2. ОСНОВНЫЕ ПАТТЕРНЫ NODE.JS

ПРОГРАММИРОВАНИЕ	
Синхронное (последовательность шагов)	**Асинхронное** (выполнение в “фоновом режиме”; следующая операция выполняется сразу, даже если предыдущая не закончена)
• Код легко понять.	• Высокая производительность.
• Каждая операция блокирует цикл.	• В более сложных приложениях необходимы более сложные механизмы контроля.
	Node.js решает обе задачи.

2.1 Коллбэки

- аргумент функции, вызываемые с результатом функции
- воплощения паттерна Реактор,
- передают результат,
- используют замыкания.
- в функциональном программировании = Continuation-Passing Style (CPS) в отличие от Direct Style, при котором результат возвращается при помощи *return*.

Синхронные	Асинхронные
См. 02-01-sync-callbacks.js	**См.:** • **02-02-async-callbacks.js** • **02-03-async-callbacks.js**

2.2.1 Непоследовательная функция на коллбэках

- синхронное и асинхронное поведение в зависимости от условий.
- **См. 02-04-unpredictable-function.js — 02-05 -unpredictable-function.js**
- функция должна быть последовательной: или синхронной, или асинхронной:
  - синхронные блокируют цикл; в синхронной использовать **Прямой стиль**: улучшает читаемость и производительность. См. **02-06-unpredictable-function-sync.js**
  - асинхронная при помощи **Отложенного исполнения:** *process.nexttick()* или *setImmediate()*. См. **02-07-unpredictable-function-async.js**

2.2.2 Правила коллбэков

- если аргумент функции, всегда последний,
- ошибка всегда первый аргумент коллбэка; проброс ошибки при помощи передачи в коллбэк. **После кого как было поймана необработанная ошибка, необходимо завершить приложение поскольку его работа нестабильная.** См. **02-08-errors.js — 02-10-errors.js.**

2.2 Модули

- кирпичики для построения приложений

- способ сокрытия переменных и функций

2.2.1 Метод проектирования Модуль

- нет неймспейсов
- чтобы предотвратить попадание переменных в глобальную область видимости используются самовызывающиеся функции

2.2.2 require

Этот метод лежит в основе *require.* Все, что внутри модуля — приватно. Содержимое переменной module.exports кэшируется и возвращается при повторном вызове require. **См. 02-12-require.js.**

В module есть переменная global: все, что записывается в нее попадает в глобальную область видимости, но не следует ее засорять.

module.exports	exports
Непосредственно объект экспорта.	Ссылка на module.exports, поэтому возможно только запись новых свойств.

require — синхронная функция, след. запись в module.exports должно быть синхронным.

Поисковой алгоритм require:

1. Файловые модули:
   - / — абсолютный путь: возвращается как есть
   - ./ — относительный путь: высчитывается
2. Модули ядра Node.js
3. Модули-пакеты — первая папка node_modules вверх по директориям до корневой папки ФС

Файловые модули и модули пакеты:

1. *{moduleName}.js*
2. *{moduleName}/index.js*
3. папка/имя в *main*-свойстве *{moduleName}/package.json*

** require.resolve()*

Этот алгоритм позволяет избежать Dependency Hell.

Кэширование модулей:

Модуль загружается один раз:

- цикличность?
- постоянство модуля

** require.cache (удалить модуль из кэша по его ключу)*

См. 02-13-circular-require.

2.2.3 Объявление модулей

1. Именованный экспорт: присвоение в exports; единственно разрешенный CommonJS
2. Перезапись module.exports в функцию (**Substack Pattern**):
   - экспорт одного функционала
   - четкая точка входа
   - разделение на важное и вторичное
   - выполнение Single Responsibility Principle: всякий модуль выполняет только одно действие, которые должно быть сокрыто в модуле

См. 02-14-substack-pattern.

3. Экспорт конструктора:
   - создать экземпляры
   - расширить прототип
   - создать новые классы на основе экспортируемого

?	Можно ли добавить методы в ES2015 классы?
?	Как проверить, что конструктор не был вызван как функция?

4. Экспорт экземпляра: почти равно **Singleton Pattern**, но не гарантирует уникальность экземпляра во всем приложении. В дополнение можно экспортировать сам конструктор: разделение на важное и вторичное.

Дополнительная литература:

- https://blog.izs.me/2013/08/designing-apis-for-asynchrony
- https://learn.javascript.ru/closures-module