应Request
  - 不能将非请求数据注入Request
  - 可以检查或者修改配置
- 只有attach了才能触发
- 顺序很重要
- 使用AdHoc快速实现

性能测试结果

参数	actix-web	rocket(async)	rocket(sync)	hyper(debug)	hyper(release)
qps	143281	113878	21117	69303	86202
Concurrency	100	100	50	100	100
Transfer rate（kps）	21408	18905	3010	7580	9428
Time per request（ms)	0.698	0.878	2.368	1.443	1.160

Thanks