Урок N1

Знакомство с Node.js

Содержание урока

- Компоненты платформы Node.js
- Обзор возможностей Node.js
- Асинхронная архитектура Node.js
- Как работают программы в Node.js
- Установка и запуск Node.js
- Hello, world!
- Менеджеры версий
- Менеджеры процессов
- Система управления пакетами NPM
- Документация

Компоненты платформы Node.js

Node или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код) и превращающая JavaScript из узко специализированного языка в язык общего назначения.

Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода/вывода через свой API (написанном на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript кода.

Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи node-webkit и AppJS для Linux, Windows и MacOS) и даже программировать микроконтроллеры (например, tessel и espruino).

В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Node.js является универсальной платформой, позволяющей решать широкий круг задач. Тем не менее, у Node.js есть несколько сильных сторон, из-за которых платформа чаще всего используется для написания либо серверных, либо консольных приложений.

Сильной стороной Node.js в плане создания сетевых приложений является крайне быстрая работа с сетевыми соединениями и способность обрабатывать сотни тысяч одновременных подключений к серверу при небольшом потреблении ресурсов.

По сравнению с популярными синхронными языками и платформами (например, PHP) это дает Node.js колоссальное преимущество при разработке приложений, в которых нужно в реальном времени обеспечить обмен сообщениями между большим количество пользователей или которые обрабатывают и генерируют большое количество входящих и исходящих соединений.

Типы приложений, которые часто разрабатываются на Node.js:

- Чаты и системы обмена мгновенными сообщениями
- Многопользовательские игры в реальном времени
- Системы одновременной работы над проектами
- Сетевые сервисы для сбора и отправки информации

Также Node.js хорошо подходит для создания «обычных» веб-приложений. С учетом того, что платформу Node.js легко установить и запустить на любом настольном компьютере (Windows, Linux, Mac), ее часто используют для создания консольных утилит. Например, с помощью Node.js созданы популярные системы сборки веб-проектов Grunt.js и Gulp.js

Особенности синхронной архитектуры на примере PHP:

- Один входящий запрос обрабатывается одним экземпляром скрипта.
- Каждый раз при запуске нового экземпляра скрипта заново выделяется память, подгружаются билиотеки и т.п., а при окончании - удаляются из памяти
- Все экземпляры независимы друг от друга и поэтому ошибки в одно экземпляре не влияют напрямую на другие экземпляры. Обмен данными между скриптами можно организовать через файлы, базу данных, сетевые запросы, memcached и т.п.
- Все команды выполняются строго последовательно. Невозможно перейти к выполнению следующей команды до завершения предыдущей и параллельно выполнять две команды

Особенности асинхронной архитектуры на примере PHP:

- Одна программа на Node.js обрабатывает множество запросов. Поэтому критическая ошибка при обработке одного запроса может привести к краху всей программы и необходимости ее перезапуска
- Синхронные (блокирующие) команды выполняются последовательно
- Асинхронные (неблокирующие) команды выполняются параллельно и при завершении генерируют события, на которые программа может реагировать с помощью функций-обработчиков
- Последовательность выполнения программы зависит от последовательности произошедших событий, на которые она реагирует после того, как обработала предыдущие события

Принцип работы похож на выполнение JavaScript скриптов в браузере. При загрузке скрипта в браузере он может выполнять операции и регистрировать обработчики различных асинхронных событий, которые браузер может генерировать, например, клики по кнопкам и т.п.

В Node.js принцип тот же с одним отличием - событиями могут быть результаты выполнения команд ввода-вывода, которые запускает на выполнение в том числе сам скрипт. Например, скрипт может запустить команду чтения большого файла и зарегистрировать обработчик, который выполнится после завершения этого действия. Также скрипт может регистрировать обработчики для «внешних» событий - например, поступающих сетевых запросов.

Всю сложную работу по организации асинхронного выполнения команд ввода-вывода и обработке событий берет на себя Node.js, в самом скрипте остается только запускать нужные команды и регистрировать обработчики нужных событий, аналогично тому, как это делается при написании JavaScript-скриптов для браузера.

Установка и запуск Node.js

Самый простой способ установить Node.js - загрузить инсталлятор с официального сайта http://nodejs.org Инсталляторы, бинарные файлы и исходники есть под все настольные системы - Windows, *nix, Mac OS.

Предпочтительнее всего вести разработку на Node.js в *nix системах или Mac OS, т.к. поддержка многих модулей под windows оставляет желать лучшего популярных модул ряда полезных модулей под windows оставляет желать лучшего.

Традиционно для использования программ на Node.js их запускают на Linux-серверах, в частности на популярных ОС Ubuntu или CentOS.

После установки Node.js вы сможете открыть терминал (для windows - набрав cmd.exe в строке выполнения программ) и выполнить в нем команду **node -v**, которая покажет текущую установленную версию.

Не забывайте следить за новостями на nodejs.org и периодически обновлять версию node, т.к. разработка идет довольно быстрыми темпами и новые обновления часто содержат важные исправления или значительные улучшения в плане быстродействия.

Hello, world!

Для запуска первой программы наберите в консоли: **node**. Вы попадете в интерактивную консоль node, прямо в которой можно набирать и выполнять команды. Наберите строчку ниже и затем нажмите Enter:

console.log("Hello, World");

После этого в консоли вы увидите приветствие и в следующей строке undefined в качестве возвращаемого этой функцией значения.

Для завершения сеанса в интерактивной консоли node, нажмите Ctrl +C, при необходимости дважды (для Mac OS - Cmd+C).

Также можно записать эту же команду в файл hello.js и запустить скрипт, указав команде node путь к файлу:

node hello.js

Менеджеры версий

При профессиональной разработке для тестирования или участия в нескольких проектах может понадобиться возможность переключения между различными версиями Node.js на одном компьютере.

Для этого рекомендуется использовать менеджеры версий Node.js вместо того, чтобы удалять старые версии и устанавливать новые.

Для windows это nodist или nvwm, для *nix и Mac OS это n или nvm. Все эти менежеры можно установить с помощью системы управления пакетами NPM, о которой будет рассказано далее в этом уроке.

Менеджеры процессов

Для более удобного запуска программ с помощью node существуют менеджеры процессов. Это программы, которые позволяют легко настроить автозапуск тех или иных программ на node при загрузке системы и их автоматический перезапуск при неожиданном завершении работы из-за критических ошибок.

Также менеджеры процессов позволяют удобно настроить логирование стандартных потоков вывода в файлы, предоставляют более удобный интерфейс для оперативного просмотра логов и общего состояния запущенных через node скриптов.

Популярные менеджеры процессов pm2 и forever разработаны для *nix систем и также устанавливаются через NPM. Для просмотра списка команд менджеров достаточно набрать в консоли pm2 или forever.

Наиболее популярные команды для pm2:

pm2 start app.js - запустит app.js
pm2 stop app - остановит скрипт с именем app
pm2 restart app - перезапустит скрипт с именем app
pm2 delete app - остановит скрипт и удалит из списка процессов
pm2 logs app - будет выводить логи скрипта app (Ctrl+C для остановки)
pm2 monit - будет выводить нагрузку скриптов (Ctrl+C для остановки)
pm2 list - выведет список управляемых процессов
pm2 startup ubuntu или pm2 startup centos - настроит автозапуск запущенных скриптов при перезагрузке системы ubuntu или centos.

Для подключения к вашим скриптам дополнительных функций в Node.js существует удобная система управления модулями NPM. По сути это публичный репозиторий (аналог магазина приложений для Google Chrome, iOS или Android) написанных для node.js дополнительных программных модулей.

Команда npm позволяет легко устанавливать, удалять или обновлять нужные вам модули, автоматически учитывая при этом все зависимости выбранного вами модуля от других. Например, вот так можно установить модуль pm2 на *nix системах **npm install pm2 -g**

NPM автоматически загрузит и установит все зависимости для данного модуля. Ключ -g означает, что модуль будет установлен и доступен глобально, т.е. всем программам на node, т.к. является универсальным и команда pm2 также должна быть доступна везде в системе.

Если какой-либо модуль нужно установить локально, т.е. только для данной программы, опустите ключ -g и для установки модуля будет использована поддиректория node_modules. Таким образом разные программы могут хранить разные независимые наборы всех необходимых для запуска зависимостей в директории node_modules.

Увидеть список локально установленных модулей можно с помощью команды **npm list**, а список глобальных с помощью **npm list -g**.

Хотя node_modules и содержит все необходимые для запуска зависимости, распространять исходный код вместе с ней может быть неудобно, т.к. в ней может храниться большое количество файлов, которые занимают ощутимый объем.

С учетом того, что все публичные NPM модули можно легко установить с помощью npm, достаточно написать для вашей программы файл **package.json** с перечнем всех необходимых для работы зависимостей и потом просто установить все нужные модули командой **npm install**.

Интерактивное руководство по составлению package.json и его секциям (ни одна из которых не является обязательной) вы найдете на странице http://package.json.nodejitsu.com/ - просто наведите курсор на нужную секцию, например dependencies.

Создайте и запустите файл hello-npm.js:

```
var ansi = require("ansi");
var cursor = ansi(process.stdout);
cursor.beep();
```

Запустите скрипт:

node hello-npm.js

Вы увидите ошибку:

Error: Cannot find module ‘ansi'

Установите недостающий пакет ansi с помощью команды:

npm install ansi

После выполнения команды вы увидите в текущей директории новую поддиректорию node_modules, куда устанавливаются все модули. После этого вы сможете запустить программу и услышать звук:

node hello-npm.js

Также для автоматической установки пакета ansi с помощью **npm install** можно создать рядом с hello-npm.js файл **package.json**:

```
{"dependencies": {
   "ansi" : "*"
}}
```

Звездочка напротив названия модуля означает, что можно установить любую последнюю доступную версию модуля. При необходимости можно прописать более детальные ограничения на версию модуля. Теперь распространять программу hello-npm.js можно с помощью всего лишь 2 файлов!

Документация

Официальная документация по Node.js API:

http://nodejs.org/api/

Список модулей NPM:

https://www.npmjs.org/
https://nodejsmodules.org/

Дополнительные ресурсы:

http://www.nodebeginner.ru/ ,
http://nodeguide.ru/

Самоконтроль

✓ Из каких компонентов состоит платформа Node.js
✓ Преимущества асинхронной архитектуры перед синхронной
✓ Что такое Event Loop и как он работает на сервере и на клиенте
✓ Зачем нужны менеджеры версий и процессов Node.js
✓ Как используется система управления пакетами NPM

Домашнее задание

1) Установить Node.js и запустить примеры из урока: hello.js в интерактивной консоли, hello.js из файла и hello-npm.js

2) Создать простейшую консольную программу с использованием хотя бы одной функции из стороннего модуля, локально установленного с помощью NPM (модуль должен отличаться от рассмотренного на уроке!).

3) Продвинутый блок: создать с помощью Node.js API консольную программу, которая будет выводить что-либо в консоль разными цветами и издавать звук(и) с помощью модуля или модулей, отличных от рассмотренного на уроке.