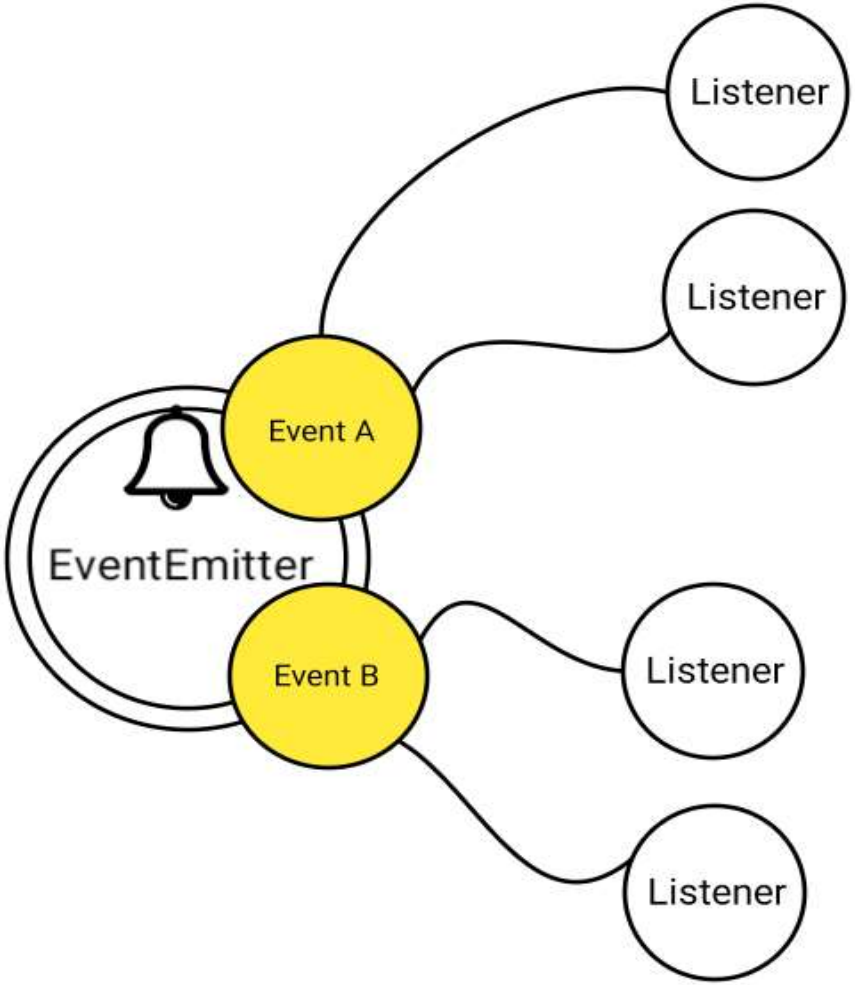
ПРОГРАММИРОВАНИЕ СЕРВЕРНЫХ КРОССПЛАТФОРМЕННЫХ ПРИЛОЖЕНИЙ

EVENT EMITTER. ТАЙМЕРЫ

Подавляющее большинство функционала Node.js применяет асинхронную событийную архитектуру, которая использует специальные объекты для генерации различных событий, которые обрабатываются специальными функциями – обработчиками или слушателями событий.

Все объекты, которые генерируют события, представляют экземпляры класса EventEmitter.



EVENT EMITTER =

JS-класс, предоставляющий функциональность для обработки событий в Node.js.

Событие программного объекта =

процесс перехода объекта из одного состояние в другое. При этом, об этом переходе могут быть извещены другие объекты. У события есть издатель (или генератор) события и могут быть подписчики (или обработчики) события.

EVENT EMITTER

Примеры подписки на события

```
//   --- EventEmitter ---

//   request.on('data', ....)

//   response.end(html);          // = response.on('end'...)

// server.on('request',  ....

// process.stdin.on('readable',
```

Производный от Event Emitter объект приобретает функциональность, позволяющую генерировать и прослушивать события

```
const EventEmitter = require('events');

class DB extends EventEmitter {
    // реализация
}
```

Для генерации событий предназначена функция emit(), а для прослушивания – функция on().

Пример использования Event Emitter

```
// отдельный модуль

const EventEmitter = require('events');

let db_data = [      // имитация базы данных
    { id: 1, name: "Иванов И.И", bday: '2001-01-01' },
    { id: 2, name: "Петров П.П", bday: '2001-01-02' },
    { id: 3, name: "Сидоров С.С", bday: '2001-01-03' },
];

class DB extends EventEmitter {
    get() { return db_data; };                  // реализация GET
    post(object) { db_data.push(object); };     // реализация POST
}

exports.DB = DB;     // экспортируется класс DB
```

Пример использования Event Emitter

```
const http = require('http');
const url = require('url');
const fs = require('fs');
let data = require('./DB.js');      // DB-объект, производный от EventEmitter

let db = new data.DB();
// --- слушатели событий ---------------------
db.on('GET', (request, response) => {
    console.log('DB.GET');
    response.end(JSON.stringify(db.get()));
});

db.on('POST', (request, response) => {
    console.log('DB.POST');
    request.on('data', data => {
        let r = JSON.parse(data);
        db.post(r);
        response.end(JSON.stringify(r));
    })
});

db.on('PUT', (request, response) => {
    console.log('DB.PUT');
    // ...реализация
});

db.on('DELETE', (request, response) => {
    console.log('DB.DELETE');
    // ...реализация
});
// -------------------------------------------

http.createServer((request, response) => {
    if (url.parse(request.url).pathname === '/') {
        let html = fs.readFileSync('./04-02.html');
        response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
        response.end(html);
    }
    else if (url.parse(request.url).pathname === '/api/db') {
        db.emit(request.method, request, response);    // для генерации событий, имя события - строка
    }
}).listen(3000);
```

```html
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>04-04/05</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/
</head>
<body>
<h1>Lec 04</h1>
<div id="get_result"></div>
<button onclick="Get()" >GET </button>
<script>···
</script>
<br/>
<div style="padding: 20px">
        <div class='row'>
            <label  class="col-2"> Идентификатор</label>
            <input type="number" class="col-3"   id = "ID"   min="0" />
        </div>
        <div class='row'>
            <label  class="col-2"> ФИО</label>
            <input type="text"   class="col-3"   id = "Name" />
        </div>
        <div class='row'>
            <label  class="col-2"> Дата рождения</label>
            <input type="date"   class="col-3"   id = "BDay"  />
        </div>
        <div class='row'>
            <button class="col-2" onclick="Post()"> POST </button>
        </div>
</div>
<script>···
</script>

</body>
</html>
```

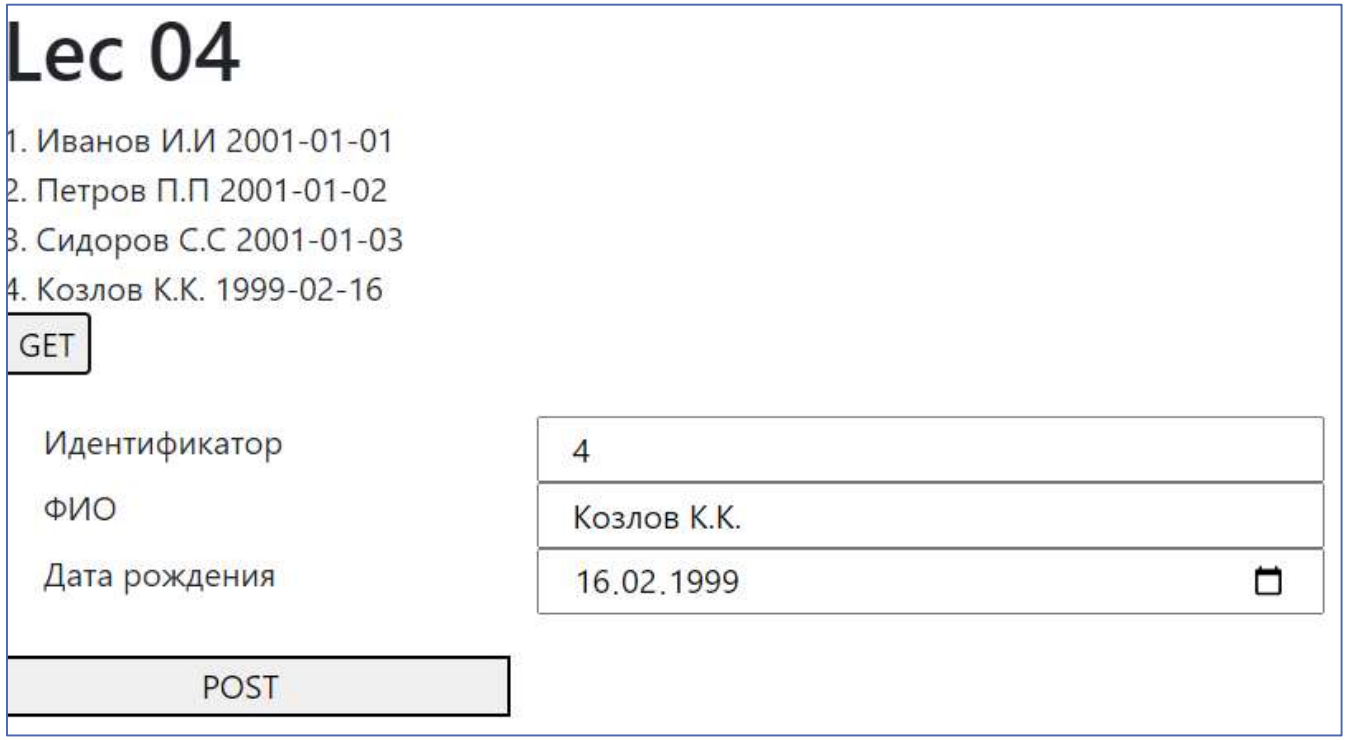
```html
<script>
    function Get(){
            console.log('GET');
            fetch('http://localhost:3000/api/db', {
                                        method: 'GET', mode: 'no-cors',
                                        headers: { 'Content-Type': 'application/json', 'Accept': 'application/json'}
                                        }
            )
            .then(response => { return response.json(); })
            .then((pdata) =>  {
                    console.log('pdata', pdata);
                    get_result.innerHTML = '';
                    pdata.forEach(el => { get_result.innerHTML  +=  ( el.id+ '. ' + el.name + ' ' +  el.bday + '<br/>'); });
            });

    }
</script>
```

```html
<script>
        function Post(){
            console.log('POST');
            fetch('http://localhost:3000/api/db', {
                                        method: 'POST', mode: 'no-cors',
                                        headers: { 'Content-Type': 'application/json', 'Accept': 'application/json'},
                                        body: JSON.stringify({id:ID.value, name: Name.value, bday:BDay.value })
                                            }
            )
            .then(response => { return response.json(); })
            .then((pdata) =>  {console.log('POST.pdata', pdata);});

        }
</script>
```

Результат (браузер)



Методы EventEmitter’a

- emit (eventName[, ...args]) – генерация события с именем eventName. Синхронно вызывает каждый слушатель (в порядке их регистрации) и передает им аргументы args.
- on (eventName, listener) – добавляет слушателя в конец массива слушателей на событие с именем eventName. Один и тот же слушатель может быть вызван для события несколько раз (зависит от того, сколько раз вызван .on().
- once (eventName, listener) – вызов слушателя только один раз.
- removeAllListeners ([eventName]) – удаляет либо всех слушателей для всех событий объекта, либо всех слушателей для указанного события eventName.
- removeListener (eventName, listener) – удаление указанного слушателя для некоторого события с именем eventName.

Таймер = механизм, позволяющий генерировать событие или выполнить некоторое действие через заданный промежуток времени.

setTimeout(), setInterval()

- реализация находится в библиотеке libuv;
- setTimeout и setInterval отноятся к макротаскам;
- коллбэки таймеров выполняются на первой фазе eventloop’a

setTimeout() выполняется только один раз через некоторый промежуток времени.

```
var start = Date.now();
var timer1 = setTimeout((p1,p2)=>{
    console.log(`timer1: p1 = ${p1}, p = ${p2}`);
    console.log('passed: ',  Date.now()-start);
}, 3000, 'p1', 'p2');

var timer2 = setTimeout((p1,p2)=>{
    console.log(`timer1: p1 = ${p1}, p = ${p2}`);
    console.log('passed: ',  Date.now()-start);
}, 4000, 'p3', 'p4');

var timer3 = setTimeout((p1,p2)=>{
    console.log(`timer1: p1 = ${p1}, p = ${p2}`);
    console.log('passed: ',  Date.now()-start);
}, 5000, 'p5', 'p6');
```

```
D:\PSCA\Lec04>
D:\PSCA\Lec04>
D:\PSCA\Lec04>node 04-03
timer1: p1 = p1, p = p2
passed:  3013
timer1: p1 = p3, p = p4
passed:  4003
timer1: p1 = p5, p = p6
passed:  5003

D:\PSCA\Lec04>
```

setInterval() выполняется регулярно через некоторый промежуток времени.

```
var start = Date.now();
var interval1 = setInterval((p1)=>{
                    console.log(`interval1: p1 = ${p1}, passed = ${Date.now() - start}`)
                }, 4000, 'параметр1');
```

```
D:\PSCA\Lec04>node 04-04
interval1: p1 = параметр1, passed = 4003
interval1: p1 = параметр1, passed = 8003
interval1: p1 = параметр1, passed = 12004
interval1: p1 = параметр1, passed = 16005
interval1: p1 = параметр1, passed = 20005
interval1: p1 = параметр1, passed = 24005
interval1: p1 = параметр1, passed = 28006
interval1: p1 = параметр1, passed = 32007
interval1: p1 = параметр1, passed = 36007
interval1: p1 = параметр1, passed = 40007
interval1: p1 = параметр1, passed = 44008
interval1: p1 = параметр1, passed = 48008
```

clearTimeout (timeoutObj) останавливает таймер, созданный с помощью setTimeout().

```
var start = Date.now();
var timer1 = setTimeout((p1)=>{
    console.log(`timer1: p1 = ${p1}`);
    console.log('passed: ',  Date.now()-start);
}, 3000, 'p1');


var timer2 = setTimeout((p1)=>{
    console.log(`timer2: p1 = ${p1}`);
    console.log('passed: ',  Date.now()-start);
}, 5000, 'p2');


var timer3 = setTimeout((p1)=>{
    console.log(`timer3: p1 = ${p1}`);
    console.log('passed: ',  Date.now()-start);
}, 10000, 'p3');


var timer4 = setTimeout((p1)=>{
    console.log(`timer4: p1 = ${p1}`);
    console.log('passed: ',  Date.now()-start);
    clearTimeout(timer2);
}, 4000, 'p4');
```

```
D:\PSCA\Lec04>node 04-05
timer1: p1 = p1
passed:  3013
timer4: p1 = p4
passed:  4003
timer3: p1 = p3
passed:  10002

D:\PSCA\Lec04>
```

В качестве параметра передается объект ранее созданного с помощью setTimeout() таймера, который необходимо отменить.

clearInterval() останавливает таймер, созданный посредством setInterval().

```
var start = Date.now();
var interval1 = setInterval((p1)=>{
                    console.log(`interval1: p1 = ${p1}, passed = ${Date.now() - start}`)
                }, 4000, 'p1');

var timer4 = setTimeout((p1)=>{
                    console.log(`timer4: p1 = ${p1}, passed = ${Date.now() - start}`);
                    clearInterval(interval1);
                }, 10000, 'p4');
```

В качестве параметра передается объект ранее созданного с помощью setTimeout() таймера, который необходимо отменить.

```
D:\PSCA\Lec04>node 04-06
interval1: p1 = p1, passed = 4003
interval1: p1 = p1, passed = 8005
timer4: p1 = p4, passed = 10002

D:\PSCA\Lec04>
```

!

Процесс Node.js работает до тех пор, пока есть события, требующие обработки.

Если выполнить для таймера unref(), то события, генерируемые таймером, не будут учитываться при завершении работы Node.js

```
var start = Date.now();

var interval1 = setInterval((p1)=>{
                    console.log(`interval1:  passed = ${Date.now() - start}`)
                }, 3000, 'p1');
interval1.unref();
```

```
D:\PSCA\Lec04>
D:\PSCA\Lec04>
D:\PSCA\Lec04>
D:\PSCA\Lec04>node 04-07

D:\PSCA\Lec04>
```

```
var start = Date.now();

var timer1 = setTimeout((p1)=>{
    console.log(`timer4: p1 = ${p1}, passed = ${Date.now() - start}`);
    clearInterval(interval1);
}, 10000, 'p4');


var interval1 = setInterval((p1)=>{
                    console.log(`interval1:  passed = ${Date.now() - start}`)
                }, 3000, 'p1');

interval1.unref();
```

```
D:\PSCA\Lec04>node 04-07
interval1:  passed = 3003
interval1:  passed = 6015
interval1:  passed = 9017
timer4: p1 = p4, passed = 10001

D:\PSCA\Lec04>
```

Метод ref() предназначен для противоположной операции.

```
var start = Date.now();

var timer1 = setTimeout(()=>{
    console.log(`timer1: passed = ${Date.now() - start}`);
}, 10000);

var timer2 = setTimeout((p1)=>{
    console.log(`timer2: passed = ${Date.now() - start}`);
    interval1.ref();
}, 14000);

var interval1 = setInterval((p1)=>{
                    console.log(`interval1:  passed = ${Date.now() - start}`)
                }, 3000);
interval1.unref();
```

```
D:\PSCA\Lec04>node 04-08
interval1:  passed = 3002
interval1:  passed = 6014
interval1:  passed = 9017
timer1: passed = 10003
interval1:  passed = 12017
timer2: passed = 14002
interval1:  passed = 15018
interval1:  passed = 18019
interval1:  passed = 21021
interval1:  passed = 24023
interval1:  passed = 27024
interval1:  passed = 30026
interval1:  passed = 33027
interval1:  passed = 36028
```

```
const start = Date.now();
const http = require('http');

const server = http.createServer((req, res) => {
    res.end('<h1>server</h1>');
}).listen(3000, () => console.log('Server is running at http://localhost:3000'));

setTimeout(() => {
    console.log(`terminate, passed ${Date.now() - start}`)
    server.close(() => console.log('Server terminate'))
}, 20000);

const interval = setInterval(() => {
    console.log(`statistic calculation: passed = ${Date.now() - start}`)
}, 3000);

interval.unref();
```

```
PS D:\NodeJS\samples\cwp_04> node .\04-09.js
Server is running at http://localhost:3000
statistic calculation: passed = 3036
statistic calculation: passed = 6037
statistic calculation: passed = 9042
statistic calculation: passed = 12051
statistic calculation: passed = 15067
statistic calculation: passed = 18075
terminate, passed 20034
Server terminate
```

```
PS D:\NodeJS\samples\cwp_04> node .\04-09.js
Server is running at http://localhost:3000
statistic calculation: passed = 3013
statistic calculation: passed = 6026
statistic calculation: passed = 9027
statistic calculation: passed = 12034
statistic calculation: passed = 15039
statistic calculation: passed = 18050
terminate, passed 20024
statistic calculation: passed = 21059
statistic calculation: passed = 24068
statistic calculation: passed = 27074
statistic calculation: passed = 30081
statistic calculation: passed = 33081
Server terminate
PS D:\NodeJS\samples\cwp_04>
```

!

Метод unref() есть не только у таймеров, есть еще у серверов (server.unref()), сетевых сокетов (socket.unref()) и др.