Peter the Great
Saint-Petersburg Polytechnic University

Конкурентность и параллелизм в Rust

Нгуен Тхи Хань Хуен
Королева Дарья

Введение

Rust полностью поддерживает параллелизм с использованием потоков ОС, мьютексов и каналов.

Основная идея — «бесстрашный параллелизм» (fearless concurrency).

Потоки в Rust

Потоки создаются через `thread::spawn()`, возвращает `JoinHandle<T>`.

Для передачи владения — ключевое слово `move`.

```rust
fn main() {
    let handle = thread::spawn(move || {
        println!("Привет из дочернего потока!");
        42
    });

    let result = handle.join().unwrap();
    println("Результат: {}", 42);
}
```

Ошибки и поведение потоков

Поток может завершиться с ошибкой (`panic!`)

`join()` возвращает `Result`, который можно обработать

Потоки в Rust — нативные, а не «виртуальные»

Каналы (Channels)

`mpsc::channel()` — механизм обмена сообщениями.

mpsc = Multi-Producer, Single-Consumer.

Отправитель — `Sender<T>`

Получатель — `Receiver<T>`

```rust
use std::sync::mpsc;
use std::thread;

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        tx.send("Привет из потока!").unwrap();
    });

    let msg = rx.recv().unwrap();
    println!("Получено: {}", msg);
}
```

Несвязанные каналы (Unbounded / Asynchronous)

`mpsc::channel()` — асинхронный, **несвязанный** без ограничений

```rust
use std::thread;
use std::time::Duration;

fn main() {
    let (tx, rx) = mpsc::channel();

    thread::spawn(move || {
        let thread_id = thread::current().id();
        for i in 1..10 {
            tx.send(format!("Сообщение {i}")).unwrap();
            println!("{thread_id:?}: отправил сообщение {i}");
        }
        println!("{thread_id:?}: готово");
    });

    thread::sleep(Duration::from_millis(100));

    for msg in rx.iter() {
        println!("Основной поток: получено {msg}");
    }
}
```

Связанные каналы (Bounded / Synchronous)

mpsc::sync_channel(n) создаёт канал фиксированного размера n.

- Вызов send() **блокирует** поток, если очередь канала заполнена.
- Сообщение будет отправлено только тогда, когда в канале появится место.

```rust
use std::sync::mpsc;
use std::thread;
use std::time::Duration;

fn main() {
    let (tx, rx) = mpsc::sync_channel(3);

    thread::spawn(move || {
        let thread_id = thread::current().id();
        for i in 1..10 {
            tx.send(format!("Сообщение {i}")).unwrap();
            println!("{thread_id:?}: отправил сообщение {i}");
        }
        println!("{thread_id:?}: готово");
    });

    thread::sleep(Duration::from_millis(100));

    for msg in rx.iter() {
        println!("Основной поток: получено {msg}");
    }
}
```

Совместное состояние (Shared State)

Потоки могут делить состояние через `Arc<Mutex<T>>`.

- `Arc` — разделяемое владение (Atomic Reference Count).
- `Mutex` — взаимное исключение (Mutual Exclusion).

lock() блокирует поток до освобождения ресурса.

Guard автоматически освобождает Mutex.

```rust
use std::sync::{Arc, Mutex};
use std::thread;

fn main() {
    let counter = Arc::new(Mutex::new(0));
    let mut handles = vec![];

    for _ in 0..10 {
        let counter = Arc::clone(&counter);
        handles.push(thread::spawn(move || {
            let mut num = counter.lock().unwrap();
            *num += 1;
        }));
    }

    for h in handles { h.join().unwrap(); }
    println!("Result: {}", *counter.lock().unwrap());
}
```

Трейты безопасности: Send и Sync

Rust гарантирует безопасность через маркеры `Send` и `Sync`.

- `Send` → тип можно передавать между потоками.
- `Sync` → тип можно делить ссылкой `&T`.

Примеры:

`i32`, `Vec<T>` — Send

`Rc<T>` — не Send
`Arc<T>` — Send + Sync

Пример 1: Send

```rust
use std::thread;

fn main() {
    let v = vec![1, 2, 3];

    // Vec<i32> реализует Send → можно передать в поток
    let handle = thread::spawn(move || {
        println!("Дочерний поток: {:?}", v);
    });

    handle.join().unwrap();
}
```

Пример 2: Rc<T> не является Send

```rust
use std::rc::Rc;
use std::thread;

fn main() {
    let rc = Rc::new(5);

    // Ошибка компиляции: Rc<i32> не реализует Send
    let handle = thread::spawn(move || {
        println!("Дочерний поток: {}", rc);
    });

    handle.join().unwrap();
}
```

Пример 3: Arc<T> является Send + Sync

```rust
use std::sync::Arc;
use std::thread;

fn main() {
    let a = Arc::new(5);
    let a2 = Arc::clone(&a);

    let handle = thread::spawn(move || {
        println!("Дочерний поток: {}", a2);
    });

    println!("Основной поток: {}", a);
    handle.join().unwrap();
}
```

Преимущества и ограничения

Преимущества:

- Безопасность — никаких гонок данных.
- Высокая производительность — zero-cost абстракции.
- Гибкость — потоки, async, каналы.

Ограничения:

- Возможны deadlock’и.
- Экосистема async требует выбора runtime (Tokio, async-std).

Список литературы и источников

The Rust Programming Language (официальная книга, «The Rust Book»)

URL: https://doc.rust-lang.org/book/

Rust Standard Library — std::thread, std::sync

URL: https://doc.rust-lang.org/std/