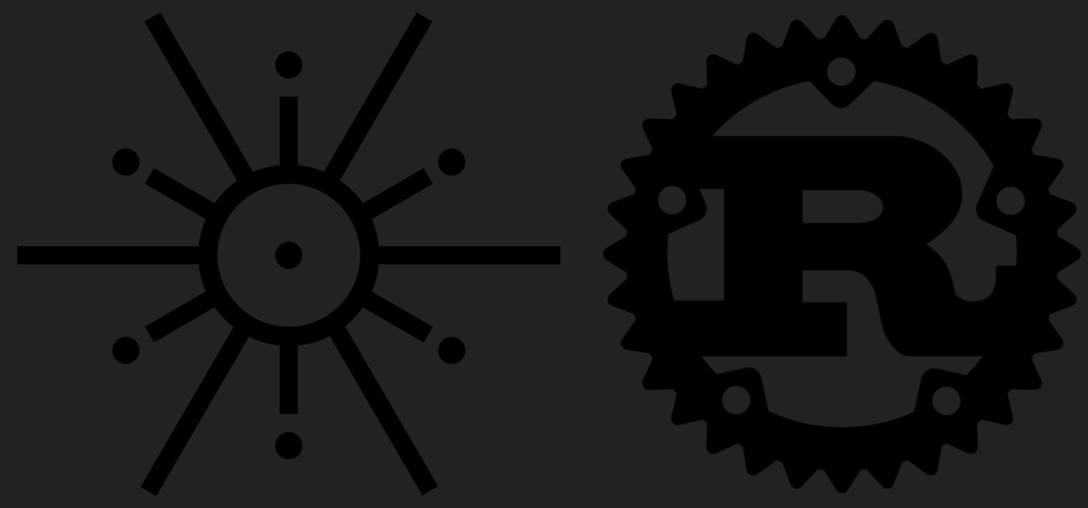

ГЛЕБ ПОМЫКАЛОВ,
GLEB@LANCASTR.COM

МИГРИРУЕМ НА TOKIO 0.2

std::future

async/await

Futures 0.3

Tokio 0.2

Futures 0.1	Futures 0.3
Future	std::future::Future
Stream	Опционально
Sink	Опционально

ASYNC/AWAIT

```
async fn get_data() ->
  Result<serde_json::Value, reqwest::Error>
{
  reqwest::Client::new()
    .post("https://jsonplaceholder.typicode.com/posts")
    .json(&serde_json::json!({
        "title": "Reqwest.rs",
        "body": "https://docs.rs/reqwest",
        "userId": 1
    }))
    .send()
    .await?
    .json()
    .await
}
```

ДОЖИДАЕМСЯ ЗАГОЛОВКОВ ОТВЕТА

ПОЛУЧАЕМ ОБЫЧНЫЙ RESULT

ДОЖИДАЕМСЯ ТЕЛА ОТВЕТА

ИЗМЕНЕНИЯ В TRAIT FUTURE

```
trait Future {
    type Item;
    type Error;

    fn poll(&mut self)-> Poll<Self::Item, Self::Error>;
}
```

```
trait Future {
    type Output;

    fn poll(self: Pin<&mut Self>, cx: &mut
Context<'_>) -> Poll<Self::Output>;
}
```

ОТСУТСТВУЕТ ERROR

СТАТИЧЕСКИЙ МЕТОД

ПОЯВИЛСЯ PIN

ПОЯВИЛСЯ CONTEXT (В НЕМ ТОЛЬКО WAKER)

ТРАНСФОРМАЦИИ ASYNC

```
async {
    let mut x = [0; 128];
    let read_into_buf_fut = read_into_buf(&mut x);
    read_into_buf_fut.await;
    println!("{:?}", x);
}
```

```
struct ReadIntoBuf<'a> {
    buf: &'a mut [u8],
}

struct AsyncFuture {
    x: [u8; 128],
    read_into_buf_fut: ReadIntoBuf<'self>,
}
```

SELF-REFERENTIAL STRUCTURE: ВАЛИДНО ТОЛЬКО ДЛЯ UNPIN

ПИНИРОВАНИЕ

Heap: `Box::pin(fut)`

Stack: `pin_mut!(fut)`

```
let foo =
   some_future();
pin_mut!(foo);
async {
   foo.await;
}
```

Variable Shadowing

```
let foo = some_future();
let mut foo = foo;
let mut foo = unsafe {
   Pin::new_unchecked(&mut foo)
};
async {
   foo.await;
}
```

He Unpin

unsafe!

Unpin

ТИП PIN

`Pin<P>` – A pinned pointer

`P: Deref`

```
Pin::new(ptr: P)
where Target: Unpin
```

```
unsafe {
   Pin::new_unchecked(ptr: P)
}
```

ОГРАНИЧЕНИЯ PIN

`Pin<Box<T>>`

Owned pointer to a pinned T

`Pin<&mut T>`  `&mut T`

Для получения доступа к полям структуры используются проекции

НАПИСАНИЕ СВОИХ FUTURE

PIN-PROJECT = “0.4.8” ИЛИ PIN-PROJECT-LITE = "0.1.4"

```
#[pin_project]
struct MyFuture<F> where F: Future<Output=()> {
    #[pin]
    inner: F,
    started_at: Option<Instant>,
}

impl<F> Future for MyFuture<F> where F: Future<Output=()> {
    type Output = ();

    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) ->
Poll<Self::Output> {
        let this = self.project();
        let started_at = this.started_at;
        if started_at.is_none() {
            *started_at = Some(Instant::now());
        };
        ready!(this.inner.poll(cx));
        let passed = Instant::now() - started_at.unwrap();
        println!("passed {:?}", passed);
        Poll::Ready(())
    }
}
```

Хотим получить Pin<&mut T>

Хотим получить &mut T

КОМБИНАТОРЫ

Два типа

Обычные

```
rx
   .map(|v| v * 2)
   .collect()
   .await
```

Похоже на Iterator

Result

```
future
   .map_ok(|x| x + 3)
   .and_then(|x| async move {
     Ok::<i32, i32>(x - 5)
   })
   .await?;
```

Похоже на Future 0.1

МАКРОС SELECT!

```
async {
    let mut timeout = delay_for(duration).fuse();
    let mut req = call_http().fuse();

    select! {
        response = req => {
            println!("Response is: {:?}", response);
        },
        _ = timeout => {
            println!("Timeout");
        },
    }
}
```

ASYNC FN В TRAIT

НЕ ПОДДЕРЖИВАЕТСЯ

ASYNC-TRAIT = "0.1.24"

```
use async_trait::async_trait;

#[async_trait]
trait Advertisement {
    async fn run(&self);
}

struct Modal;

#[async_trait]
impl Advertisement for Modal {
    async fn run(&self) {
        …
    }
}
```

-> Box<dyn Future>

ИЗМЕНЕНИЯ В TOKIO

- Используются features почти для всех частей библиотеки
- Увеличена скорость многопоточного scheduler'а
- Stream и Sink - опционален
- Появились async синхронизационные примитивы: Mutex, RwLock, Barrier
- Broadcast и Watch

ПРИМЕР

Удобно создаем Runtime

Используем функцию accept() вместо стрима

Комбинаторы не нужны - используем как итераторы

```
#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let addr = env::args()
        .nth(1)
        .unwrap_or_else(|| "127.0.0.1:8080".to_string());

    let mut listener = TcpListener::bind(&addr).await?;
    println!("Listening on: {}", addr);

    loop {
        let (socket, _) = listener.accept().await?;

        tokio::spawn(async move {
            let mut framed = BytesCodec::new().framed(socket);

            while let Some(message) = framed.next().await {
                match message {
                    Ok(bytes) => println!("bytes: {:?}", bytes),
                    Err(err) => println!("Socket closed with error: {:?}", err),
                }
            }
            println!("Socket received FIN packet and closed connection");
        });
    }
}
```

МОЙ ОПЫТ

- Комбинаторы нужны редко
- Свои Future, Stream, Sink нужно писать очень редко
- Код после миграции стал качественнее
- Писать асинхронный код приятно
- Редко возникают не понятные ошибки

ПОЛНОЦЕННЫЙ RUST, НО ПИСАТЬ ПОЧТИ ТАК ЖЕ ПРОСТО, КАК НА GO

СПАСИБО!

ИСПОЛЬЗУЙТЕ RUST