노드 내장 객체 알아보기

global

- 노드의 전역 객체
  - 브라우저의 window 같은 역할
  - 모든 파일에서 접근 가능
  - window처럼 생략도 가능
    - console, require도 global의 속성
    - global.console
    - global.require
  - global 객체 내부 확인 : 콘솔에서 global
- global 속성에 값을 대입하면 다른 파일에서도 사용 가능

```
// globalA.js
module.exports = () => global.message;

// globalB.js
const A = require('./globalA')

global.message = '안녕하세요';
console.log(A());
```

console 객체

- 브라우저의 console 객체와 매우 유사
  - 노드에서는 window대신 global 객체안에 들어있음
  - 보통 디버깅을 위해 사용
    - 개발 중 변수에 값이 제대로 들어 있는지 확인
    - 에러 발생시 에러 내용을 콘솔에 표시할때
    - 코드 실행시간을 알아볼때
- 자주 쓰는 친구들
  - console.time, console.timeEnd : 시간 로깅
  - console.error : 에러 로깅
  - console.log : 평범한 로그
  - console.dir : 객체 로깅
  - console.trace : 호출스택 로깅

타이머 메소드

- 타이머 기능을 제공하는 함수로, global 객체에 포함
- 타이머 함수들은 모두 아이디를 반환하며 아이디를 사용해 타이머 취소 가능
- set 메소드에 clear 메서드가 대응 됨
  - set 메소드의 리턴값을 clear 메소드에 넣어 취소
  - set 메소드

        - setTimeout(콜백함수, 밀리초) : 주어진 밀리초 이후에 콜백함수 실행
        - setInterval(콜백함수, 밀리초) : 주어진 밀리초마다 콜백함수를 반복 실행합니다
        - setImmediate(콜백함수) : 콜백함수 즉시 실행
    - clear 메소드
        - clearTimeout(아이디) : setTimeout을 취소
        - clearInterval(아이디) : setInterval 취소
        - clearImmediate(아이디) : setImmediate 취소
- 예제

```
const timeout = setTimeout(() => {
    console.log('1.5초 후 실행');
}, 1500);

const interval = setInterval(() => {
    console.log('1초마다 실행');
}, 1000);

const timeout2 = setTimeout(() => {
    console.log('실행도지 않습니다');
}, 3000);

setTimeout(() => {
    clearTimeout(timeout2);
    clearInterval(interval);
}, 2500);

const immediate = setImmediate(() => {
    console.log('즉시 실행')
})

const immediate2 = setImmediate(() => {
    console.log('실행되지 않습니다');
});

clearImmediate(immediate2);
```

    - setTimeout(함수, 0) 보다 setImmediate가 나음

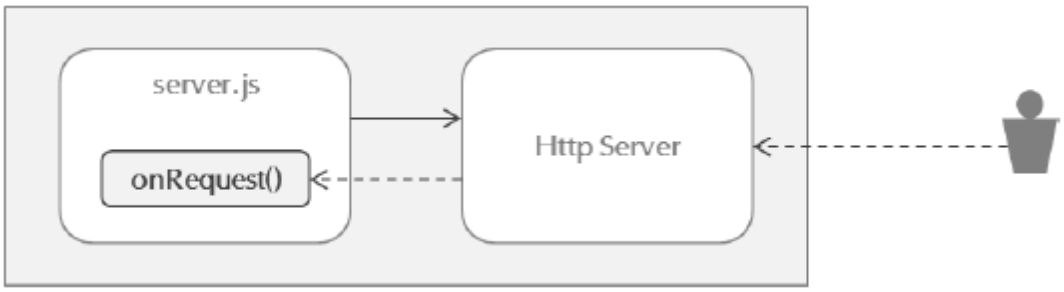
콜백함수

- 함수 속에서 또 다시 함수 콜을 하는 것
- Node.js에서 가장 핵심적인 부분
    - 무엇인가 일을 다른 객체에게 시키고
    - 그 일이 끝나는 것을 기다리는 것이 아니라
    - 그 객체가 나를 다시 부를 때까지 내 할일을 하고 있는 것
- Non-Block 이며 비동기 방식의 함수로 사용
- 우리가 흔히 생각하는 일반적인 함수 : 입력, 출력이 있음
- 자바스크립트에서는 출력값이 없고 그 대신 콜백함수를 입력 받는 함수들이 많이 있음
- 콜백 함수는 다른 함수에 인자로 넘어가서 실행된 로직을 담게 됨

HTTP 서버 콜백 호출

- 비동기 Callback 호출을 위해 onRequest() 함수를 이벤트 리스너로 등록

- request 이벤트에 대한 콜백 열할
- 클라이언트가 http 요청을 보내면 HTTP Server에 request 타입 이벤트가 발생하고 이 이벤트는 비동기로 처리
- 처리가 완료되면 onRequest() 함수 호출



__filename, __dirname

- 노드에서 파일사이에 모듈관계가 있는 경우가 많아
  현재 파일의 경로나 파일명을 알아야 하는 경우 많음
- 이때 다음 키워드로 경로에 대한 정보를 제공
- __filename : 현재 파일경로
- __dirname : 현재 폴더경로

process

- 현재 실행중인 노드 프로세스에 대한 정보를 담고 있음
  - 컴퓨터마다 출력값이 PPT와 다를 수 있음

process.env

- 시스템 환경 변수들이 들어있는 객체
  - 비밀키(데이터베이스 비밀번호, 서드파티 앱 키 등)를 보관하는 용도로도 쓰임
  - 서비스가 해킹당해 코드가 유출되었을 때 비밀번호가 코드에 남아 추가 피해 발생 가능, 중요 비밀번호는 다음과 같이 process.env의 속성으로 대체

```
const secretId = process.env.SECRET_ID;
const secretCode = process.env.SECRET_CODE;
```

process.nextTick(콜백)

- 이벤트 루프가 다른 콜백 함수들보다 nextTick의 콜백함수를 우선적으로 처리함
  - 너무 남용하면 다른 콜백 함수들 실행이 늦어짐
  - 비슷한 경우로 promise가 있음

```
setImmediate(() => {
    console.log('immediate');
});
process.nextTick(() => {            // 제일 먼저 처리
    console.log('nextTick');
});
setTimeout(() => {
    console.log('timeout');
}, 0);
Promise.resolve().then(() => console.log('promise'));
```

process.exit(코드)

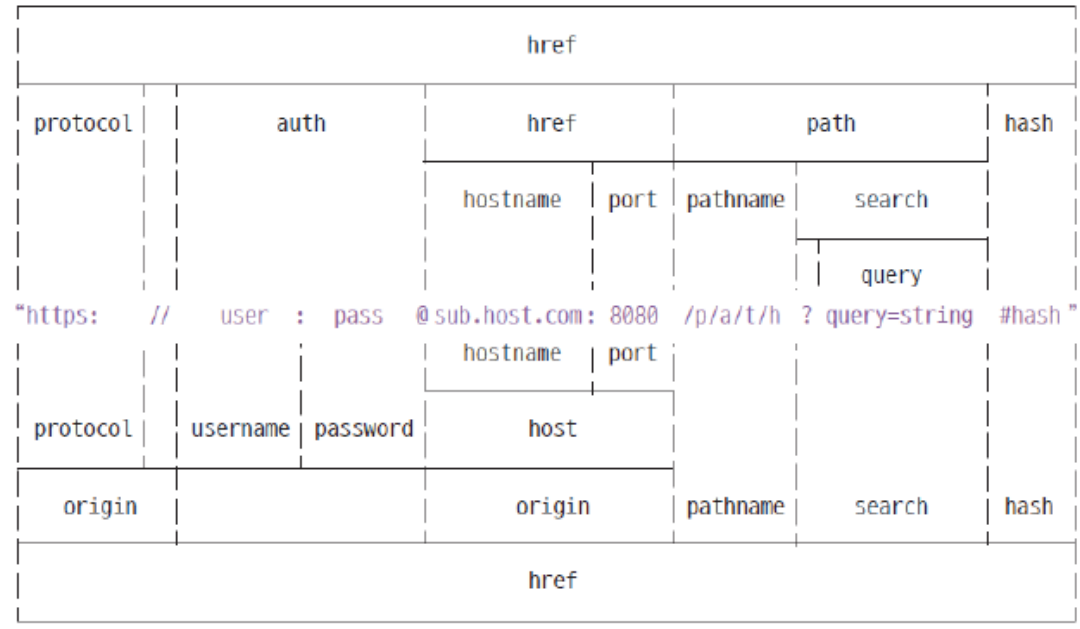
- 현재의 프로세스를 멈춤
  - 인자로 코드 번호를 줄 수 있음
  - 인자로 코드가 없거나 0이면 정상 종료
  - 이외의 코드는 비정상 종료를 의미함
- 서버에서 사용하면 서버가 멈추므로 서버에는 거의 사용하지 않음
- 서버외의 독립적인 프로그램에서 수동으로 노드를 멈출 때 사용

노드 내장 모듈 알아보기

url 모듈

- 인터넷 주소를 쉽게 조작하도록 도와주는 모듈
  - url 처리에 크게 두 가지 방식이 있음(기존 노드 방식, WHATWG 방식)
  - 아래 그림에서 궁ㄴ데 주소를 기준으로 위쪽은 기존 노드 방식, 아래쪽은 WHATWG 방식

그림 3-7 WHATWG와 노드의 주소 체계



url 모듈 메서드

- 기존 노드 방식 메서드
  - url.parse(주소) : 주소를 분해함
    - WHATWG와 차이 : username, password 대신 auth 속성, searchParams 대신 query 속성

- url.format(객체) :
  WHATWG 방식의 url과 기존 노드의 url 모두 사용 가능, 분해된 url 객체를 원래 상태로 조립

한글 깨지는 문제 해결

- response.writeHead(statusCode, obejct) 사용
  - response 객체의 메소드에서 헤더 정보를 응답에 작성해서 내보내는 것
  - 첫 번째 인자는 상태코드, 두번째 인자에는 헤더 정보를 연관배열로 정리

```
response.writeHead(200,{'Content-Type':'text/plain'})
// 200 정상코드, Content-Type 헤더에 text/plain 값 설정
// 이 콘텐츠는 표준 텍스트라는 정보
response.writeHead(200,{'Content-Type':'text/plain; charset=utf-8'})
// UTF-8 방식으로 인코딩 문제 해결
```

단방향 암호화(crypto)

- 암호화는 가능하지만 복호화는 불가능
  - 암호화 : 평문 -> 암호
  - 복호화 : 암호 -> 평문
- 단방향 암호화의 대표 주자는 해시 기법
  - 문자열을 고정된 길이의 다른 문자열러 바꿈


```
const crypto = require('crypto');

console.log('base64: ', crypto.createHash('sha512').update('비밀번
호').digest('base64'));
// createHash(사용할 해시 알고리즘) : md5, sha1은 취약점 발견, sha512면 충분
// update(변환할 문자열) : 변환할 문자열 넣어줌
// digest(인코딩) : 인코딩할 알고리즘 넣어줌 base64, hex, latin1 중 base64 애용
//                  결과문자열이 가장 짧음
```

pbkdf2

- 컴퓨터의 발달로 기존 암호화 알고리즘이 위협받는 중
  - sha512가 취약해지면 sha3으로 넘어가야함
  - 현재는 pbkdf2나 bcrypt, scrpyt 알고리즘으로 비밀번호 암호화
  - Node는 pbkdf2와 scrypt 지원

그림 3-9 pbkdf2

- 예제

```
const crypto = require('crypto');

crypto.randomBytes(64, (err, buf) => {
    const salt = buf.toString('base64');
    console.log('salt:', salt);
    crpyto.pbkdf2('비밀번호', salt, 100000, 64, 'sha512', (err, key) => {
        console.log('password:', key.toString('base64'));
    });
});
```

양방향 암호화 메서드

- 대칭형 암호화 (암호문 복호화 가능)
  - Key 사용
  - 암호화/복호화시 같은 Key 사용
- crypto.createCipher(알고리즘, 키)
  - 암호화 알고리즘과 키를 넣어줌
- cipher.update(문자열, 인코딩, 출력 인코딩)
  - 암호화할 대상과 대상의 인코딩, 출력 결과물의 인코딩 넣어줌
  - 보통 문자열은 utf-8, 암호는 base64 사용
- cipher.final(출력 인코딩)
  - 출력 결과물의 인코딩을 넣어주면 암호화
- crypto.createDecipher(알고리즘, 키)
  - 복호화할 때 사용, 암호화할 때 사용했던 알고리즘과 키 넣어줘야 함
- decipher.update(문자열, 인코딩, 출력 인코딩)
  - 암호화된 문장, 그 문장의 인코딩, 복호화할 인코딩 넣어줌
  - base64, utf-8 순으로 넣어줌(cipher와 역순)
- decipher.final(출력 인코딩)
  - 복호화 결과물의 인코딩

```
const crpyto = require('crypto');

const cipher = crypto.createCipher('aes-256-cbc', '열쇠');
let result = cipher.update('암호화할 문장', 'utf8', 'base64');
result += cipher.final('base64');
console.log('암호화:',result);

const decipher = crypto.createDecipher('aes-256-cbc', '열쇠');
let result2 = decipher.update(result, 'base64', 'utf8');
reesult2 += decipher.final('utf8');
console.log('복호화', result2);
```

util

- 각종 편의 기능을 모아둔 모듈
    - deprecated, promisify 자주 쓰임