html academy
интерактивные
онлайн-курсы

Раздел 2:

Модули, пакеты и отладчик

В этой лекции

– Модули

– Что такое NPM?

– Настроим проект

– Подключим зависимости

– Научимся отлаживать программу

Модули

Модули и загрузчики модулей

Модули и загрузчики модулей

– Модуль

Модули и загрузчики модулей

- Модуль
  - изолирует пространство имен

Модули и загрузчики модулей

- Модуль
  - изолирует пространство имен
  - описывает зависимости

Модули и загрузчики модулей

- Модуль
  - изолирует пространство имен
  - описывает зависимости
  - описывает то, что он предоставляет наружу

Модули и загрузчики модулей

- Модуль
  - изолирует пространство имен
  - описывает зависимости
  - описывает то, что он предоставляет наружу
- Загрузчик модулей

Модули и загрузчики модулей

- Модуль
  - изолирует пространство имен
  - описывает зависимости
  - описывает то, что он предоставляет наружу
- Загрузчик модулей
  - ищет модуль

Модули и загрузчики модулей

- Модуль
  - изолирует пространство имен
  - описывает зависимости
  - описывает то, что он предоставляет наружу
- Загрузчик модулей
  - ищет модуль
  - загружает модуль по требованию

Модули и загрузчики модулей

- Модуль
  - изолирует пространство имен
  - описывает зависимости
  - описывает то, что он предоставляет наружу
- Загрузчик модулей
  - ищет модуль
  - загружает модуль по требованию
  - кеширует уже загруженные модули

Экспорт именованых значений из модуля

```
// Файл father.js

module.exports.name = `Папа`;
module.exports.age = 12;
module.exports.male = true;
```

Экспорт именованых значений из модуля

```
// Файл father.js

exports.name = `Папа`;
exports.age = 12;
exports.male = true;
```

Экспорт по умолчанию из модуля

```
// Файл mother.js

module.exports = {
   name: `Мама`,
   age: 12,
   male: false
};
```

Загрузка значений из модуля

```
// Файл family.js

const fatherName = require(`./father`).name;
const motherName = require(`./mother`).name;

const makeChild = () => `Отец: ${fatherName}, мать: ${motherName}`;

module.exports = makeChild();
```

Ограничения пути модуля

```
// Относительные пути до файлов:
require('/utils/bar'); // Абсолютный путь
require('./bar');      // Относительный путь от текущего модуля
require('../bar');     // Относительный путь от родительской папки

// Зарезервированный синтаксис для зависимостей:
require('bar');        // Синтаксис поиска во внешнем модулей
require('utils/bar');  // Синтаксис поиска относительно внешнего модуля
```

Особенности *require*

Особенности *require*

– Когда Node.js встречает встроенную функцию *require* он интерпретирует строчку, которую в неё передали как путь до модуля который нужно загрузить

Особенности *require*

– Когда Node.js встречает встроенную функцию *require* он интерпретирует строчку, которую в неё передали как путь до модуля который нужно загрузить

– Модуль загружается ровно один раз:
После того как Node.js однажды загрузил модуль — он его кеширует (сохраняет результат выполнения модуля), т.е. никакой модуль не будет загружен и выполнен дважды, неважно сколько раз он был подключён

Особенности *require*

- Когда Node.js встречает встроенную функцию *require* он интерпретирует строчку, которую в неё передали как путь до модуля который нужно загрузить

- Модуль загружается ровно один раз:
  После того как Node.js однажды загрузил модуль — он его кеширует (сохраняет результат выполнения модуля), т.е. никакой модуль не будет загружен и выполнен дважды, неважно сколько раз он был подключён

- Node.js всегда использует UNIX-style пути (через /)

Особенности *require*

- Когда Node.js встречает встроенную функцию *require* он интерпретирует строчку, которую в неё передали как путь до модуля который нужно загрузить

- Модуль загружается ровно один раз:
  После того как Node.js однажды загрузил модуль — он его кеширует (сохраняет результат выполнения модуля), т.е. никакой модуль не будет загружен и выполнен дважды, неважно сколько раз он был подключён

- Node.js всегда использует UNIX-style пути (через /)

- Пути в регистрозависимых и регистронезависимых файловых системах ведут себя по разному:
  Более того в регистронезависмых системах по разному загруженые модули могут приводить к повторной загрузке модуля

Функция подключения модуля *require*

Функция подключения модуля *require*

– Расширение файла можно опустить

Функция подключения модуля *require*

- Расширение файла можно опустить
- Если сослаться на папку с модулем *index.js*, то Node.js подключит загрузит этот модуль из папки

Функция подключения модуля *require*

- Расширение файла можно опустить
- Если сослаться на папку с модулем *index.js*, то Node.js подключит загрузит этот модуль из папки
- Выражение *require.main === module* — вернёт *true* только в том случае, если этот модуль был входным файлом (точкой входа)

Модульная обёртка (*Module wrapper*)

```
function load(info) {
  const module = {
    exports: {},
    require: (path) => { /* ... */ }
  };
  ((exports, require, module, __filename, __dirname) => {
    // Файл mother.js

    module.exports = {
      name: `Мама`,
      age: 12,
      male: false
    };

    exports.name = `Светлана`;
  })(module.exports, module.require, module);
  return module.exports;
}
```

Тело модуля

nodejs.org/docs/latest/api/modules.html#modules_the_module_wrapper

Область видимости модуля

nodejs.org/docs/latest/api/modules.html#modules_the_module_scope

Область видимости модуля

– *module* — объект, описывает текущий модуль

nodejs.org/docs/latest/api/modules.html#modules_the_module_scope

Область видимости модуля

– *module* — объект, описывает текущий модуль

– *exports* — ссылка на объект *module.exports*

nodejs.org/docs/latest/api/modules.html#modules_the_module_scope

Область видимости модуля

– *module* — объект, описывает текущий модуль

– *exports* — ссылка на объект *module.exports*

– *require* — функция, которая подключает модули

nodejs.org/docs/latest/api/modules.html#modules_the_module_scope

Область видимости модуля

– *module* — объект, описывает текущий модуль

– *exports* — ссылка на объект *module.exports*

– *require* — функция, которая подключает модули

– *__filename* — абсолютный путь до модуля

nodejs.org/docs/latest/api/modules.html#modules_the_module_scope

Область видимости модуля

– *module* — объект, описывает текущий модуль

– *exports* — ссылка на объект *module.exports*

– *require* — функция, которая подключает модули

– *__filename* — абсолютный путь до модуля

– *__dirname* — абсолютный путь до директории в которой лежит модуль

nodejs.org/docs/latest/api/modules.html#modules_the_module_scope

CommonJS

```
// Файл mother.js

module.exports = {
  name: `Мама`,
  age: 12,
  male: false
};
```

```
// Файл father.js

module.exports = {
  name: `Папа`,
  age: 12,
  male: true
};
```

```
// Файл family.js

const fatherName = require(`./father`).name;
const motherName = require(`./mother`).name;

const makeChild = () => `Отец: ${fatherName}, мать: ${motherName}`;

module.exports = makeChild();
```

www.commonjs.org

CommonJS ES2015+

```
// Файл mother.js

module.exports = {
  name: `Мама`,
  age: 12,
  male: false
};
```

```
// Файл father.js

module.exports = {
  name: `Папа`,
  age: 12,
  male: true
};
```

```
// Файл family.js

const {name: fatherName} = require(`./father`);
const {name: motherName} = require(`./mother`);

const makeChild = () => `Отец: ${fatherName}, мать: ${motherName}`;

module.exports = makeChild();
```

www.commonjs.org

ES2015 Модули

```
// Файл mother.mjs

export default {
  name: `Мама`,
  age: 12,
  male: false
};
```

```
// Файл father.mjs

export const name = `Папа`;
export const age = 12;
export const male = true;
```

```
// Файл family.mjs

import {name as fatherName} from './father';
import mother from './mother';
const motherName = mother.name;

const makeChild = () => `Отец: ${fatherName}, мать: ${motherName}`;

export default makeChild();
```

ES2015 модули

ES2015 модули

– Поддерживаются начиная с версии 9

ES2015 модули

– Поддерживаются начиная с версии 9

– Находятся за флагом *--experimental-modules*

ES2015 модули

– Поддерживаются начиная с версии 9

– Находятся за флагом *--experimental-modules*

– Работают только с файлами расширения *.mjs*

Ограничения модулей

Ограничения модулей

– Никогда не кладите объект в переменную *exports*.
  Лучше всегда использовать *module.exports*

Ограничения модулей

– Никогда не кладите объект в переменную *exports*.
  Лучше всегда использовать *module.exports*

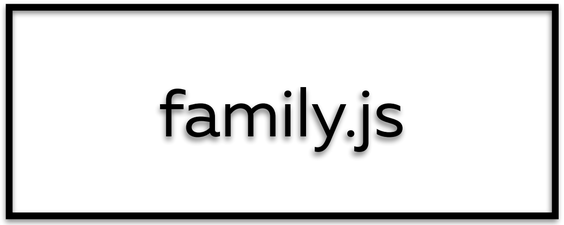
– Никогда не экспортируйте значения асинхронно или внутри условий, т.к. они могут оказаться непроинициализированы

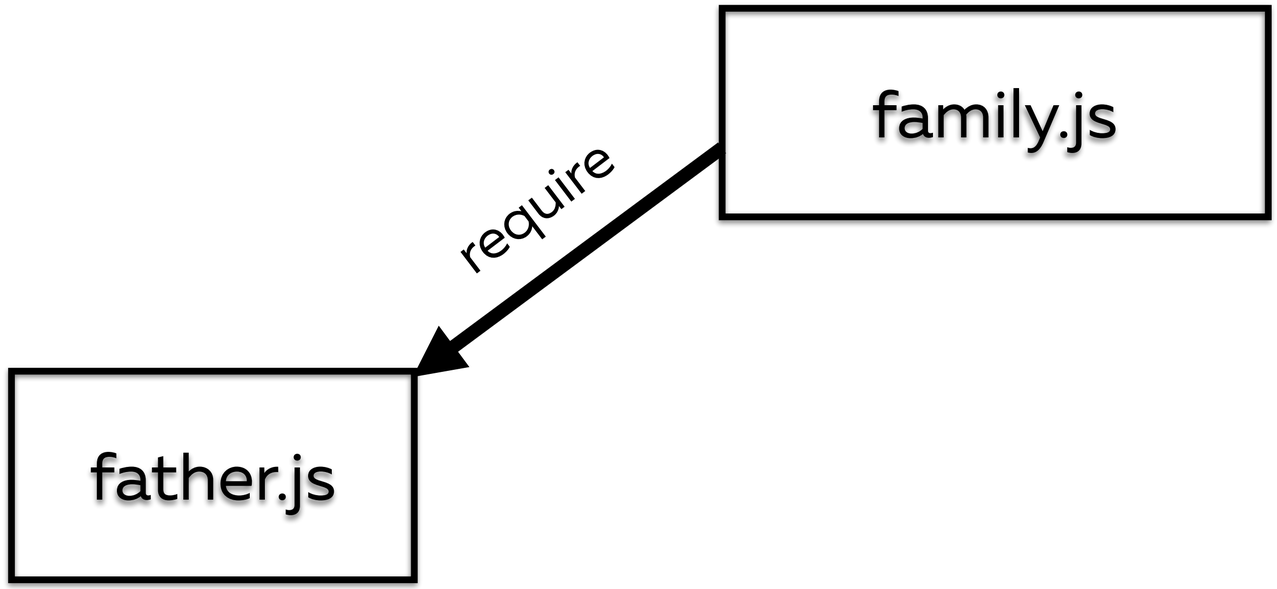
Ограничения модулей

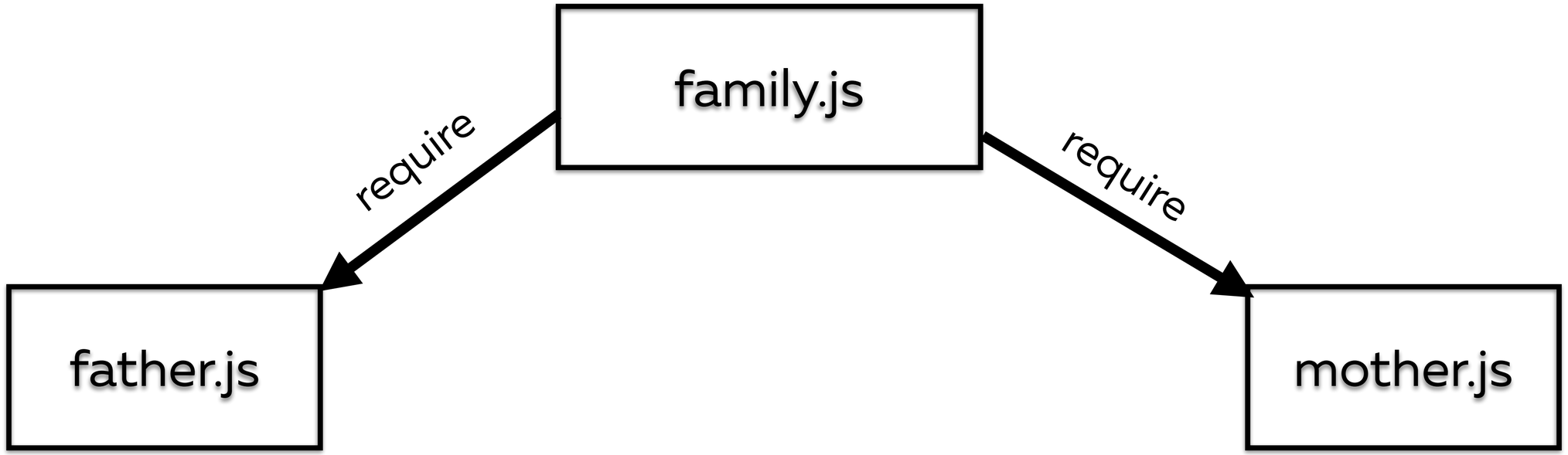
- Никогда не кладите объект в переменную *exports*.
  Лучше всегда использовать *module.exports*
- Никогда не экспортируйте значения асинхронно или внутри условий, т.к. они могут оказаться непроинициализированы
- Расширение файла в зависимости можно опустить

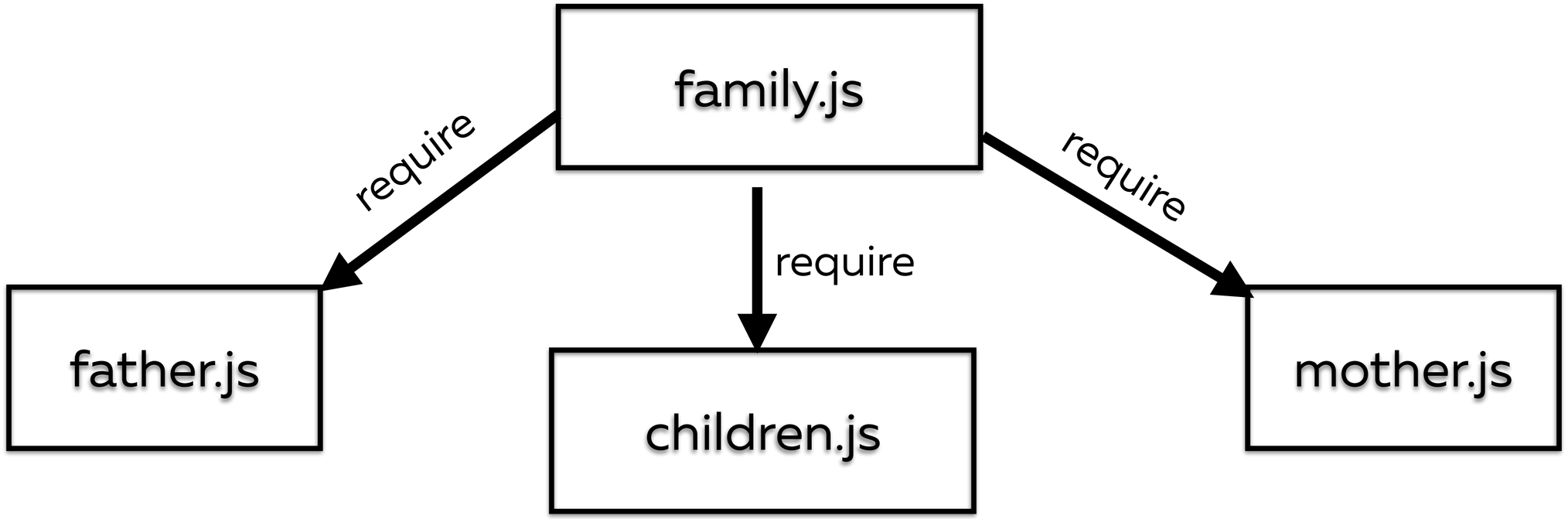
Ограничения модулей

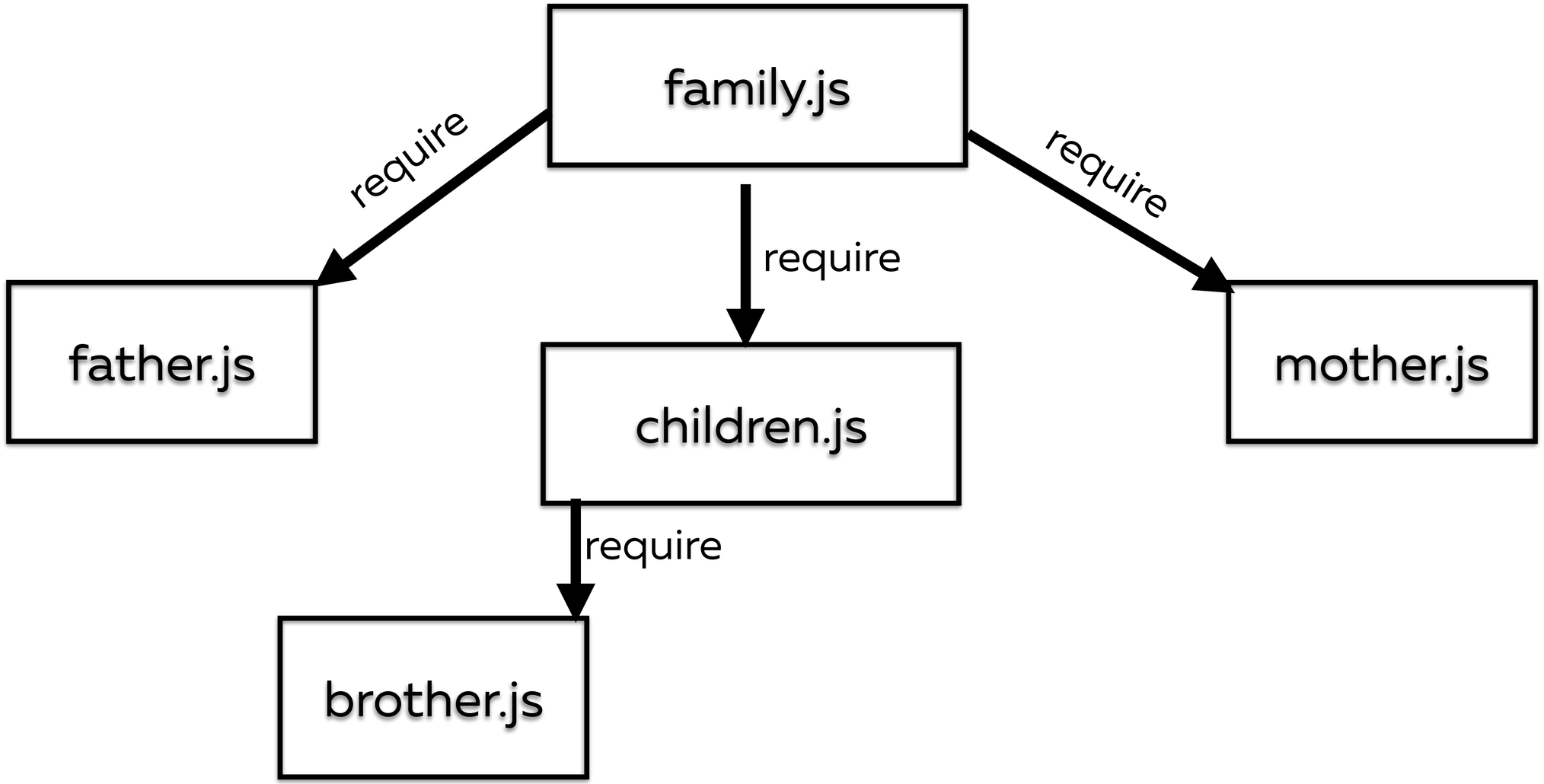
- Никогда не кладите объект в переменную *exports*.
  Лучше всегда использовать *module.exports*
- Никогда не экспортируйте значения асинхронно или внутри условий, т.к. они могут оказаться непроинициализированы
- Расширение файла в зависимости можно опустить
- Избегайте циклов

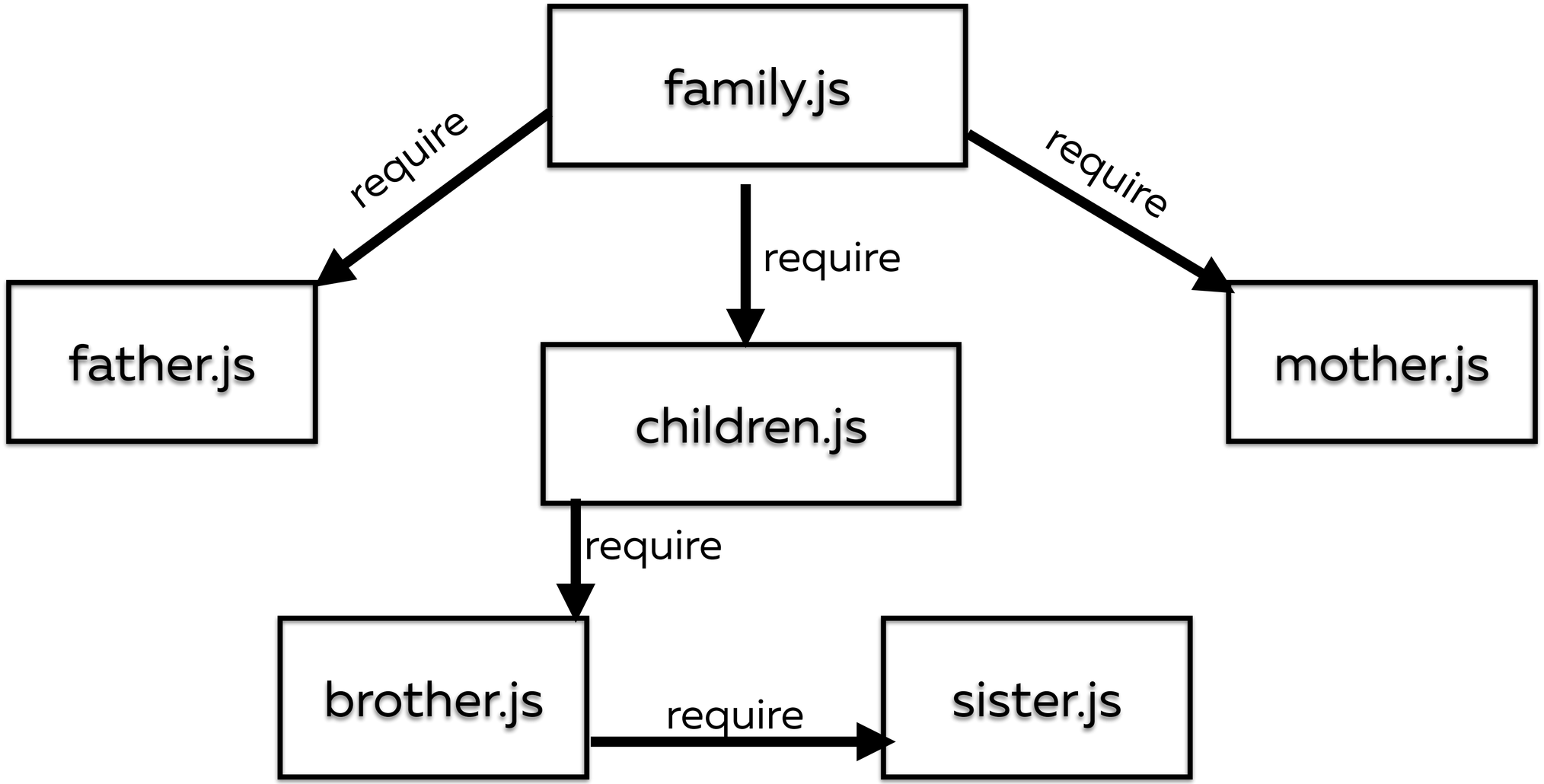

family.js

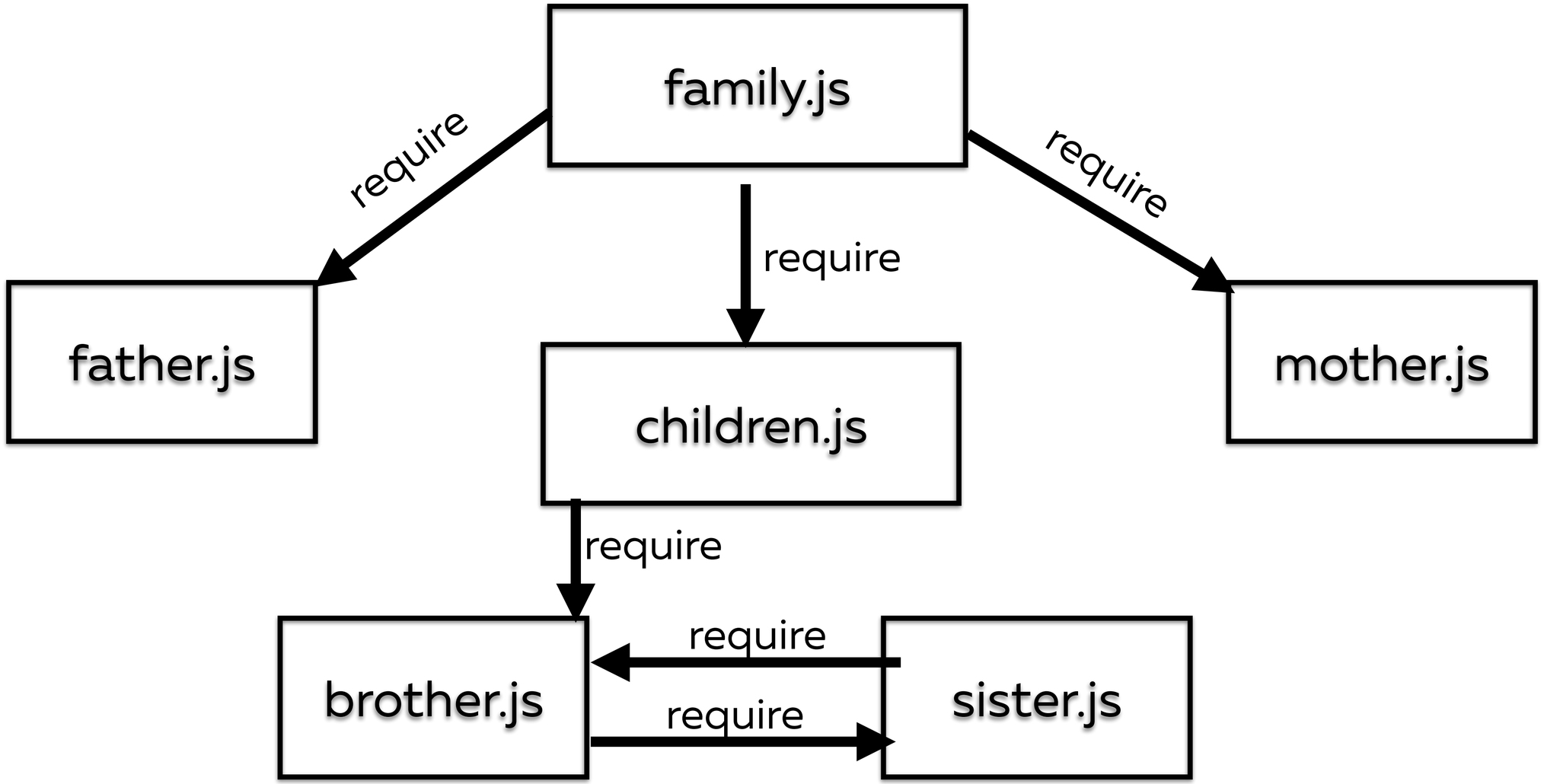

family.js
require
father.js

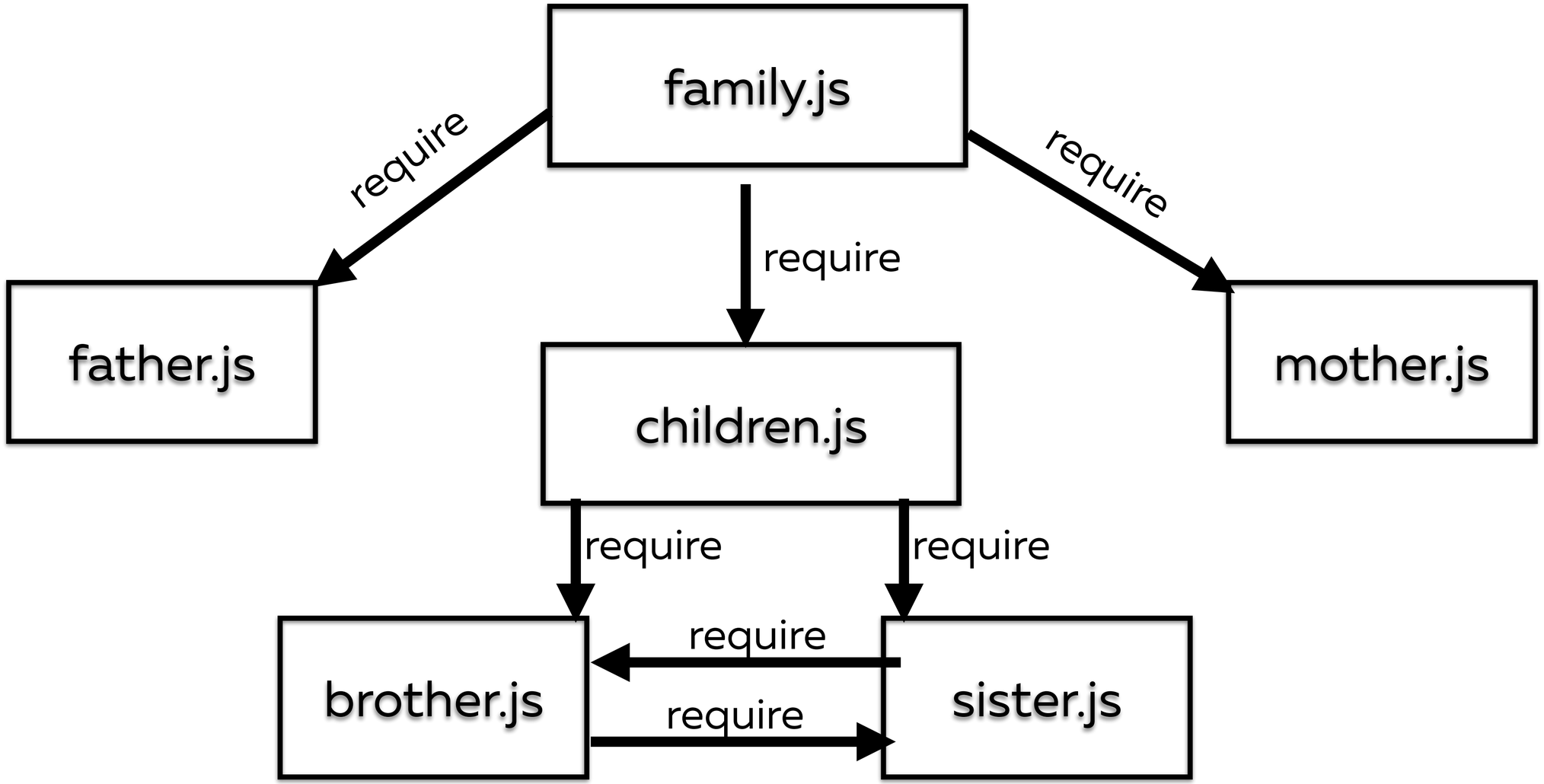

family.js
require
require
father.js
mother.js


family.js
require
require
require
father.js
children.js
mother.js


family.js
require
require
require
father.js
children.js
mother.js
require
brother.js


family.js
require
require
require
father.js
children.js
mother.js
require
brother.js
require
sister.js


family.js
require
require
require
father.js
children.js
mother.js
require
require
require
brother.js
sister.js


family.js
require
require
require
father.js
children.js
mother.js
require
require
require
require
brother.js
sister.js

NPM (Node Package Manager)

пакетный менеджер — программа позволяющая управлять завимисимостями вашего проекта

Какие задачи решает NPM

Какие задачи решает NPM

– Описание вашего проекта

Какие задачи решает NPM

– Описание вашего проекта

– Управление внешними зависимостями в проектах:

Какие задачи решает NPM

- Описание вашего проекта
- Управление внешними зависимостями в проектах:
  - на стадии разработки проекта (*devDependency*)

Какие задачи решает NPM

- Описание вашего проекта
- Управление внешними зависимостями в проектах:
  - на стадии разработки проекта (*devDependency*)
  - на стадии работы проекта (*dependency*)

Демо

Создадим свой NPM-пакет

Создание нового проекта NPM

Создание нового проекта NPM

– *npm init*

Создание нового проекта NPM

– *npm init*

– Обязательные поля:

Создание нового проекта NPM

- *npm init*
- Обязательные поля:
  - Имя пакета (*package name*)

Создание нового проекта NPM

- *npm init*
- Обязательные поля:
  - Имя пакета (*package name*)
  - Версия (*version*)

Создание нового проекта NPM

- *npm init*
- Обязательные поля:
  - Имя пакета (*package name*)
  - Версия (*version*)
  - Описание (*description*)

Создание нового проекта NPM

- *npm init*
- Обязательные поля:
  - Имя пакета (*package name*)
  - Версия (*version*)
  - Описание (*description*)
  - Точка входа (*entry point*)

Создание нового проекта NPM

- *npm init*
- Обязательные поля:
  - Имя пакета (*package name*)
  - Версия (*version*)
  - Описание (*description*)
  - Точка входа (*entry point*)
- Прочие поля: команда для запуска тестов, имя владельца, лицензия...

Создание нового проекта NPM

- *npm init*
- Обязательные поля:
  - Имя пакета (*package name*)
  - Версия (*version*)
  - Описание (*description*)
  - Точка входа (*entry point*)
- Прочие поля: команда для запуска тестов, имя владельца, лицензия...
- Чтобы создать проект по умолчанию *npm init --yes*

Рыба

```
{
  "name": “@intensive-nodejs/codeandmagick”,
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Демо

Добавим зависимости

npm install (npm i)

npm install (npm i)

– без аргументов — устанавливает все зависимости прописанные в *package.json*

npm install (npm i)

– без аргументов — устанавливает все зависимости прописанные в *package.json*

– *<package-name>* — устанавливает новую зависимость/зависимости в *node_modules*

npm install (npm i)

- без аргументов — устанавливает все зависимости прописанные в *package.json*
- *<package-name>* — устанавливает новую зависимость/зависимости в *node_modules*
- *--save (-S)* — устанавливает зависимости и прописывает их в *dependencies* в package.json

npm install (npm i)

– без аргументов — устанавливает все зависимости прописанные в *package.json*

– *<package-name>* — устанавливает новую зависимость/зависимости в *node_modules*

– *--save (-S)* — устанавливает зависимости и прописывает их в *dependencies* в package.json

– *--save-dev (-D)* — устанавливает зависимости и прописывает их в *devDependencies* в package.json

npm install (npm i)

– без аргументов — устанавливает все зависимости прописанные в *package.json*

– *<package-name>* — устанавливает новую зависимость/зависимости в *node_modules*

– *--save (-S)* — устанавливает зависимости и прописывает их в *dependencies* в package.json

– *--save-dev (-D)* — устанавливает зависимости и прописывает их в *devDependencies* в package.json

– *--exact (-E)* — прописывает фиксированную *semver* версию в зависимости

Пакеты/проекты NPM

Пакеты/проекты NPM

– Публичные — размещенные на публичных серверах (реестрах NPM) и доступны всем в интернете. Самый популярный и крупный — https://www.npmjs.com

Пакеты/проекты NPM

– Публичные — размещенные на публичных серверах (реестрах NPM) и доступны всем в интернете. Самый популярный и крупный — https://www.npmjs.com

– Приватные — размещенные на приватных серверах, либо доступные только с проверкой прав доступа на публичных серверах

Пакеты/проекты NPM

- Публичные — размещенные на публичных серверах (реестрах NPM) и доступны всем в интернете. Самый популярный и крупный — https://www.npmjs.com
- Приватные — размещенные на приватных серверах, либо доступные только с проверкой прав доступа на публичных серверах
- Неопубликованные (непубликуемые) — пакеты, которые не нуждаются в публикации, например, конечные приложения, сайты и т.д.

Именование

Именование

– В рамках одного NPM реестра (NPM registry) имя пакета должно быть уникальным

Именование

– В рамках одного NPM реестра (NPM registry) имя пакета должно быть уникальным

– Допускается объединять пакеты в логические пространства имён (scope)

Именование

– В рамках одного NPM реестра (NPM registry) имя пакета должно быть уникальным

– Допускается объединять пакеты в логические пространства имён (scope)

– Отсутствует премодерация пакетов

Версионирование

7.6.54

1 2 3

1 — мажорная версия

2 - минорная версия

3 - патч-версия

Установка пакетов

Установка пакетов

– Команда:

npm i <package name>

Установка пакетов

– Команда:

  npm i <package name>

– Пакеты ставятся по умолчанию в папку *node_modules* текущего проекта

Установка пакетов

- Команда:
  npm i <package name>
- Пакеты ставятся по умолчанию в папку *node_modules* текущего проекта
- Чтобы связать пакет как необходимый для этого проекта его можно прописать в package.json или использовать специальные ключи:

Установка пакетов

- Команда:
  npm i <package name>
- Пакеты ставятся по умолчанию в папку *node_modules* текущего проекта
- Чтобы связать пакет как необходимый для этого проекта его можно прописать в package.json или использовать специальные ключи:
  - *--save* — сохраняет зависимость как необходимую для запуска

Установка пакетов

- Команда:
  npm i <package name>
- Пакеты ставятся по умолчанию в папку *node_modules* текущего проекта
- Чтобы связать пакет как необходимый для этого проекта его можно прописать в package.json или использовать специальные ключи:
  - *--save* — сохраняет зависимость как необходимую для запуска
  - *--save-dev* — сохраняет зависимость как зависимость времени разработки

Пример установленных зависимостей

```
"dependencies": {
  "colors": "^1.1.2",
  "commander": "^2.11.0"
},
"devDependencies": {
  "mocha": "^3.5.3"
}
```

Семантическое версионирование (semver)

Семантическое версионирование (semver)

– Версия зафиксирована – 1.2.3

Семантическое версионирование (semver)

– Версия зафиксирована — 1.2.3

– Разрешает автоматически обновлять патч-версию — 1.2 или 1.2.x или ~1.2.3

Семантическое версионирование (semver)

- Версия зафиксирована — 1.2.3
- Разрешает автоматически обновлять патч-версию — 1.2 или 1.2.x или ~1.2.3
- Разрешается автоматически обновить минорную версию и патч версию — 1 или 1.x или ^1.2.3

Семантическое версионирование (semver)

- Версия зафиксирована — 1.2.3
- Разрешает автоматически обновлять патч-версию — 1.2 или 1.2.x или ~1.2.3
- Разрешается автоматически обновить минорную версию и патч версию — 1 или 1.x или ^1.2.3
- Разрешает обновить на последнюю мажорную версию — * или x

Семантическое версионирование (semver)

- Версия зафиксирована — 1.2.3
- Разрешает автоматически обновлять патч-версию — 1.2 или 1.2.x или ~1.2.3
- Разрешается автоматически обновить минорную версию и патч версию — 1 или 1.x или ^1.2.3
- Разрешает обновить на последнюю мажорную версию — * или x
- Разрешает обновить на версию равную или большей исходной — >=1.2.3

Семантическое версионирование (semver)

- Версия зафиксирована — 1.2.3
- Разрешает автоматически обновлять патч-версию — 1.2 или 1.2.x или ~1.2.3
- Разрешается автоматически обновить минорную версию и патч версию — 1 или 1.x или ^1.2.3
- Разрешает обновить на последнюю мажорную версию — * или x
- Разрешает обновить на версию равную или большей исходной — >=1.2.3
- Разрешает использовать любую версию, но не старше заданной — <=1.2.3

Дерево зависимостей (npm list)

```
└─┬ mocha@3.5.3
  ├── browser-stdout@1.3.0
  ├─┬ commander@2.9.0
  │ └── graceful-readlink@1.0.1
  ├─┬ debug@2.6.8
  │ └── ms@2.0.0
  ├── diff@3.2.0
  ├── escape-string-regexp@1.0.5
  ├─┬ glob@7.1.1
  │ ├── fs.realpath@1.0.0
  │ ├─┬ inflight@1.0.6
  │ │ ├── once@1.4.0 deduped
  │ │ └── wrappy@1.0.2
  │ ├── inherits@2.0.3
  │ ├─┬ minimatch@3.0.4
  │ │ └─┬ brace-expansion@1.1.8
  │ │   ├── balanced-match@1.0.0
  │ │   └── concat-map@0.0.1
  │ ├─┬ once@1.4.0
  │ │ └── wrappy@1.0.2 deduped
  │ └── path-is-absolute@1.0.1
  ├── growl@1.9.2
  ├── he@1.1.1
  ├── json3@3.3.2
  ├─┬ lodash.create@3.1.1
  │ ├─┬ lodash._baseassign@3.2.0
  │ │ ├── lodash._basecopy@3.0.1
  │ │ └─┬ lodash.keys@3.1.2
```

Использование модулей

```
require(`colors`);

console.log(`hello`.green); // outputs green text
console.log(`i like cake and pies`.underline.red); // outputs red underlined text
console.log(`inverse the color`.inverse); // inverses the color
console.log(`OMG Rainbows!`.rainbow); // rainbow
console.log(`Run the trap`.trap); // Drops the bass

console.log(`\x1b[1;31mbold red text\x1b[0m\\n`);
```

Алгоритм загрузки модулей

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля
    1. загрузить модуль

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля
    1. загрузить модуль
2. Если путь начинается с '/'

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля
    1. загрузить модуль
2. Если путь начинается с '/'
    1. загрузить модуль по абсолютному пути

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля
    1. загрузить модуль
2. Если путь начинается с '/'
    1. загрузить модуль по абсолютному пути
3. Если путь начинается с './' или '../'

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля
    1. загрузить модуль
2. Если путь начинается с '/'
    1. загрузить модуль по абсолютному пути
3. Если путь начинается с './' или '../'
    1. попробовать загрузить как файл относительно текущего модуля

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля
    1. загрузить модуль
2. Если путь начинается с '/'
    1. загрузить модуль по абсолютному пути
3. Если путь начинается с './' или '../'
    1. попробовать загрузить как файл относительно текущего модуля
    2. попробовать загрузить как папку

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля
    1. загрузить модуль
2. Если путь начинается с '/'
    1. загрузить модуль по абсолютному пути
3. Если путь начинается с './' или '../'
    1. попробовать загрузить как файл относительно текущего модуля
    2. попробовать загрузить как папку
4. Попробовать загрузить как зависимость

Алгоритм загрузки модулей

1. Если путь совпадает с именем встроенного модуля
    1. загрузить модуль
2. Если путь начинается с '/'
    1. загрузить модуль по абсолютному пути
3. Если путь начинается с './' или '../'
    1. попробовать загрузить как файл относительно текущего модуля
    2. попробовать загрузить как папку
4. Попробовать загрузить как зависимость
5. Выкинуть ошибку, что такой модуль не найден

Run-скрипты

Run-скрипты

– Позволяют автоматизировать стандартные процессы:

Run-скрипты

– Позволяют автоматизировать стандартные процессы:

  – сборку

Run-скрипты

- Позволяют автоматизировать стандартные процессы:
  - сборку
  - тестирование

Run-скрипты

- Позволяют автоматизировать стандартные процессы:
  - сборку
  - тестирование
  - запуск кода

Run-скрипты

- Позволяют автоматизировать стандартные процессы:
  - сборку
  - тестирование
  - запуск кода
  - любые дополнительные задачи

Run-скрипты

- Позволяют автоматизировать стандартные процессы:
  - сборку
  - тестирование
  - запуск кода
  - любые дополнительные задачи
- *Run*-скрипты прописываются в *package.json* в поле *scripts*

Run-скрипты

- Позволяют автоматизировать стандартные процессы:
  - сборку
  - тестирование
  - запуск кода
  - любые дополнительные задачи
- *Run*-скрипты прописываются в *package.json* в поле *scripts*
- *Run*-скрипт запускается командой: *npm run <script name>*

Run-скрипты

- Позволяют автоматизировать стандартные процессы:
  - сборку
  - тестирование
  - запуск кода
  - любые дополнительные задачи
- *Run*-скрипты прописываются в *package.json* в поле *scripts*
- *Run*-скрипт запускается командой: *npm run <script name>*
- Некоторые стандартные скрипты имеют сокращения, например:
  npm run test === npm test === npm t

Зачем нужен пакетный менеджер

Зачем нужен пакетный менеджер

– Переиспользование кода

Зачем нужен пакетный менеджер

– Переиспользование кода

– Автоматизация стандартных операции в проекте

Зачем нужен пакетный менеджер

– Переиспользование кода

– Автоматизация стандартных операции в проекте

– Удобство отслеживания изменений и обновлений

Зачем нужен пакетный менеджер

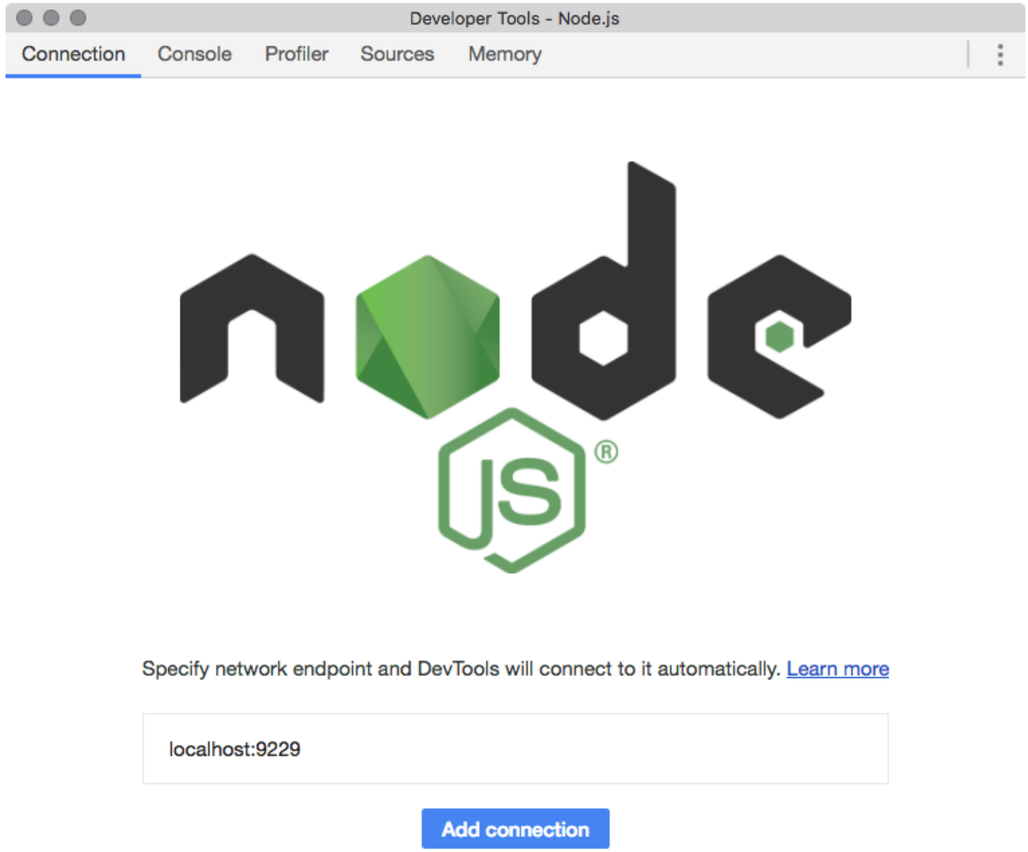
– Переиспользование кода

– Автоматизация стандартных операции в проекте

– Удобство отслеживания изменений и обновлений

– Разделение кода на модули

Debugger


Developer Tools - Node.js
Connection
Console
Profiler
Sources
Memory
Specify network endpoint and DevTools will connect to it automatically. Learn more
localhost:9229
Add connection

node inspect

node inspect

– Встроенный Node.js текстовый дебаггер

node inspect

– Встроенный Node.js текстовый дебаггер

– Запускается из командной строки: node inspect <js-file-entry-point> [params]

Chrome DevTools

medium.com/@paul_irish/debugging-node-js-nightlies-with-chrome-devtools-7c4a1b95ae27

Chrome DevTools

– Позволяет использовать отладчик встроенный в браузер *Chrome*

medium.com/@paul_irish/debugging-node-js-nightlies-with-chrome-devtools-7c4a1b95ae27

Chrome DevTools

– Позволяет использовать отладчик встроенный в браузер *Chrome*

– Требует установленный браузер *Chrome*

medium.com/@paul_irish/debugging-node-js-nightlies-with-chrome-devtools-7c4a1b95ae27

Chrome DevTools

– Позволяет использовать отладчик встроенный в браузер *Chrome*

– Требует установленный браузер *Chrome*

– Запустить программу в отладочном режиме: *node --inspect*

medium.com/@paul_irish/debugging-node-js-nightlies-with-chrome-devtools-7c4a1b95ae27

Chrome DevTools

- Позволяет использовать отладчик встроенный в браузер *Chrome*
- Требует установленный браузер *Chrome*
- Запустить программу в отладочном режиме: *node --inspect*
- Запустить программу и остановить выполнение до подключения отладчика: *node --inspect-brk*

medium.com/@paul_irish/debugging-node-js-nightlies-with-chrome-devtools-7c4a1b95ae27

Chrome DevTools

– Позволяет использовать отладчик встроенный в браузер *Chrome*

– Требует установленный браузер *Chrome*

– Запустить программу в отладочном режиме: *node --inspect*

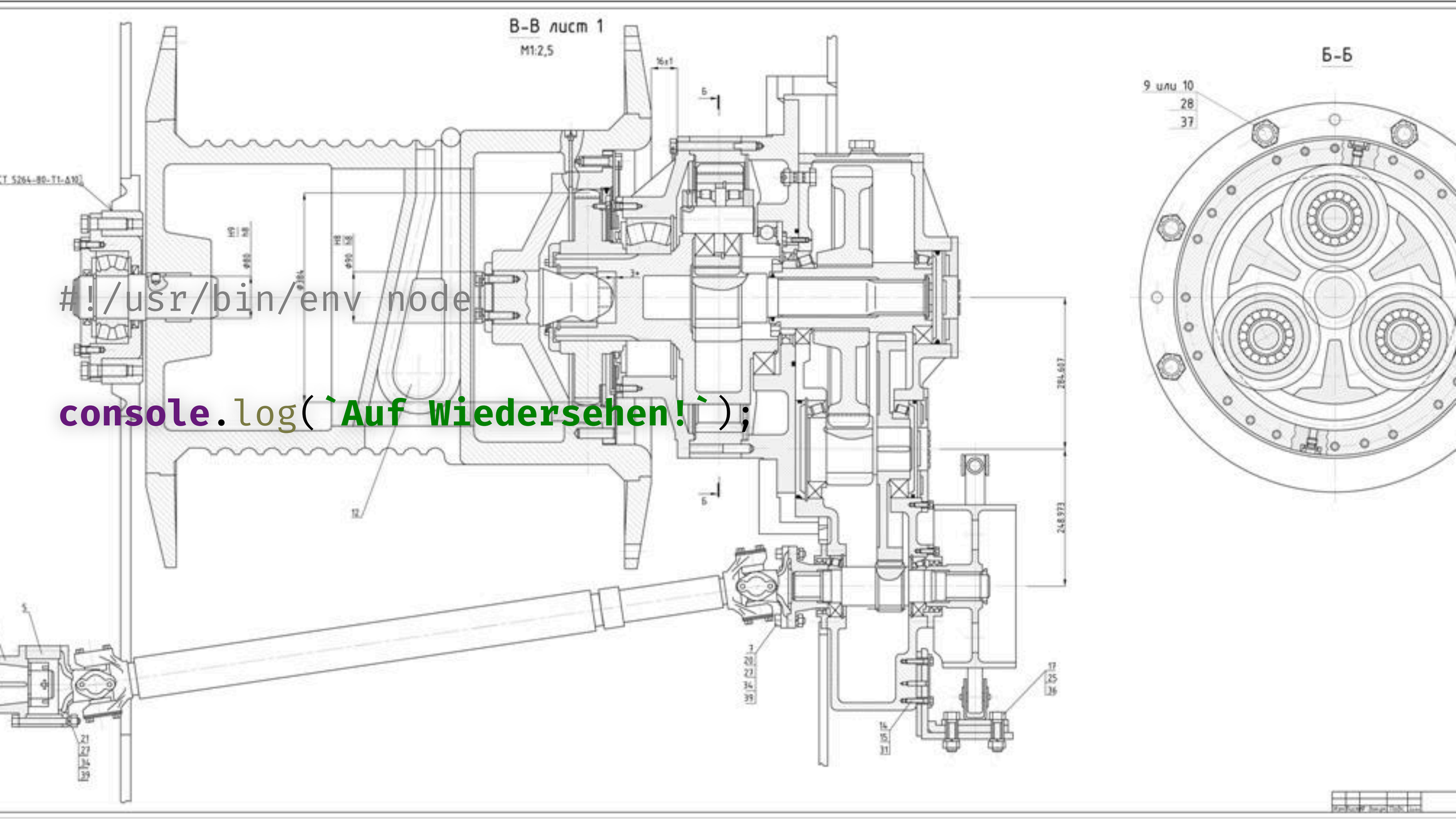
– Запустить программу и остановить выполнение до подключения отладчика: *node --inspect-brk*

– После запуска идём в *Chrome* и открываем новую вкладку с адресом: *chrome://inspect*

medium.com/@paul_irish/debugging-node-js-nightlies-with-chrome-devtools-7c4a1b95ae27

Встроенный отладчик в редакторе

Некоторые редакторы так же предоставляют свой отладчик для отладки кода. Но процесс запуска и отладки кода в каждом случае индивидуален для каждого редактора




```
#!/usr/bin/env node

console.log(`Auf Wiedersehen!`);
```