# Методы асинхронного программирования

*Александр Рулёв*

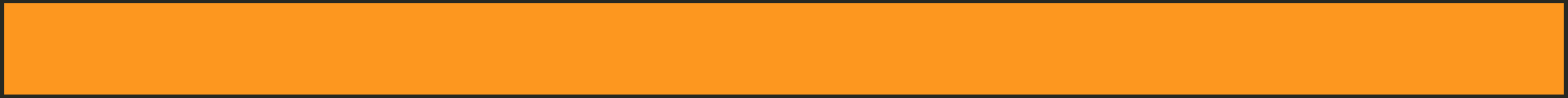
goo.gl/sfBQuW

I / O

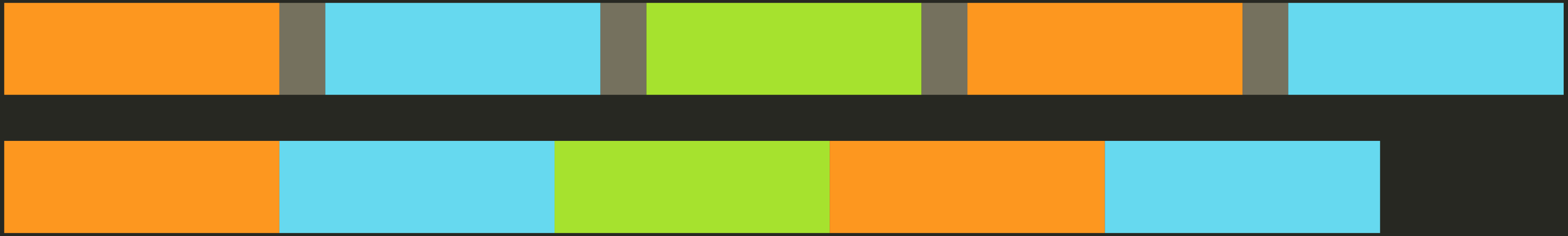
```java
byte[] dataBuffer = new byte[1024];
byte[] resultBuffer = new byte[1024];

// ...

int count = stream.read(dataBuffer);

// ... process data

stream.write(resultBuffer, 0, resultCount);

// ...
```

```java
class MyThread extends Thread {
  // ...
  public void run() {
    // ...
    int count = this.inputStream.read(this.dataBuffer);
    // ...
    this.outputStream.write(this.resultBuffer, 0, resultCount);
    // ...
  }
}

// ...
Thread thread = new MyThread(input, output);
thread.start()
// ...
```

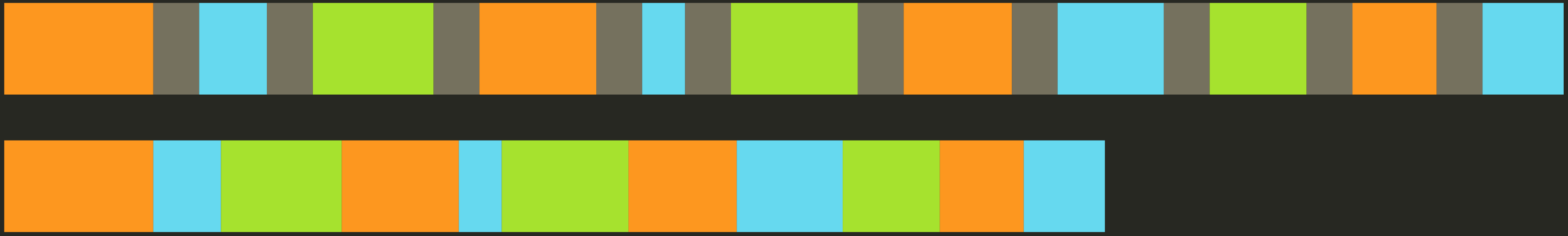
1 поток:



3 потока:



3 потока, короткие задачи:



```java
while (true) {
  for (ReadTask task : readTasks) {
    int count = inputStream.read(
      task.buffer,
      task.bufferOffset
    );

    task.bufferOffset += count;

    if (task.bufferOffset >= task.requiredCount) {
      task.notify();
      tasks.remove(task);
    }
  }

  // ...
}
```

```java
SelectionKey key1 = socket1.channel.keyFor(selector);
key1.interestOps(key1.interestOps() | SelectionKey.OP_READ);

SelectionKey key2 = socket2.channel.keyFor(selector);
key2.interestOps(key2.interestOps() | SelectionKey.OP_WRITE);

selector.select(); // Blocks

for (SelectionKey key : selector.selectedKeys()) {
    // Read/Write
}
```

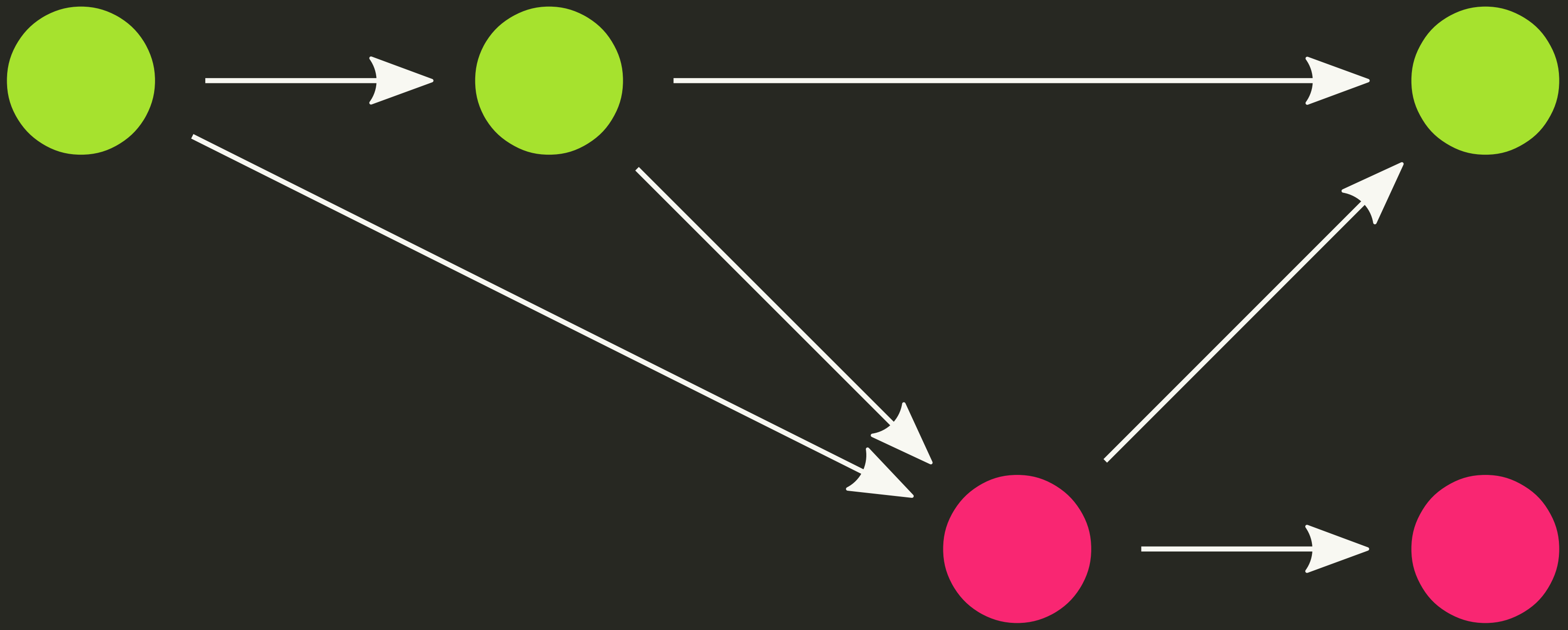
UI

```
getInput('Your name is: ', function(name) {
    getInput('Your favourite book is: ', function(book) {
        print(`Hi, ${name}! ${book} is really good.`);
    });
});
```

```
getFirstValue(function(error, firstValue) {
    if (error) { handleError(error); return; }

    getSecondValue(firstValue, function(error, secondValue) {
        if (error) { handleError(error); return; }

        getThirdValue(secondValue, function(error, thirdValue) {
            if (error) { handleError(error); return; }

            doSomething(thirdValue);
        });
    });
});
```

```
  (Обещание А)
и (функция из А в (Обещание Б))

→ (Обещание Б)
```

```
      (Обещание (А + Е))
и         (А → (Обещание (Б + Е2)))
и/или (Е → (Обещание (Б + Е2)))

→ (Обещание (Б + Е2))
```



```
function MyPromise(executor) {
    let isResolved = false;
    let value;

    let waiters = [];

    executor(function(result) {
        if (result && (typeof result.then === 'function')) {
            result.then(resolve);
        } else {
            resolve(result);
        }

        function resolve(result) {
            isResolved = true;
            value = result;

            waiters.forEach(f => f(value));
            waiters = null;
        }
    });

    this.then = function(onSuccess) {
        if (isResolved) {
            return new MyPromise(function(nextOnSuccess) {
                nextOnSuccess(onSuccess(value));
            });
        }

        return new MyPromise(function(nextOnSuccess) {
            waiters.push(function(value) {
                nextOnSuccess(onSuccess(value));
            });
        });
    };
}
```

```
getFirstValue()
  .then(function(firstValue) {
    return getSecondValue(firstValue);
  })
  .then(function(secondValue) {
    return getThirdValue(secondValue);
  })
  .then(function(thirdValue) {
    return doSomething(thirdValue);
  }, function(error) {
    return handleError(error);
  });
```

## async / await

```
async function justDoIt() {
    let firstValue = await getFirstValue();
    let secondValue = await getSecondValue(firstValue);
    let thirdValue = await getThirdValue(secondValue);

    return doSomething(thirdValue);
}
```

## Корутины

```javascript
function* threeNumbers() {
    console.log('start');
    yield 1;
    console.log('1-2');
    yield 2;
    console.log('2-3');
    return 3;
}

let gen = threeNumbers();

gen.next(); // > 'start'
            // {value: 1, done: false}
gen.next(); // > '1-2'
            // {value: 2, done: false}
gen.next(); // > '2-3'
            // {value: 3, done: true}
```

```javascript
function* sumCalculator() {
  let total = 0;

  while (true) {
    let x = yield total;
    total += x;
  }
}

let gen = sumCalculator();

gen.next();

gen.next(17);
gen.next(19);

console.log(gen.next(6).value);  // 42
```

```
function* sumThenMultiply(initial) {
    let number = initial;

    let add = yield;
    number += add;

    let multiply = yield;
    number *= multiply;

    return number;
}
```

```javascript
function sumThenMultiply(initial) {
    let number = initial;

    let step = 1;

    return { next: function(input) {
        switch (step) {
        case 1:
            step = 2;
            return { value: undefined, done: false };
        case 2:
            number += input;
            step = 3;
            return { value: undefined, done: false };
        case 3:
            number *= input;
            step = 4;
            return { value: number, done: true };
        case 4:
            return { value: undefined, done: true };
        }
    } };
}
```

```
function* justDoIt() {
    let firstValue = yield getFirstValue();
    let secondValue = yield getSecondValue(firstValue);
    let thirdValue = yield getThirdValue(secondValue);

    return doSomething(thirdValue);
}
```

```
function executeAsyncGenerator(gen) {
  let val;

  return new Promise(function(resolve, reject) {
    genNext(false);

    loop(val.value);

    function loop(lastValue) {
      while (!val.done) {
        if (lastValue && (typeof lastValue.then === 'function')) {
          lastValue.then(genNext.bind(null, true), genThrow.bind(null, true));

          return;
        } else {
          genNext(false, lastValue);
        }
      }

      resolve(lastValue);
    }

    function genNext(callLoop, value) {
      try { val = gen.next(value); } catch (e) { reject(e); return; }
      if (callLoop) loop(val.value);
    }
    function genThrow(callLoop, value) {
      try { val = gen.throw(value); } catch (e) { reject(e); return; }
      if (callLoop) loop(val.value);
    }
  });
}
```

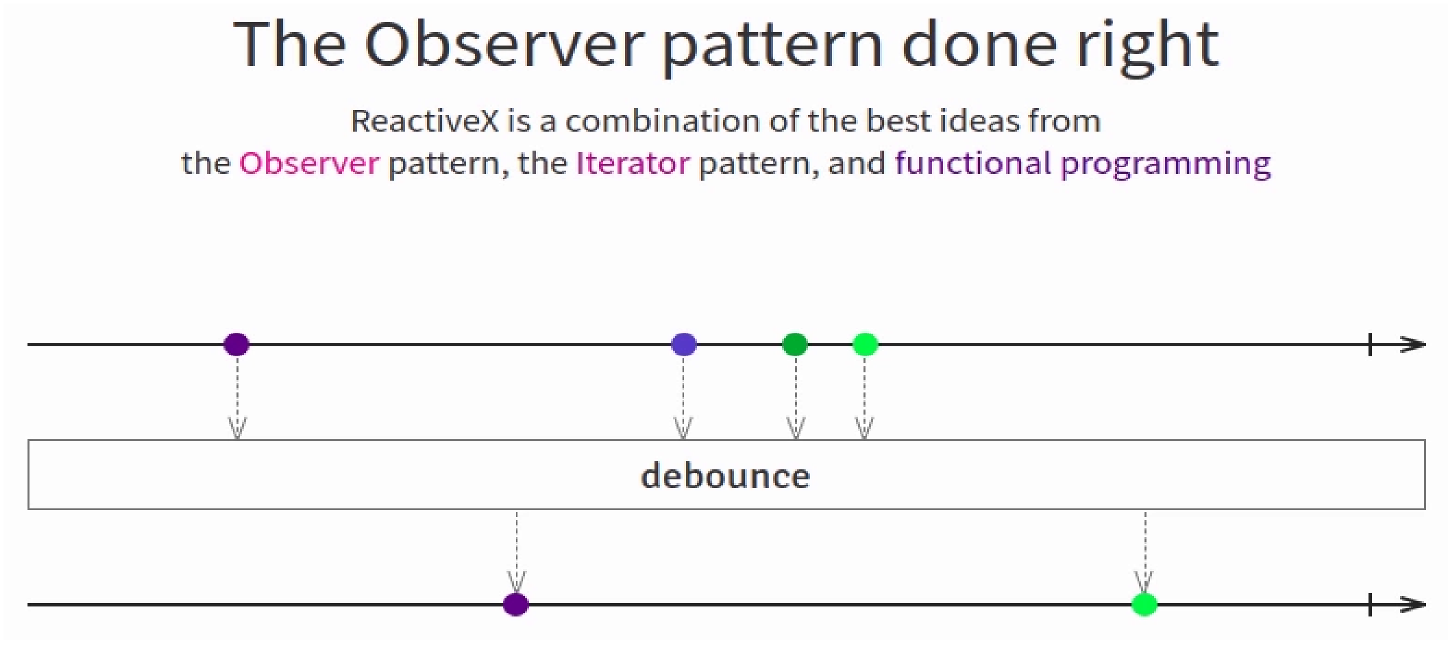
```
async function justDoIt() {
    let firstValue = await getFirstValue();
    let secondValue = await getSecondValue(firstValue);
    let thirdValue = await getThirdValue(secondValue);

    return doSomething(thirdValue);
}
```

## Асинхронные генераторы

```python
import asyncio

async def slowSequence():
    i = 1

    while True:
        await asyncio.sleep(1)
        yield i
        i += 1
```

```
  файл
→ поток блоков байт
→ utf-8 строки
→ фильтрация
→ вывод
```

# ReactiveX



The Observer pattern done right

ReactiveX is a combination of the best ideas from
the Observer pattern, the Iterator pattern, and functional programming

## Отмена выполнения асинхронных операций

```javascript
let timeoutId = setTimeout(function() {
    console.log('setTimeout fired');
}, 1000);

// ...

clearTimeout(timeoutId);
```

```javascript
let request = new XMLHttpRequest();

request.addEventListener('load', function() {
    // ...
});

// ...

request.abort();
```

```
const controller = new AbortController();
const signal = controller.signal;

setTimeout(() => controller.abort(), 5000);

fetch(url, { signal }).then(response => {
  return response.text();
}).then(text => {
  console.log(text);
});
```

```javascript
let computation = doSomeAsync().result(function(data) {
    let requestParams = f(data);
    return doRequest(requestParams);
});

let running = computation.run();

// ...

running.cancel().run();
```

## Альтернативы

## Goroutines

```go
func numbers() {
    for i := 1; i <= 5; i++ {
        time.Sleep(250 * time.Millisecond)
        fmt.Printf("%d ", i)
    }
}
func alphabets() {
    for i := 'a'; i <= 'e'; i++ {
        time.Sleep(400 * time.Millisecond)
        fmt.Printf("%c ", i)
    }
}
func main() {
    go numbers()
    go alphabets()
    time.Sleep(3000 * time.Millisecond)
    fmt.Println("main terminated")
}
```

## Акторы

## Актор

Принимает сообщения и реагирует на них:

- Отправлением сообщений другим акторам
- Созданием новых акторов
- Изменением своего внутреннего состояния, влияющего на обработку последующих сообщений

```erlang
ping(0, Pong_PID) ->
    Pong_PID ! finished,
    io:format("ping finished~n", []);

ping(N, Pong_PID) ->
    Pong_PID ! {ping, self()},
    receive
        pong ->
            io:format("Ping received pong~n", [])
    end,
    ping(N - 1, Pong_PID).

pong() ->
    receive
        finished ->
            io:format("Pong finished~n", []);
        {ping, Ping_PID} ->
            io:format("Pong received ping~n", []),
            Ping_PID ! pong,
            pong()
    end.

start() ->
    Pong_PID = spawn(tut15, pong, []),
    spawn(tut15, ping, [3, Pong_PID]).
```

## Вопросы?

Контакты: ruliov.hypershape.club