Node -> node.js교과서 p132~p169

DNS(Domain Name System) : 사람이 읽을수있는 도메인 이름

인터넷상의 모든 컴퓨터는 숫자를 사용하여 서로를 찾고 통신하는데 이러한숫자를 IP주소라고하며 [www.naver.com](http://www.naver.com)과 같은 도메인 이름을 입력하여도 접근가능하다

-Node에서 ip나 dns정보 얻는법

해당 도메인의 IP주소는 dns.lookup(도메인)으로 얻어낼수있으며

해당레코드 A(ipv4주소), AAAA(ipv6주소), NS(네임서버), SOA(도메인정보), CNAME(별칭,www가붙은주소는 별칭인 경우가많다), MX(메일서버)로 해당 레코드에대한 도메인의 정보를얻어올수있다

dns.resovle(도메인,별칭레코드)

파일경로 : node/ch3/dns.mjs

import dns from 'dns/promises';

const ip = await dns.lookup('gilbut.co.kr');

console.log('iP:'+ip);

const a = await dns.resolve('gilbut.co.kr','A');

console.log('A',a);

const mx = await dns.resolve('gilbut.co.kr','MX');

console.log('MX',mx);

const cname = await dns.resolve('www.gilbut.co.kr','CNAME');

console.log('CNAME',cname);

const any= await dns.resolve('gilbut.co.kr','ANY');

console.log('ANY',any);

-Node의 crypto를 이용하여 암호화

단방향 암호화: 복화화할수 없는 암호화 방식을 뜻하며 해시함수라고부르기도한다 비밀번호같은 경우 단방향 암호화를 보통사용한다

1. 입력받은 비밀번호를 같은 암호화 알고리즘으로 암호화한후

2. 데이터베이스에저장된 비밀번호를 앞의 알고리즘을 이용하여 암호화된문자열을 비교한다

createHash(사용할알고리즘): md5,sha1이미 취약점이 발견 sha256, sha512등이 가능 sha512가 현재 충분 취약해지면 더 강화된 알고리즘으로 바꿔야한다

update(문자열) : 변환시킬 문자열

digest(인코딩) : 인코딩할 알고리즘을 선택 base64, hex, latin1이 주로사용 그중 base64의결과 문자열이 짧아서 애용

현재는 주로 pbkdf2나 bcrypt, scrypt 라는 알고리즘으로 주로 암호화하는데 이중 Node에서 지원하는 pbkdf2는 기존의 문자열에 salt라는 문자열을 붙인후 해시알고리즘을 반복해서 적용하는것으로

1.64바이트의 salt문자열을 만들어주는데 고정시킬수도있으며 예제에서는 randomBytes()를 이용해 생성하였다

2. randomBytes()를이용해 생성하였다면 콜백함수를 이용해 그결과 (err,buf)를통해 생성된 buf를 buf.toString(인코딩형식)을 변수 const salt에 넣어준다

3.crypto.pbkdf2()알고리즘의 매개변수는 순서대로

-‘변환시킬문자열’

-2에 생성한 salt

-알고리즘 반복횟수

-출력바이트

-‘해시알고리즘’

-(발생할수있는애러,변환된결과값=>버퍼형식으로 변환됨)

파일경로:node/ch3/hash.js

const crypto = require('crypto');

const hasycrypto1 = crypto.createHash('sha512').update('비밀번호').digest('base64');

console.log('base64:',hasycrypto1);

const hasycrypto2 = crypto.createHash('sha512').update('비밀번호').digest('base64');

console.log('base64:',hasycrypto2);

const hexcrypto = crypto.createHash('sha512').update('비밀번호').digest('hex');

console.log('hex:',hexcrypto);

console.time('duration\_time');

crypto.randomBytes(64, (err,buf)=>{

    const salt = buf.toString('base64');

    console.log('salt :',salt);

    crypto.pbkdf2('비밀번호',salt, 1100000, 64, 'sha512',(err, key)=>{ //알고리즘동작하는 시간을 근거로 횟수를 다음과 같이 설정하였다

            console.log('password: ', key.toString('base64'));//반환되는 key값이 버퍼형식임을알수있고 따라서 출력을위해 toString()를 사용하였다

            console.timeEnd('duration\_time');// 해당 알고리즘이 동작하는 시간을측정을하는데 1초정도걸리게 반복횟수를 정해주면좋다

    })

})

양방향 암호화 :암호화된 문자열을 복호화할수있다

파일경로 : node/ch3/cipher.js

const crypto = require('crypto');

const  key = 'abcdefghijklmnopqrstuvwxyz123456';//aes-256-cbc알고리즘의경우 key는 32바이트

const iv = '1234567890123456';//aes-256-cbc알고리즘의경우 iv는 16바이트로 초기화벡터

const cipher = crypto.createCipheriv('aes-256-cbc',key,iv);//변수cipher에 암호화 알고리즘과,key, iv인 객체를 넣어주고

let result = cipher.update('암호화할 문장','utf-8','base64');//암호화할 대상과 대상의 인코딩방식, 출력결과물의 인코딩방식을 넣어준다

result += cipher.final('base64');//출력결과물의 인코딩을 넣어 암호화

console.log('암호화 결과 : ',result);

const decipher = crypto.createDecipheriv('aes-256-cbc',key,iv);

let result2 = decipher.update(result,'base64','utf-8');

result2 += decipher.final('utf-8');

console.log('복호화 결과 : ',result2);

-worker\_threads모듈

노드에서 멀티 스레드 방식으로 작업하는 방법으로 CPU의 퍼포먼스를 위한 솔루션

|  |  |
| --- | --- |
| 기존 브라우저 특징 | worker\_threads특징 |
| 하나의 프로세스  하나의 스레드  하나의 이벤트루프  하나의 js 엔진 인스턴스  하나의 node.js인스턴스 | 하나의 프로세스  여러개의 스레드  스레드별 하나의 이벤트 루프  스레드 별 하나의 js 엔진 인스턴스  스레드 별 하나의 node.js 인스턴스 |