Node.js in 2020 #2

Выйди и зайди нормально

github.com/HowProgrammingWorks

Timur Shemsedinov

Chief Technology Architect at Metarhia
Lecturer at Kiev Polytechnic Institute

github.com/tshemsedinov

Проблемы

реальные и мнимые, пути их решения и тупики

Что беспокоит умы:

- Какой фреймворк самый лучший?
- Какая структура папок правильная?
- Какую СУБД и ORM/ODM взять для работы?
- Как масштабировать сервер?
- Что быстрее, колбеки или промисы?
- Куда поместить ..., в модель или контроллер?
- Будем ли разделать на микросервисы?

Короткие ответы

- Фреймворк не важен, разделять делить слои
- Структуры данных важнее структуры папок
- Можно брать любую СУБД если это PG
- Масштабироваться? Вам точно это нужно?
- Колбеки и промисы влияют на <1% скорости
- Забудьте MVC, узнайте SOLOD, GRASP, GoF
- Проектируйте интерфейсы, не микросервисы

О чем лучше подумать

- Обработка ошибок
- Проектирование программных абстракций
- Зацепление и связность и закон Деметры
- GRASP принцип “Информационный эксперт”
- Очереди запросов и пулы объектов
- Как минимизировать I/O и CPU задачи
- Изоляция контекстов запросов

Слои

- Слой доступа к данным
- Слой сетевого взаимодействия
- Слой бизнес-логики

GRASP, SOLID, GoF

Как это может быть связано с JavaScript?

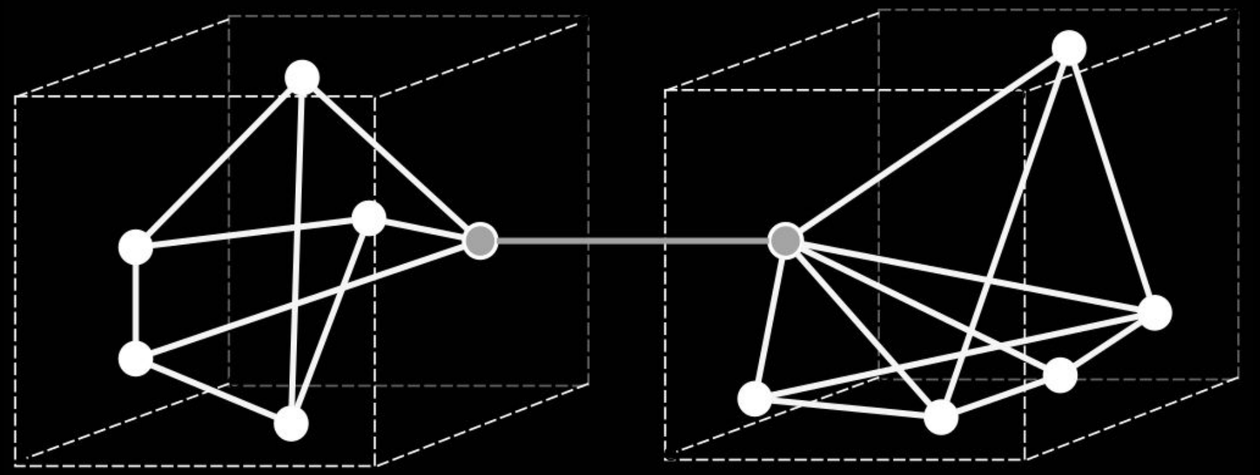
GRASP

General responsibility assignment software patterns (распределение ответственности)

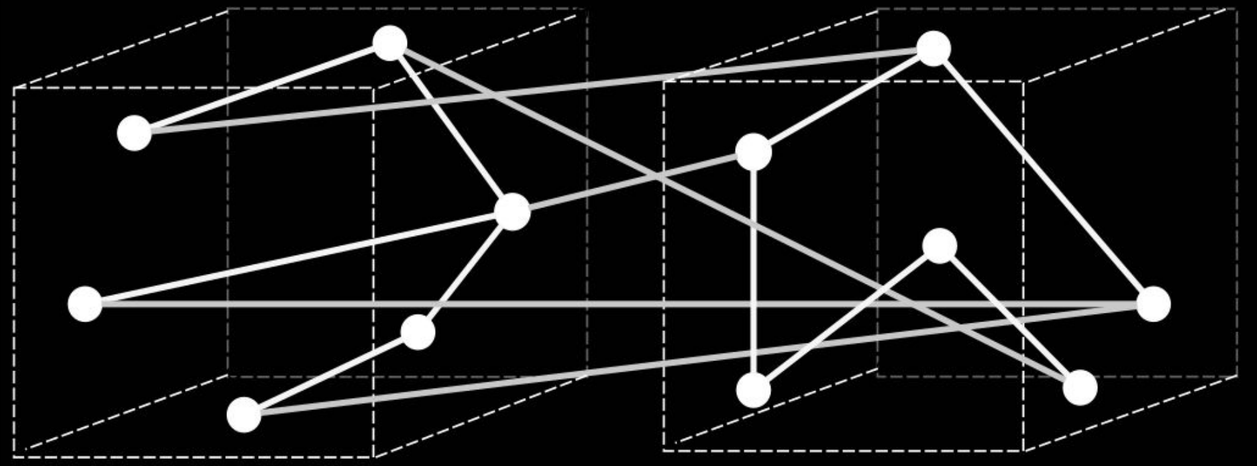
Книга “Применение UML и шаблонов проектирования” // Крэг Ларман

GRASP: Coupling and cohesion

Cohesion (связность) внутри модуля или программного компонента



Coupling (зацепление) между модулями



GRASP:

General responsibility assignment software patterns (распределение ответственности)

Low Coupling
Information Expert
Controller
Pure Fabrication
Protected Variations

High Cohesion
Creator
Polymorphism
Indirection

SOLID: задачи

Майкл Фэзерс (Michael Feathers)
Роберт Мартин (Robert Martin, Uncle Bob)

Что они дают:

- Облегчение модификации и расширения
- Улучшение владения кодом и TTM
- Способность быстро понимать друг друга

SOLID: 5 принципов

- The Single Responsibility Principle
- The Open Closed Principle
- The Liskov Substitution Principle
- The Interface Segregation Principle
- The Dependency Inversion Principle
  (не путать с dependency injection и inversion of control)

Шаблоны GoF

Gang of Four (GoF): Эрих Гамм, Ричард Хелм, Ральф Джонсон, Джон Влиссидес

Design Patterns — Elements of Reusable Object-Oriented Software (23 шаблона)

Классификация шаблонов

Порождающие: абстрактная фабрика, строитель, фабричный метод, пул, синглтон...
Структурные: адаптер, мост, компоновщик, декоратор или обертка, фасад, прокси...
Поведенческие: цепочка обязанностей, команда, обсервер, итератор, стратегия...
Коммуникационные: cqs, cqrs, event sourcing...

Node.js Security in 2020

Node.js Security Aspects

- Concurrency model for I/O
- Common path traversal vulnerability
- SQL injection, XSRF, XSS etc.
- Resource leaks (memory, handlers, etc.)
- Passwords should be stored as hash with salt
- Load control (DoS/DDoS)

Security issues

- Dependencies: see your node_modules
  - Unreliable dependencies
  - Malicious modules from NPM
- Sandbox escaping (vm)
- Buffer vulnerabilities
- Regular expressions

Tools Node.js

- Linter may help github.com/nodesecurity/eslint-plugin-security
- We have npm audit and it may even fix multiple problems automatically npm audit fix
- Special tools: ~~nsp~~, snyk
- Githib have built-in Security Alert

SQLI (SQL Injection)

Hello! See what they say about you:
http://bank-web-site.com/accounts?
name='marcus'%20OR%201=1%20--%20

https://bit.ly/2XZpJMt

“SELECT * from Accounts where name=” + name

XSRF (Cross-Site Request Forgery)

They can send you:

Hello! See what they say about you:
http://payment-system.com/api/transfer?
amount=1000&destination=card-number

XSS (Cross-Site Scripting)

They can send you:

Hello! See what they say about you:
http://control-panel.com/help.php?q=
%3Cscript%3Ealert('Hello');%3C/script%3E

CSP (Content Security Policy)

Browser have built-in layer to create security policy to solve XSS problem

https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

Path traversal

```
const serveFile = fileName => {
  const filePath = path.join(STATIC_PATH, fileName);
  return fs.createReadStream(filePath);
};

http.createServer((req, res) => {
  const url = decodeURI(req.url);
  serveFile(url).pipe(res);
}).listen(8000);
```

```
curl -v http://127.0.0.1:8000/%2e%2e/1-traversal.js
```

Path traversal fixed

```
const serveFile = fileName => {
  const filePath = path.join(STATIC_PATH, fileName);
  if (!filePath.startsWith(STATIC_PATH)) {
    throw new Error(`Access denied: ${name}`);
  }
  return fs.createReadStream(filePath);
};
http.createServer((req, res) => {
  const url = decodeURI(req.url);
  serveFile(url).pipe(res);
}).listen(8000);
```

What’s next:

- httponly cookies
  https://www.owasp.org/index.php/HttpOnly
- HTTP Headers:
  - X-XSS-Protection
  - X-Frame-Options
  - X-Content-Type-Options
    etc.

OWASP (Open Web App. Security Project)

See this site:

https://owasp.org/

Node.js

Patterns and Antipatterns

Node.js antipatterns classification

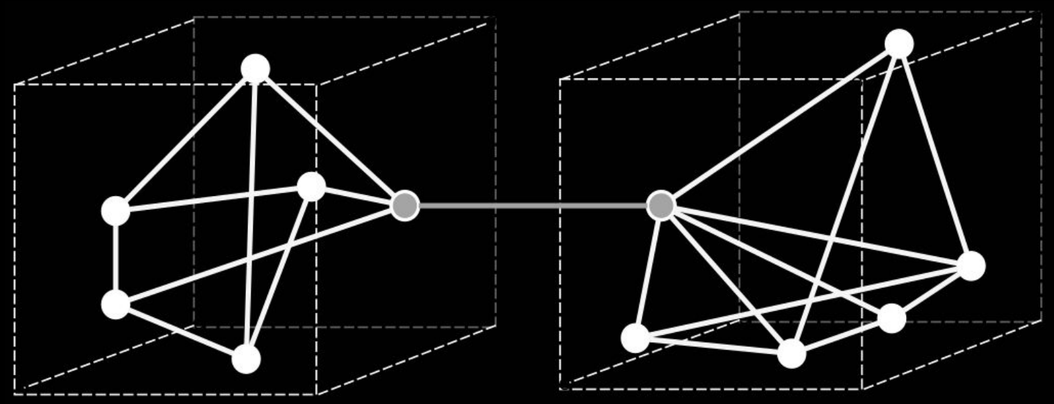
- Structure and arch.
- Initialization
- Dependency issues
- Application state
- Middlewares
- Context isolation
- Security issues
- Asynchronity issues
- Blocking operations
- Memory leaks
- Databases and ORM
- Error handling

No layers, everything mixed

- Configuration and Dependency management
- Network protocols related code (http, tcp, tls...)
- Request parsing, Cookies, Sessions
- Logging, Routing, Business-logic
- I/O: fs, Database queries
- Generating responses and error generation
- Templating, etc.

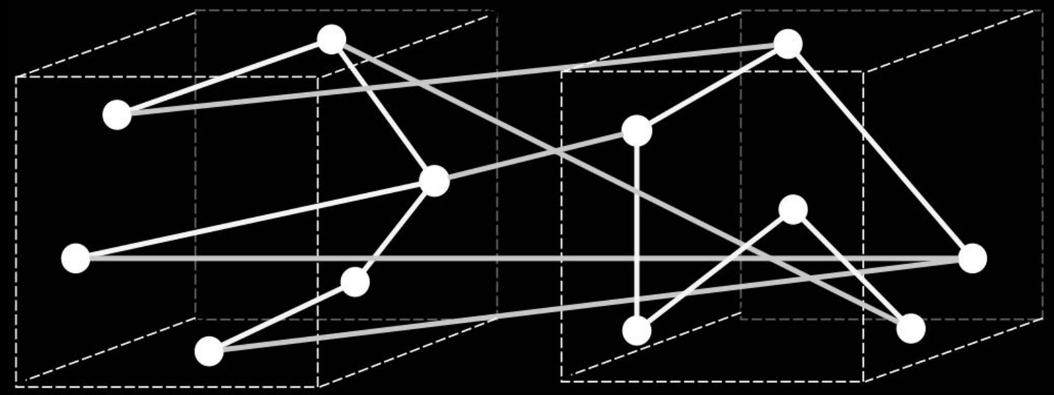
Middlewares

Middlewares is an extremely bad idea for low coupling and high cohesion



Middlewares changes:

- Socket state
- Db connection state
- Server state



Don’t create global state

```
let groupName;

app.use((req, res, next) => {
  groupName = 'idiots'; next();
});

app.get('/user', (req, res) => {
  if (groupName === 'idiots') {
    res.end('I know you!');
  }
});
```

Don’t mixin to req, res, ctx

```
app.use((req, res, next) => {
  res.groupName = 'idiots';
  next();
});

app.get('/user', (req, res) => {
  if (res.groupName === 'idiots') {
    res.end('I know you!');
  }
});
```

Special place for the state: res.locals

```
app.use((req, res, next) => {
  res.locals.groupName = 'idiots';
  next();
});

app.get('/user', (req, res) => {
  if (res.locals.groupName === 'idiots') {
    res.end('I know you!');
  }
});
```

Don’t mixin methods

```
app.get('/user/:id', (req, res, next) => {
  req.auth = (login, password) => { /* auth */ };
  next();
});

app.get('/user/:id', (req, res) => {
  if (req.auth(req.params.id, '111')) {
    res.end('I know you!');
  }
});
```

Don’t require in middleware / handler

```
app.get((req, res, next) => {
  req.db = new require('pg').Client();
  req.db.connect();
  next();
});

app.get('/user/:id', (req, res) => {
  req.db.query('SELECT * from USERS', (e, r) => {
    ...
  });
});
```

Don’t connect DB from handlers

```
app.get((req, res, next) => {
  req.db = new Pool(config);
  next();
});

app.get('/user/:id', (req, res) => {
  req.db.query('SELECT * from USERS', (e, r) => {
  });
});
```

Connection leaks is easy

```
const db = new Pool(config);

app.get('/user/:id', (req, res) => {
  req.db.query('SELECT * from USERS', (err, r) => {
    if (err) throw err;
    // Prepare data to reply client
  });
});
```

Don’t use blocking operations

- Sync calls like fs.readFileSync
- Console output like console.log
- Remember that require is synchronous
- Long loops (including for..of and for await)
- Serialization: JSON.parse, JSON.stringify
- Iteration: loops, Array.prototype.map, etc.
- CPU-intensive: zlib, crypto

Loop: for await of is blocking

```
(async () => {
  let ticks = 0;
  const timer = setInterval(() => ticks++, 10);
  const numbers = new Array(1000000).fill(1);
  let i = 0;
  for await (const number of numbers) i++;
  clearInterval(timer);
  console.dir({ i, ticks });
})();

// { i: 1000, ticks: 0 }
```

AsyncArray (short version)

```
class AsyncArray extends Array {
  [Symbol.asyncIterator]() {
    let i = 0;
    return {
      next: () => new Promise(resolve => {
        setTimeout(() => resolve({
          value: this[i], done: i++ === this.length
        }), 0);
      })
    };
  }
}  // github.com/HowProgrammingWorks/NonBlocking
```

Loop: for await of + AsyncArray

```
(async () => {
  let ticks = 0;
  const timer = setInterval(() => ticks++, 10);
  const numbers = new AsyncArray(10000000).fill(1);
  let i = 0;
  for await (const number of numbers) i++;
  clearInterval(timer);
  console.dir({ i, ticks });
})();
```

```
// { i: 10000, ticks: 1163 }
```

https://github.com/HowProgrammingWorks/NonBlocking

Memory leaks

- References
  - Global variables
  - Mixins to built-in Classes
  - Singletons, Caches
- Closures / Function contexts
  - Recursive closures
  - Require in the middle of code
  - Functions in loops

Memory leaks

- OS and Language Objects
  - Descriptors: files, sockets...
  - Timers: setTimeout, setInterval
- Events / Subscription / Promises
  - EventEmitter
  - Callbacks, Not resolved promises

Links

github.com/HowProgrammingWorks/AbstractionLayers

github.com/HowProgrammingWorks/MemoryLeaks

Use global, Luke!

for interfaces but don’t use require for data

Почему нельзя global?

- Нельзя, потому, что нельзя
- Способствует протеканию состояния
- Global один на все приложение

Почему require не всегда хорошо?

- Если экспортировать структуру данных или коллекцию (array, map, set) то же, что и global
- Если экспортировать синглтон класс,
  это то же, что и global
- Миксинить к глобальным или загруженным через require объектам, модифицировать встроенные классы, это то же, что и global

Связываем через интерфейсы

- Важно связывать программные компоненты через интерфейсы или контракты, но не через данные
- А как передаются ссылки на интерфейсы, через global, require, object composition, object aggregation, abject association, события или еще как... это не столь важно...

Errors

кто не обрабатывает ошибки, тот

Callback-last, error-first

```
const fn = (arg1, arg2, callback) => {
  if (...) callback(null, result);
  else callback(new Error('message'));
};

fn(arg1, arg2, (err, result) => {
  if (err) {
    console.log(err.message);
    return;
  }
  console.log(result);
});
```

Error ignoring

```
fn1(arg1, arg2, (err, res1) => {
  fn2(res1, arg3, (err, res2) => {
    fn3(res2, arg4, arg5, (err, res3) => {
      doSomething(arg5, res3);
    });
  });
});
```

Error ignoring

```
const cb3 = (err, res3) => {
  doSomething(arg6, res3);
};
const cb2 = (err, res2) => {
  fn3(res2, arg4, arg5, cb3);
};
const cb1 = (err, res1) => {
  fn2(res1, arg3, cb2);
};
fn1(arg1, arg2, cb1);
```

Promise.catch

```
doSomething(arg1, arg2)
  .then(
    data => { console.log({ data }); },
    err => { console.log({ err }); }
  );

doSomething(arg1, arg2)
  .then(data => { console.log({ data }); })
  .catch(err => { console.log({ err }); });
```

Promise.catch

```
doSomething(arg1, arg2)
  .then(
    data => { console.log({ data }); },
    err => { console.log({ err }); }
  );

doSomething(arg1, arg2)
  .then(data => { console.log({ data }); })
  .catch(err => { console.log({ err }); });
```

.then(fulfilled, rejected).catch(rejected)

```
doSomething(arg1, arg2)
  .then(
    data => { console.log({ data }); },
    err => { console.log({ err }); }
  )
  .catch(err => { console.log({ err }); });

// Помним про .finally()
```

Error ignoring

```
(async () => {

  const file1 = await readFile('file1.ext');
  const file2 = await readFile('file2.ext');
  console.dir({ file1, file2 });

})();
```

Error ignoring

```
(async () => {

  try {
    const file1 = await readFile('file1.ext');
    const file2 = await readFile('file2.ext');
  } catch (err) {
    console.error(err);
  }

})();
```

Catch unhandled errors

```
process.on('uncaughtException', err => {
  console.log('on uncaughtException: ' + err.message);
  process.exit(1);
});

process.on('warning', warning => {
  console.log({ warning });
});

// UnhandledPromiseRejectionWarning: Error: msg...
```

Catch unhandled errors

```
process.on('rejectionHandled', promise => {
  console.log({ rejectionHandled: { promise } });
});

process.on('multipleResolves', (type, p, reason) => {
  console.log({
    multipleResolves: { type, promise: p, reason }
  });
});
```

Data Access Layer

SQL, noSQL and Hybrid DB
Query builder, ORM & ODM

Ссылки

- Слой доступа к данным, курсор, транзакция
  https://youtu.be/CRcSWtWVvrA

- Работа с базами данных в Node.js
  на примере PostgreSQL
  https://youtu.be/2tDvHQCBt3w

Event sourcing

CQS, CQRS & Command patterns for server scailing

Ссылки

- Паттерн Команда (Command)
  https://youtu.be/vER0vYL4hM4

- CQS, CQRS, Event Sourcing
  https://youtu.be/T2tRc80Q8Qw
  https://youtu.be/kFNtKiK2SPs
  https://youtu.be/xp5MVKEqxY4

Node.js Starter Kit

no dependencies, 15-20 kb with pg drivers + 1.2 mb and ws + 0.24 mb

github.com
/HowProgrammingWorks
/NodejsStarterKit

Questions?

github.com/tshemsedinov
https://youtube.com/TimurShemsedinov
github.com/HowProgrammingWorks/Index

Весь курс по ноде (>35.5 часов)
https://habr.com/ru/post/485294/

t.me/HowProgrammingWorks
t.me/NodeUA

timur.shemsedinov@gmail.com