Node.js 치트시트

Node.js 핵심 개념

Node.js는 확장성 있는 네트워크 애플리케이션 개발을 위해 설계된 서버 측 JavaScript 실행 환경입니다.

- **Node.js**: Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임 환경입니다.
- **비동기 I/O**: Non-blocking I/O 모델을 채택하여 데이터 집약적인 실시간 애플리케이션에서 높은 성능을 발휘합니다.
- **이벤트 기반**: 이벤트 루프를 통해 비동기 작업을 효율적으로 관리하고 처리합니다.
- **단일 스레드 구조**: 메인 실행 스레드는 하나이지만, 내부적으로는 파일 시스템이나 네트워크 작업 등을 위해 스레드 풀(Libuv)을 활용합니다.
- **NPM (Node Package Manager)**: 세계 최대의 오픈 소스 라이브러리 생태계를 보유한 패키지 관리 도구입니다.

설치 및 버전 관리

- **Node.js 설치**: 공식 홈페이지(https://nodejs.org)에서 최신 LTS 버전을 다운로드하는 것을 권장합니다.
- **버전 확인**: 터미널에서 `node -v` 또는 `node --version`으로 설치된 버전을 확인합니다.
- **NPM 버전 확인**: `npm -v` 명령어를 사용합니다.
- **nvm (Node Version Manager)**: 한 시스템 내에서 여러 버전의 Node.js를 쉽게 설치하고 전환할 수 있게 돕는 도구입니다.
  - 설치: `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash`
  - 버전 설치: `nvm install 18` (18버전 최신 설치), `nvm install node` (최신 안정 버전 설치)
  - 버전 전환: `nvm use 18.17.0`
  - 기본 버전 설정: `nvm alias default 18.17.0`

NPM 기본 명령어

패키지 관리

- **프로젝트 초기화**: `npm init`을 실행하여 `package.json` 파일을 생성합니다. (`-y` 옵션 사용 시 기본값으로 즉시 생성)
- **패키지 설치**: `npm install <패키지명>` (또는 `npm i`)
- **전역 설치**: 시스템 전체에서 명령어로 사용할 패키지는 `-g` 옵션을 붙여 설치합니다.
- **개발 의존성 설치**: 개발 시에만 필요한 도구(테스트 도구, 린터 등)는 `--save-dev` 또는 `-D` 옵션을 사용합니다.
- **프로덕션 의존성 설치**: 애플리케이션 실행에 필요한 패키지는 `--save` 또는 `-S` 옵션을 사용합니다. (기본값)
- **패키지 제거**: `npm uninstall <패키지명>`
- **일괄 설치**: `package.json`에 명시된 모든 의존성을 설치할 때는 인자 없이 `npm install`을 실행합니다.

패키지 정보 및 관리

- **목록 확인**: `npm list` (로컬 설치 목록), `npm list -g` (전역 설치 목록)
- **보안 검사**: `npm audit`으로 설치된 패키지의 보안 취약점을 검사하고 `npm audit fix`로 자동 수정합니다.
- **업데이트**: `npm update`를 통해 패키지를 허용된 범위 내의 최신 버전으로 업데이트합니다.

핵심 기본 모듈

File System (fs)

파일 읽기, 쓰기 등 파일 시스템 조작을 담당합니다. 최근에는 Promise 기반의 API 사용이 권장됩니다.

```
const fs = require('fs');
const fsPromises =
require('fs').promises;

// 비동기 방식 (Callback)
fs.readFile('file.txt', 'utf8', (err,
data) => {
  if (err) throw err;
  console.log(data);
});

// Promise 방식 (Async/Await와 함께 사용
권장)
const data = await
fsPromises.readFile('file.txt', 'utf8');
```

HTTP 모듈

기본적인 웹 서버 기능을 구현할 수 있는 저수준 모듈입니다.

```
const http = require('http');
const server = http.createServer((req,
res) => {
  res.writeHead(200, {'Content-Type':
'text/plain; charset=utf-8'});
  res.end('안녕하세요, Node.js 서버입니
다!');
});
server.listen(3000);
```

Path 모듈

파일이나 디렉토리 경로를 운영체제에 독립적으로 안전하게 처리합니다.

```
const path = require('path');
path.join('/users', 'john', 'docs'); //
경로 결합
path.resolve('file.txt');                //
절대 경로로 변환
path.extname('index.html');              //
확장자 추출 (.html)
```

비동기 프로그래밍 패턴

Node.js 환경에서는 성능을 위해 비동기 패턴을 숙지하는 것이 필수적입니다.

Callback (콜백)

전통적인 방식으로, 작업 완료 시 호출될 함수를 인자로 전달합니다.

```
function fetchData(callback) {
  setTimeout(() => callback(null, '데이터
수신 완료'), 1000);
}
```

Promise (프로미스)

비동기 작업의 미래 결과를 나타내는 객체로, 체이닝을 통해 가독성을 높입니다.

Async/Await (에이싱크/어웨이트)

프로미스를 기반으로 비동기 코드를 동기 코드처럼 직관적으로 작성할 수 있게 해주는 현대적인 방식입니다. (가장 권장됨)

```
async function main() {
  try {
    const result = await
asyncOperation();
    console.log(result);
  } catch (err) {
    console.error('에러 발생:', err);
  }
}
```

Express.js 프레임워크

Node.js를 위한 가장 대중적인 웹 프레임워크로, 미들웨어를 활용한 유연한 라우팅을 제공합니다.

기본 서버 설정

```
const express = require('express');
const app = express();

app.use(express.json());            //
JSON 요청 바디 파싱
app.use(express.static('public')); // 정
적 파일 서빙

app.get('/', (req, res) => {
  res.send('메인 페이지');
});

app.listen(3000, () => console.log('서버
실행 중...'));
```

라우팅 및 파라미터 처리

- `req.params`: URL 경로 내의 변수 접근 (`/users/:id`)
- `req.query`: URL 쿼리 스트링 접근 (`/search?q=keyword`)
- `req.body`: POST 요청 등에서 전달된 바디 데이터 접근

환경 설정 및 시스템 연동

- **dotenv**: `.env` 파일에 정의된 환경 변수를 `process.env`로 로드합니다. 보안이 중요한 설정값을 관리할 때 필수입니다.
- **process 객체**: 실행 중인 프로세스 정보에 접근합니다.
  - `process.env.NODE_ENV`: 현재 실행 환경 (development, production 등)
  - `process.cwd()`: 현재 작업 디렉토리 경로 확인

디버깅 및 모니터링

- **node –inspect**: 내장 디버거를 활성화하여 Chrome DevTools에서 디버깅할 수 있게 합니다.
- **nodemon**: 개발 중 파일 소스가 수정되면 자동으로 서버를 재시작해 주는 유용한 도구입니다.
- **winston**: 프로덕션 환경에서 체계적인 로그 관리를 위해 사용하는 대표적인 로깅 라이브러리입니다.

테스팅 도구

- **Jest**: Facebook에서 만든 올인원 테스트 프레임워크로, 단순하면서도 강력한 기능을 제공합니다.
- **Mocha & Chai**: 유연한 설정이 가능한 테스트 러너(Mocha)와 단언 라이브러리(Chai) 조합입니다.

유용한 패키지 추천

- **Lodash**: 자바스크립트 객체 및 배열 조작을 위한 유틸리티 모음.
- **Axios**: 브라우저와 Node.js를 모두 지원하는 강력한 HTTP 클라이언트.
- **Bcrypt**: 비밀번호 보안 해싱을 위한 라이브러리.
- **JWT (JSON Web Token)**: 토큰 기반 인증 구현 시 사용.
- **ORM/ODM**: Sequelize (SQL), Mongoose (MongoDB) 등을 통해 데이터베이스 작업을 객체 지향적으로 수행합니다.

성능 최적화 및 프로덕션 관리

클러스터링 (Clustering)

단일 스레드인 Node.js가 멀티코어 CPU의 자원을 최대한 활용할 수 있도록 프로세스를 복제하여 관리하는 기법입니다.

스트림 (Streams)

대용량 파일을 처리할 때 메모리 부하를 줄이기 위해 데이터를 조각 단위로 읽고 쓰는 방식입니다.

PM2 프로세스 매니저

프로덕션 환경에서 Node.js 애플리케이션을 안정적으로 운영하기 위한 표준 도구입니다.

- 서비스 무중단 재시작 및 자동 복구 지원
- 클러스터 모드 실행 지원 (`pm2 start app.js -i max`)
- 로그 모니터링 및 리소스 사용량 확인

Docker 컨테이너화

애플리케이션과 실행 환경을 하나로 묶어 어디서나 동일하게 동작하도록 배포하는 현대적인 표준 방식입니다.