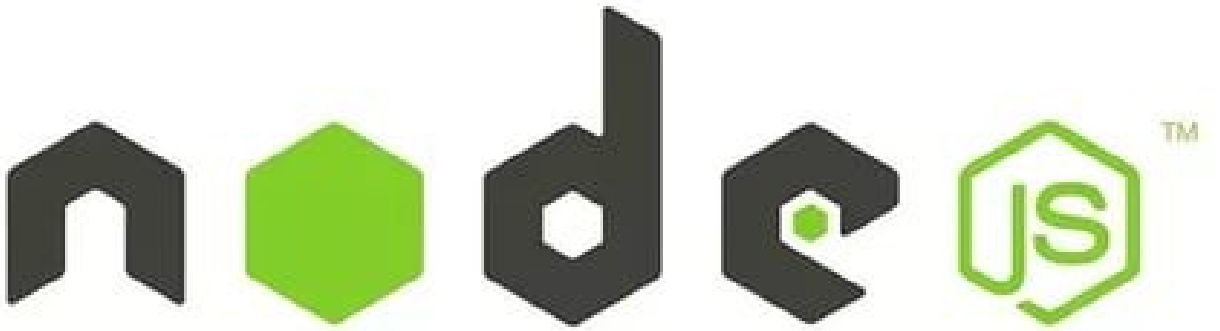

node
JS
TM

# Node.js - архитектура

- Объект Global
- Объект Process

# 1 | Объект Global

Объект **global** является объектом глобального пространства имен. В некоторой степени он похож на объект window браузера тем , что предоставляет доступ к глобальным свойствам и методам и на него не нужно ссылаться непосредственно по имени. В Node.js переменная, объявленная в любом месте модуля,  будет определена только в этом модуле , чтобы она стала глобальной, необходимо объявить её как свойство объекта global

```
global.val = 10;
console.log(global);
```

# 1 | Будет ли конфликт имен при подключении модуля ?

Что такое модуль ?
Модулем считается файл с кодом JavaScript. В этом файле ключевым словом **export** помечаются переменные и функции, которые могут быть использованы снаружи. Другие модули могут подключать их через вызов **require.**

```
file: mod1.js

var globalValue;

exports.setGlobal  = function(val) {
     globalValue = val;
}

exports.returnGlobal = function() {
    return  globalValue
}
```

# 1 | Будет ли конфликт имен при подключении модуля ?

В данном примере переменная globalValue из файла `mod1.js` не будет экспортирована в контекст `server.js` и это не приведет к потенциальному конфликту глобальных переменных. Мы имеем доступ только к тем методам и переменным которые явно определили через ключевое слово `exports`

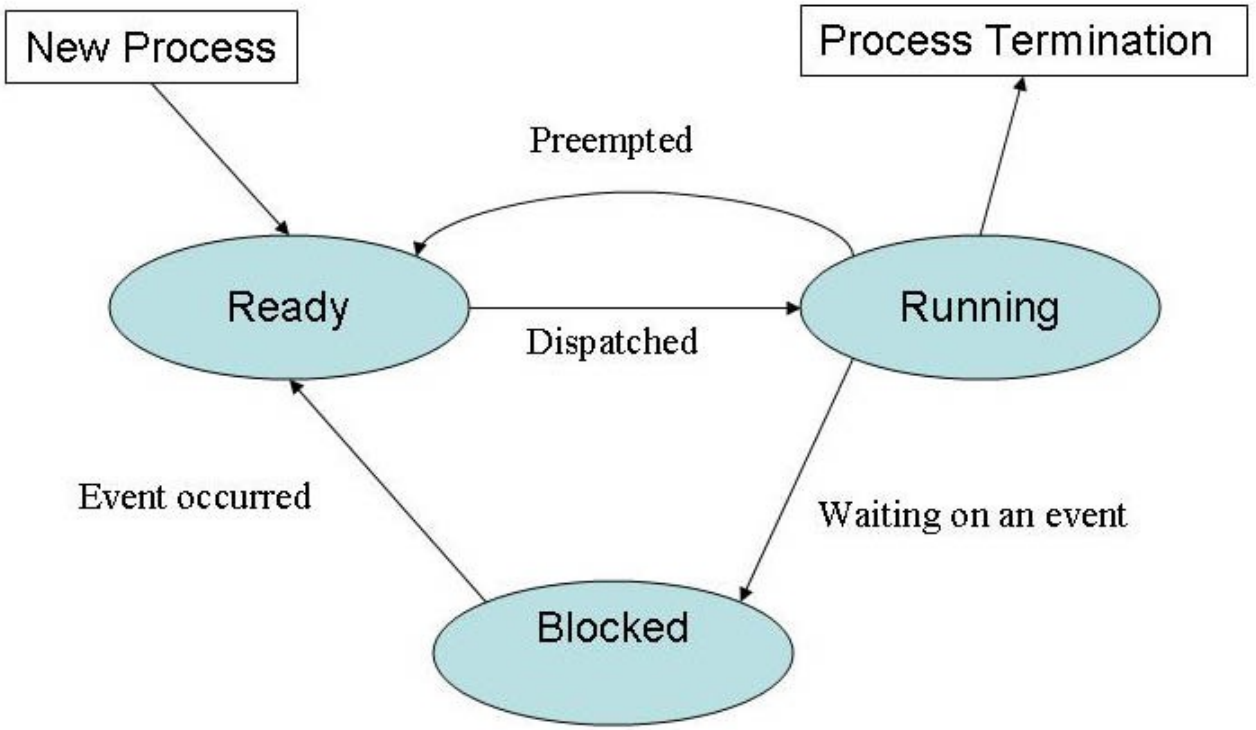
file: server.js

```
var mod1 = require('./mod1.js');
mod1.setGlobal(100);
var val = mod1.returnGlobal();
console.log(global);
```

# 2 | Объет process

Каждое Node.js-приложение - это экземпляр объекта Process  наследующий его свойства. Свойства и методы, несут информацию о приложении и контексте его исполнения.

# Process Node.js в OS



# 2 | Свойства объекта process

Сведения о запущенном процессе можно получить из его свойств.

```
console.log(process.execPath); // путь запуска
console.log(process.version);  // версия Node
console.log(process.platform); // платформа
console.log(process.arch); //аппаратная архитектура
console.log(process.title); // описание
console.log(process.pid); // pid процесса
```

# 2 | Свойства объекта process

Свойство process.argv содержит массив аргументов командной строки:

```
process.argv.forEach(function(val, index, array) (
     console.log(index + ' - ' + val);
1);
```

Запустим скрипт с несколькими аргументами

```
>node process.js foo bar 1
```

```
0 - node
1 - /home/www/process.js
2 - foo
3 - bar
4 - 1
```

# 2 | Свойства объекта process

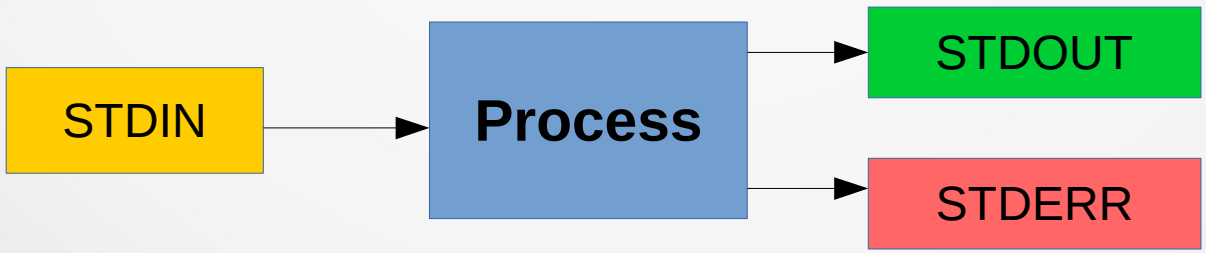
Свойство process.env возвращает объект, хранящий пользовательское окружение процесса, и практически незаменим при отладке приложений.

```
console.log(process.env);
```

```
{ XDG_VTNR: '7',
SSH_AGENT_PID: '950',
XDG_SESSION_ID: '1',
SHELL: '/bin/bash',
TERM: 'xterm',
LANG: 'ru_RU.UTF-8',
GDM_LANG: 'ru_RU.utf8',
GDMSESSION: 'openbox',
SHLVL: '1'}
```

# 2 | Процессы ввода/вывода

Стандартные процессы ввода/вывода операционной системы stdin,stdout и stderror - также имеют свое воплощение в объекте process.



# 2 | Процессы ввода/вывода

```
process.stdin.setEncoding('utf8');
process.stdin.resume();
process.stdin.on('data', function(chunk) {

if (chunk !== "end\n") {
    process.stdout.write('data: ' + chunk);
} else {
    process.kill();
}});
```

Сначала устанавливаем кодировку ввода. Затем вызываем метод process.stdin.resume(), возобновляющий работу потока process.stdin (он по умолчанию приостановлен). Далее к стандартному вводу запущенного процесса привязывается обработчик на событие поступления данных. В функции обратного вызова, process.stdout просто пишет поступившие данные в консоль (до того, пока не будет встречено слово end и перевод строки). Запустим этот код и напечатаем что-нибудь в консоли.

# 2 | Сигнальные события

У объекта Process, как и у всех javaScript-объектов, есть свои события, с которыми можно связать необходимые обработчики. Нас интересуют события, специфичные для этого объекта, и прежде всего это так называемые сигнальные события (Signal Events), генерирующиеся при получении процессом сигнала .  Сигналы могут быть стандартные, POSIX-ов -SIGINT, SIGUSR1, SIGTSTP и др.

# 2 | Сигнальные события

```
process.on('SIGHUP', function(){
        console.log('Got a SIGHUP');
});

setInterval(function(){ console.log('Running');}, 10000);
console.log('PID: ', process.pid);
```

После запуска кода из другой консоли, зная PID запущенного процесса, можно послать ему требуемый сигнал:

```
>kill -s SIGHUP 2645
```

# 2 | Child Process - параллелизм в Node.js

Метод child_process.fork() запускает дочерние процессы. Продемонстрировать его работу можно следующим простым кодом

main.js:

```
var cp = require('child_process');
var child1 = cp.fork(__dirname + '/child1.js');
var child2 = cp.fork(__dirname + '/child2.js');
while(cp){
   console.log{"runnin main");
}
```

Содержимое child1.js

```
while(1) {  console.log("running: process 1"); }
```

Содержимое child2.js

```
while(1) {  console.log("running: process 2"); }
```

После запуска все выполняется параллельно

# 2 | Child Process - параллелизм в Node.js

Еще один метод - child.send() - самый интересный. В соответствии со своим названием он отсылает сообщения. Отсылает сообщения от процесса к процессу

**main.js:**

```
var cp = require('child_process');
var child = cp.fork(__dirname + '/sub.js');
child.on ( 'message',  function (data) {
   console.log ( 'Main got message: ', data);
});
child.send ({ hello: 'child' });
```

**sub.js:**

```
process.on('message', function(m) { console. log ( 'Child
got message: ', m);});
process.send({ foo: 'bar' }) ;
```

**Результат:**

```
$ node process
Main got message: { foo: 'bar' }
Child got message: { hello: 'child' }
```

# Полезный ссылки

https://nodejs.org - официальный сайт

https://goo.gl/nHlYO8 - изучаем Node.js