네트워크서비스 프로토콜

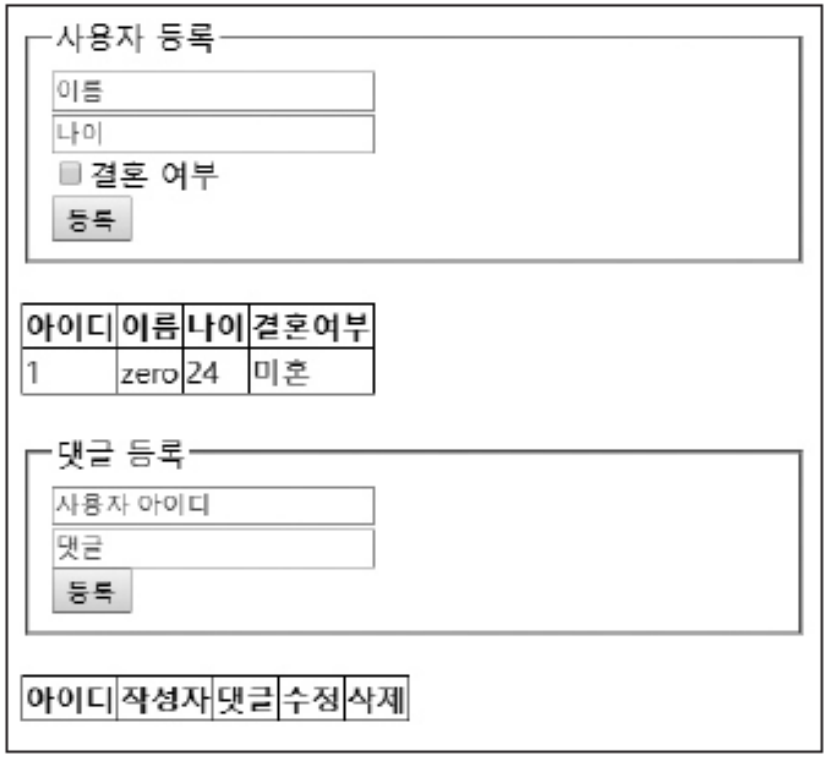
5주차 1

» 소프트웨어학부
» 김형균 교수

수업에 들어가며

- 전반기 학습목표
  - 노드 개념 이해하기
    - 노드 내장 객체 알아보기
    - 노드 내장 모듈 사용하기
    - 파일 시스템 접근하기
    - 이벤트 이해하기
    - 예외 처리하기
  - 노드서버 활용 예제 실습
    - 익스프레스 웹서버 설치
    - MySQL 설치 및 DB 생성
    - 시퀄라이즈 사용

그림 7-59 접속 화면



수업에 들어가며

- 오늘의 학습목표
  - 지난시간 http모듈 웹서버 개념
  - 콜백함수
  - 노드 개념 이해하기
    - 노드 내장 객체 알아보기
    - 노드 내장 모듈 사용하기

복습 : 노드로 http 서버 만들기

server1.js

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

http.createServer((req, res) => {
  res.write('<h1>Hello Node!</h1>');
  res.end('<p>Hello Server!</p>');
}).listen(8080, () => {
  console.log('8080번 포트에서 서버 대기 중입니다!');
});
```

노드 내장 객체 알아보기

1. global p.82

Node.js 교과서

노드의 전역 객체

- 브라우저의 window같은 역할
- 모든 파일에서 접근 가능
- window처럼 생략도 가능
  (console, require도 global의 속성)

- global.console
- global.require

- global 객체 내부 확인(콘솔)

콘솔

```
$ node
> global
```

명령 프롬프트 - node

```
Microsoft Windows [Version 10.0.18362.356]
(c) 2019 Microsoft Corporation. All rights reserved.

C:₩Users₩kim>node
> global
Object [global] {
  DTRACE_NET_SERVER_CONNECTION: [Function],
  DTRACE_NET_STREAM_END: [Function],
  DTRACE_HTTP_SERVER_REQUEST: [Function],
  DTRACE_HTTP_SERVER_RESPONSE: [Function],
  DTRACE_HTTP_CLIENT_REQUEST: [Function],
  DTRACE_HTTP_CLIENT_RESPONSE: [Function],
  COUNTER_NET_SERVER_CONNECTION: [Function],
  COUNTER_NET_SERVER_CONNECTION_CLOSE: [Function],
  COUNTER_HTTP_SERVER_REQUEST: [Function],
  COUNTER_HTTP_SERVER_RESPONSE: [Function],
  COUNTER_HTTP_CLIENT_REQUEST: [Function],
  COUNTER_HTTP_CLIENT_RESPONSE: [Function],
  global: [Circular],
  process:
   process {
     title: '명령 프롬프트 - node',
     version: 'v10.16.3',
     versions:
      { http_parser: '2.8.0',
        node: '10.16.3',
        v8: '6.8.275.32-node.54',
```

2. global 속성 공유

» 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());
```

콘솔

```
$ node globalB
안녕하세요
```

3. console 객체

» 브라우저의 console 객체와 매우 유사

- 노드에서는 window대신 global 객체안에 들어있음
- Console객체는 보통 디버깅을 위해 사용됨
  - 개발 중 변수에 값이 제대로 들어 있는지 확인
  - 에러 발생시 에러 내용을 콘솔에 표시할때
  - 코드 실행시간을 알아볼때

3. console 객체

- console.time, console.timeEnd: 시간 로깅
- console.error: 에러 로깅
- console.log: 평범한 로그
- console.dir: 객체 로깅
- console.trace: 호출스택 로깅

```
> Array(2) ["C:\Program Files\nodejs\node.exe", "c:\test\nodejs-book-master\ch3\3.4\console.js"]
  평범한 로그입니다 쉼표로 구분해 여러 값을 찍을 수 있습니다
  abc 1 true
  에러 메시지는 console.error에 담아주세요
> Object {outside: Object}
> Object {colors: false, depth: 2}
  시간측정: 2.89892578125ms
  시간측정: 3.316ms
> Object {outside: Object}
> Object {colors: true, depth: 1}
  console.trace()
      at b (file:///c:/test/nodejs-book-master/ch3/3.4/console.js:25:10)
      at a (file:///c:/test/nodejs-book-master/ch3/3.4/console.js:28:2)
      at (anonymous) (file:///c:/test/nodejs-book-master/ch3/3.4/console.js:30:0)
      at Module._compile (internal/modules/cjs/loader.js:775:13)
      at Module._extensions..js (internal/modules/cjs/loader.js:789:9)
      at Module.load (internal/modules/cjs/loader.js:653:31)
      at tryModuleLoad (internal/modules/cjs/loader.js:593:11)
      at Module._load (internal/modules/cjs/loader.js:585:2)
      at Module.runMain (internal/modules/cjs/loader.js:831:11)
      at startup (internal/bootstrap/node.js:283:18)
      at bootstrapNodeJSCore (internal/bootstrap/node.js:622:2)
  Trace: 에러 위치 추적
      at b (c:\test\nodejs-book-master\ch3\3.4\console.js:25:11)
      at a (c:\test\nodejs-book-master\ch3\3.4\console.js:28:3)
      at Object.<anonymous> (c:\test\nodejs-book-master\ch3\3.4\console.js:30:1)
      at Module._compile (internal/modules/cjs/loader.js:775:14)
      at Object.Module._extensions..js (internal/modules/cjs/loader.js:789:10)
      at Module.load (internal/modules/cjs/loader.js:653:32)
      at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
      at Function.Module._load (internal/modules/cjs/loader.js:585:3)
      at Function.Module.runMain (internal/modules/cjs/loader.js:831:12)
      at startup (internal/bootstrap/node.js:283:19)
  전체시간: 8.31396484375ms
```

```
const string = 'abc';
const number = 1;
const boolean = true;
const obj = {
  outside: {
    inside: {
      key: 'value',
    },
  },
};
console.log(process.argv);
console.time('전체시간');
console.log('평범한 로그입니다 쉼표로 구분해 여러 값을 찍을 수 있습니다');
console.log(string, number, boolean);
console.error('에러 메시지는 console.error에 담아주세요');

console.dir(obj, { colors: false, depth: 2 });
console.dir(obj, { colors: true, depth: 1 });

console.time('시간측정');
for (let i = 0; i < 100000; i++) {}
console.timeEnd('시간측정');

function b() {
  console.trace('에러 위치 추적');
}
function a() {
  b();
}
a();

console.timeEnd('전체시간');
```

5. 타이머 메서드

- 타이머 기능을 제공하는 함수로 global 객체에 포함됨
- 타이머 함수들은 모두 아이디를 반환하며 아이디를 사용해 타이머를 취소할 수 있음
- set 메서드에 clear 메서드가 대응됨
  - set 메서드의 리턴 값(아이디)을 clear 메서드에 넣어 취소

- set 메서드
  - setTimeout(콜백 함수, 밀리초): 주어진 밀리초(1000분의 1초) 이후에 콜백 함수를 실행합니다.
  - setInterval(콜백 함수, 밀리초): 주어진 밀리초마다 콜백 함수를 반복 실행합니다.
  - setImmediate(콜백 함수): 콜백 함수를 즉시 실행합니다.
- clear 메서드
  - clearTimeout(아이디): setTimeout을 취소합니다.
  - clearInterval(아이디): setInterval을 취소합니다.
  - clearImmediate(아이디): setImmediate를 취소합니다.

6. 타이머 예제

» 다음 예제의 콘솔 출력을 맞춰보자

- setTimeout(콜백, 0)보다 setImmediate 권장

그림 3-4 실행 순서

timer.js

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

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

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

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

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

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

clearImmediate(immediate2);
```

콜백(CallBack)함수란?

친구들과 즐겁게 시내를 돌아다니다가, 집에 갈 때 사갈 떡볶이를 사가려고 한다.
그런데 이게 무슨일이람. 떡볶이가 너무 많이 밀려서 시간이 조금 걸린다고 한다 !
그래서 나는 전화번호를 주고, 조리가 끝나면 받아갈테니 **전화를 다시 주라고 하였다!**

- "함수 속에서 또 다시 함수 콜을 한다."
- Node.js 에서 가장 핵심적인 부분
  - 무엇인가 일을 다른 객체에게 시키고,
  - 그 일이 끝나는 것을 기다리는 것이 아니라
  - 그 객체가 나를 다시 부를때까지 내 할일을 하고 있는 것
- Non-Block이며 비동기(Asynchronou)방식의 함수로 사용됨

콜백(CallBack)함수란?

- 우리가 흔히 생각하는 일반적인 함수란 입력(파라미터)이 있고 출력(리턴값)이 있습니다.
- 하지만 자바스크립트에서는 출력값이 없고 그 대신에 콜백 함수를 입력받는 함수들이 많이 있습니다.
- 콜백 함수는 다른 함수에 인자로 넘어가서 실행될 로직을 담게 됩니다.

```
function add(a,b){
    var result = a+b;
    return result;
}

const plus = add(5,10);
console.log(plus);
```

```
plus = function(a,b,callback){
    var result = a+b;
    callback(result);
}

plus(5,10, function(res){
    console.log(res);
})
```

콜백(CallBack)함수 실습문제

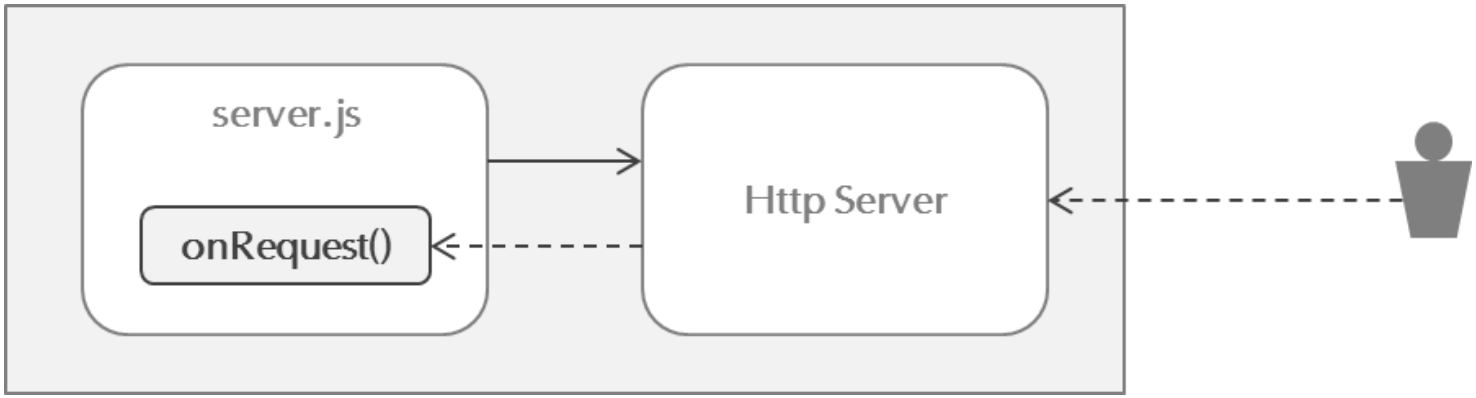
- 유저 ID를 인자로 받아 DB나 API 연동 없이 임의의 유저 객체를 리턴하는 findUser()라는 함수이다.
- 이를 콜백함수로 형태인 findUserAndCallBack() 정의해 보자.

```
function findUser(id) {
    const user = {
      id: id,
      name: "User" + id,
      email: id + "@test.com"
    };
    return user;
  }

  const user = findUser(1);
  console.log("user:", user);
```

Http서버 Callback 호출

- 비동기 Callback 호출 위해 onRequest() 함수를 이벤트 리스너로 등록
- 'request' 이벤트에 대한 Callback 역할
- 클라이언트가 Http 요청을 보내면 Http Server에 'request' 타입 이벤트가 발생하고 이 이벤트는 비동기로 처리
- 처리가 완료되면 onRequest() 함수가 호출



nodejs-book-master > ch4 > 4.1 > JS server1_onreq.js > ...

```
1   const http = require('http');
2
3
4
5
6
7
8
9
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
C:\Program Files\nodejs\node.exe --inspect-brk=40955 nodejs-book
Debugger listening on ws://127.0.0.1:40955/8afcc502-a64f-4a9d-95
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
server has started.
request received.
request received.
request received.
request received.
```

7. __filename, __dirname

- 노드에서 파일사이에 모듈관계가 있는 경우가 많아 현재 파일의 경로나 파일명을 알아야 하는 경우 많음
- 이때 다음 키워드로 경로에 대한 정보를 제공함
- __filename: 현재 파일 경로
- __dirname: 현재 폴더(디렉터리) 경로

filename.js

```
console.log(__filename);
console.log(__dirname);
```

콘솔

```
$ node filename.js
C:\Users\zerocho\filename.js
C:\Users\zerocho
```

9. process

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

콘솔

```
$ node
> process.version
v10.0.0 // 설치된 노드의 버전입니다.
> process.arch
x64 // 프로세서 아키텍처 정보입니다. arm, ia32 등의 값일 수도 있습니다.
> process.platform
win32 // 운영체제 플랫폼 정보입니다. linux나 darwin, freebsd 등의 값일 수도 있습니다.
> process.pid
14736 // 현재 프로세스의 아이디입니다. 프로세스를 여러 개 가질 때 구분할 수 있습니다.
> process.uptime()
199.36 // 프로세스가 시작된 후 흐른 시간입니다. 단위는 초입니다.
> process.execPath
C:\\Program Files\\nodejs\\node.exe // 노드의 경로입니다.
> process.cwd()
C:\\Users\\zerocho // 현재 프로세스가 실행되는 위치입니다.
> process.cpuUsage()
{ user: 390000, system: 203000 } // 현재 cpu 사용량입니다.
```

10. process.env

- 시스템 환경 변수들이 들어있는 객체
  - 비밀키(데이터베이스 비밀번호, 서드파티 앱 키 등)를 보관하는 용도로도 쓰임
  - 서비스가 해킹당해 코드가 유출되었을때 비밀번호가 코드에 남아 추가 피해가 발생할 수 있으므로 중요 비밀번호는 다음과 같이 process.env 의속성으로 대체합니다.

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

11. process.nextTick(콜백)

» 이벤트 루프가 다른 콜백 함수들보다 nextTick의 콜백 함수를 우선적으로 처리함

- 너무 남용하면 다른 콜백 함수들 실행이 늦어짐
- 비슷한 경우로 promise가 있음
- 아래 예제에서 setImmediate, setTimeout보다 promise와 nextTick이 먼저 실행됨

nextTick.js

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

콘솔

```
$ node nextTick
nextTick
promise
timeout
immediate
```

12. 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 멈출때 사용

exit.js

```
let i = 1;
setInterval(() => {
  if (i === 5) {
    console.log('종료!');
    process.exit();
  }
  console.log(i);
  i += 1;
}, 1000);
```

콘솔

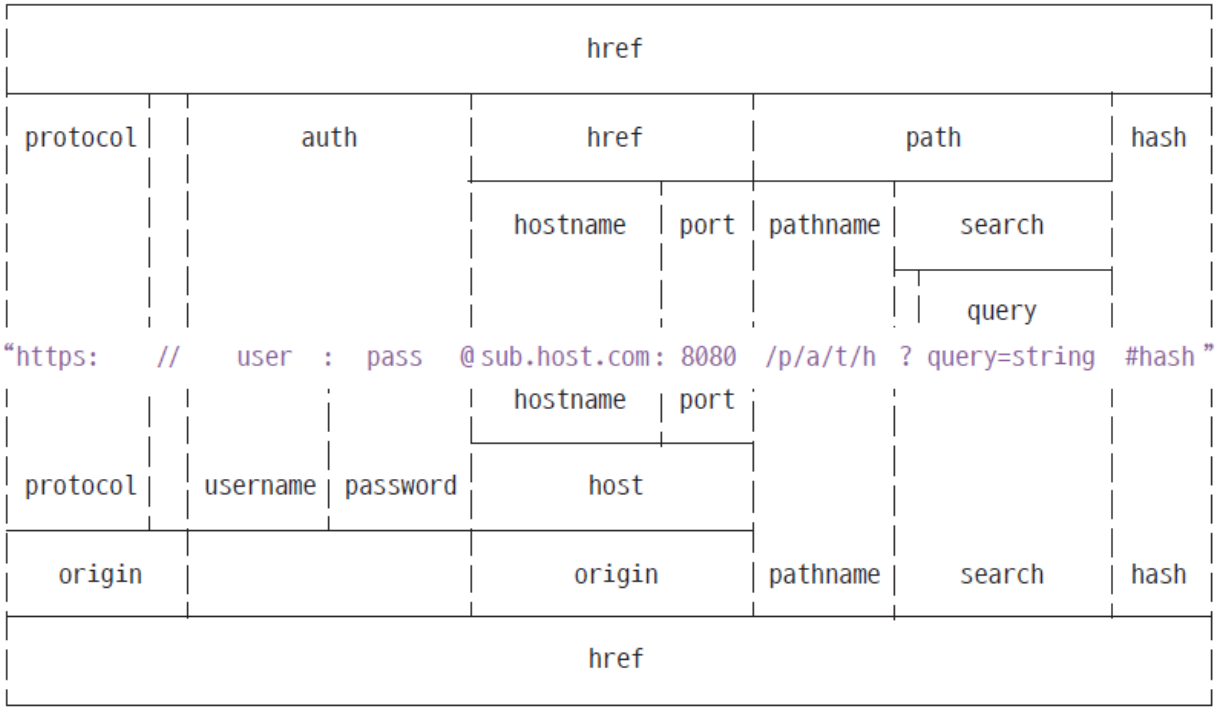
```
$ node exit
1
2
3
4
종료!
```

3.5 노드 내장 모듈 사용하기

6. url 모듈

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

그림 3-7 WHATWG와 노드의 주소 체계



9. url 모듈 메서드

- 기존 노드 방식 메서드
  - url.parse(주소): 주소를 분해합니다. WHATWG 방식과 비교하면 username과 password대신 auth 속성이 있고, searchParams 대신 query가 있습니다.
  - url.format(객체): WHATWG 방식의 url과 기존 노드의 url 모두 사용할 수 있습니다. 분해되었던 url 객체를 다시 원래 상태로 조립합니다.

7. url 모듈 예제


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

const { URL } = url;
const myURL = new URL('https://cs.kookmin.ac.kr/major/curriculum2019');
console.log('new URL():', myURL);
console.log('url.format():', url.format(myURL));
console.log('------------------------------');
const parsedUrl = url.parse('https://cs.kookmin.ac.kr/major/curriculum2019');
console.log('url.parse():', parsedUrl);
console.log('url.format():', url.format(parsedUrl));
```

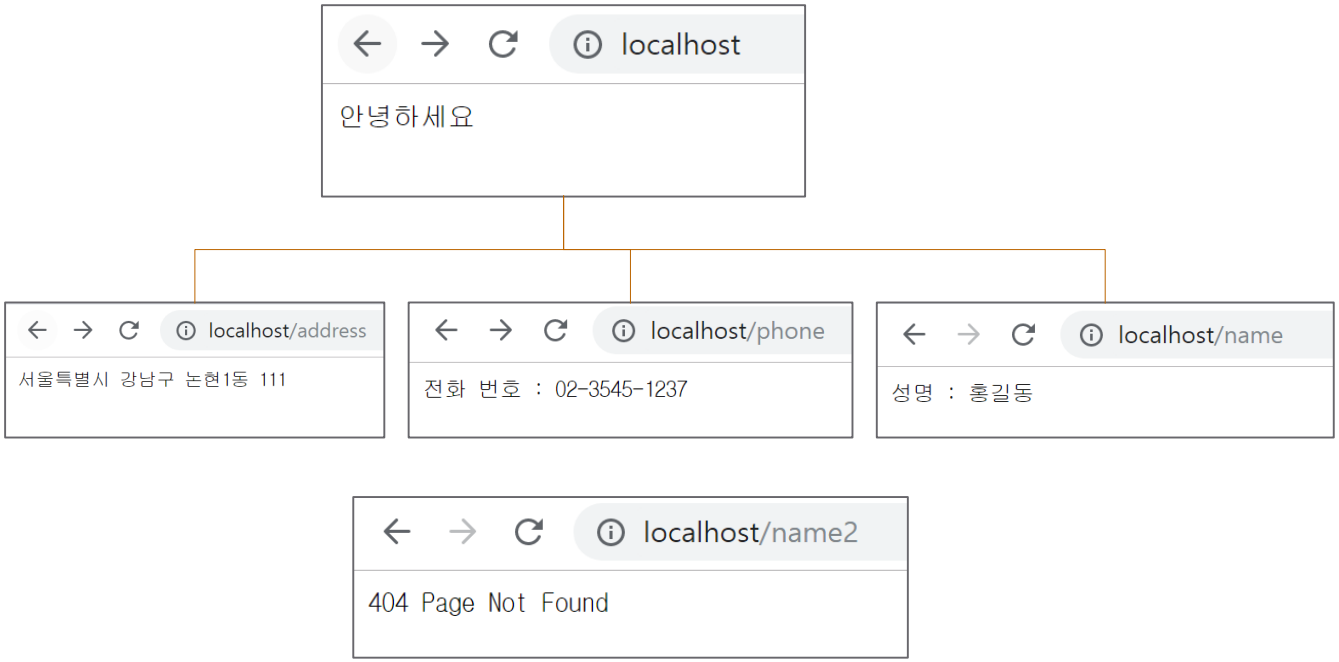
```
Debugger attached.
new URL():
> URL {Symbol(context): URLContext, Symbol(query): URLSearchParams}
url.format(): https://cs.kookmin.ac.kr/major/curriculum2019
------------------------------
url.parse():
> Url {protocol: "https:", slashes: true, auth: null, host: "cs.kookmin.ac.kr", port: null, …}
url.format(): https://cs.kookmin.ac.kr/major/curriculum2019
```

한글 깨지는(인코딩) 문제 해결하기

- response.writeHead(statusCode, object) 사용
  - response 객체의 메소드에서 헤더 정보를 응답에 작성해서 내보내는 것이다.
  - 첫번째 인자는 상태 코드를 지정하고 두번째인수에 헤더 정보를 연관 배열로 정리한 것이다.
- response.writeHead(200,{'Content-Type': 'text/plain'})
  - 200 정상코드 전송
  - Content-type이라는 헤더 정보에 'text/plain'의 값을 설정
  - "이 콘텐츠는 표준 텍스트이다"라는 것이 클라이언트에 전달된다.
- response.writeHead(200, {'Content-Type':'text/plain; charset=utf-8'});

실습문제

- http 서버를 이용해 아래와 같이 브라우저에 접속한 상태에서 요청한 자원이 소스코드에서 정의한 /, /address , /phone , /name 에 메시지를 브라우저에 출력해 보자.(응답포트:80)
- 주소를 요청시 콘솔에 요청된 url과 해당 path를 각각 출력한다.




```
Debugger attached.
Server is running...
/
resource path=/
/address
resource path=/address
/phone
resource path=/phone
/name
resource path=/name
```

13. 단방향 암호화(crypto)

» 암호화는 가능하지만 복호화는 불가능
- 암호화: 평문을 암호로 만듦
- 복호화: 암호를 평문으로 해독

» 단방향 암호화의 대표 주자는 해시 기법
- 문자열을 고정된 길이의 다른 문자열로 바꾸는 방식
- abcdefgh 문자열 -> qvew

그림 3-8 해시 함수

crypto.createHash('sha512').update('비밀번호').digest('base64')

- createHash(알고리즘): 사용할 해시 알고리즘을 넣어줍니다. md5, sha1, sha256, sha512 등이 가능하지만, md5와 sha1은 이미 취약점이 발견되었습니다. 현재는 sha512 정도로 충분하지만, 나중에 sha512마저도 취약해지면 더 강화된 알고리즘으로 바꿔야 합니다.
- update(문자열): 변환할 문자열을 넣어줍니다.
- digest(인코딩): 인코딩할 알고리즘을 넣어줍니다. base64, hex, latin1이 주로 사용되는데, 그중 base64가 결과 문자열이 가장 짧아 애용됩니다. 결과물로 변환된 문자열을 반환합니다.

hash.js

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

console.log('base64:', crypto.createHash('sha512').update('비밀번호').digest('base64'));
console.log('hex:', crypto.createHash('sha512').update('비밀번호').digest('hex'));
console.log('base64:', crypto.createHash('sha512').update('다른 비밀번호').
digest('base64'));
```

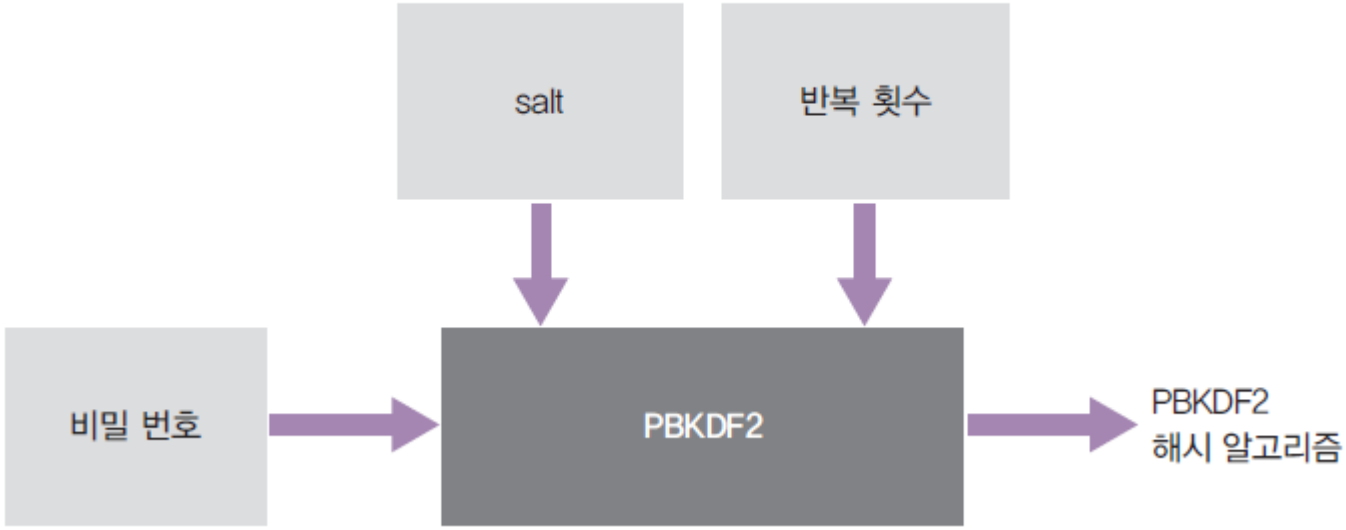
콘솔

```
$ node hash
base64: dvfV6nyLRRt3NxKSlTHOkkEGgqW2HRtfu19Ou/psUXvwlebbXCboxIPmDYOFRIpqav2eUTBFuHaZri5x
+usy1g==
hex: 76f7d5ea7c8b451b773712929531ce92410682a5b61d1b5fbb5f4ebbfa6c517bf095e6db5c26e8c483e
60d8385448a6a6afd9e513045b87699ae2e71faeb32d6
base64: cx49cjC8ctKtMzwJGBY853itZeb6qxzXGvuUJkbWTGn5VXAFbAwXGEOxU2Qksoj+aM2GWPhc107mmkyo
hXMsQw==
```

14. pbkdf2

- 컴퓨터의 발달로 기존 암호화 알고리즘이 위협받고 있음
  - sha512가 취약해지면 sha3으로 넘어가야함
  - 현재는 pbkdf2나, bcrypt, scrypt 알고리즘으로 비밀번호를 암호화
  - Node는 pbkdf2와 scrypt 지원

그림 3-9 pbkdf2



15. pbkdf2 예제

» 컴퓨터의 발달로 기존 암호화 알고리즘이 위협받고 있음

- crypto.randomBytes로 64바이트 문자열 생성 -> salt 역할
- pbkdf2 인수로 순서대로 비밀번호, salt, 반복 횟수, 출력 바이트, 알고리즘
- 반복 횟수를 조정해 암호화하는 데 1초 정도 걸리게 맞추는 것이 권장됨

pbkdf2.js

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

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

콘솔

```
$ node pbkdf2
salt: OnesIj8wznyKgHva1fmulYAgjf/OGLmJnwfy8pIABchHZF/Wn2AM2Cn/9170Y1AdehmJ0E5CzLZULps+da
F6rA==
password: b4/FpSrZulVY28trzNXsl4vVfhOKBPxyVAvwnUCWvF1nnXS1zsU1Paq2p68VwUfhB0LDD44hJOf+tL
e3HMLVmQ==
```

16. 양방향 암호화 메서드

- 대칭형 암호화(암호문 복호화 가능)
  - Key가 사용됨
  - 암호화할 때와 복호화 할 때 같은 Key를 사용해야 함
- crypto.createCipher(알고리즘, 키)
  - 암호화 알고리즘과 키를 넣어줍니다. 암호화 알고리즘은 aes-256-cbc를 사용했습니다.
- cipher.update(문자열, 인코딩, 출력 인코딩)
  - 암호화할 대상과 대상의 인코딩, 출력 결과물의 인코딩을 넣어줍니다. 보통 문자열은 utf8 인코딩을, 암호는 base64를 많이 사용합니다.
- cipher.final(출력 인코딩) : 출력 결과물의 인코딩을 넣어주면 암호화가 완료
- crypto.createDecipher(알고리즘, 키)
  - 복호화할 때 사용합니다. 암호화할 때 사용했던 알고리즘과 키를 그대로 넣어주어야 합니다.
- decipher.update(문자열, 인코딩, 출력 인코딩)
  - 암호화된 문장, 그 문장의 인코딩, 복호화할 인코딩을 넣어줍니다. createCipher의 update()에서 utf8, base64 순으로 넣었다면 createDecipher의 update()에서는 base64, utf8 순으로 넣으면 됩니다.
- decipher.final(출력 인코딩) : 복호화 결과물의 인코딩을 넣어줍니다.

17. 양방향 암호화

cipher.js

```
const crypto = 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');
result2 += decipher.final('utf8');
console.log('복호화:', result2);
```

콘솔

```
$ node cipher
암호화: JwGZuUveVRTttFegg8CaU3Tyqf8A48G6aHcaH6PnVpE=
복호화: 암호화할 문장
```

18. util

각종 편의 기능을 모아둔 모듈

- deprecated와 promisify가 자주 쓰임

util.js

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

const dontUseMe = util.deprecate((x, y) => {
  console.log(x + y);
}, 'dontUseMe 함수는 deprecated되었으니 더 이상 사용하지 마세요!');
dontUseMe(1, 2);

const randomBytesPromise = util.promisify(crypto.randomBytes);
randomBytesPromise(64)
  .then((buf) => {
    console.log(buf.toString('base64'));
  })
  .catch((error) => {
    console.error(error);
  });
```

콘솔

```
$ node util
3
(node:7264) DeprecationWarning: dontUseMe 함수는 deprecated되었으니 더 이상 사용하지 마세요!
60b4RQbrx1j130×4r95fpZac9lmcHyitqwAm8gKsHQKF8tcNhvcTfW031XaQqHlRKzaVkcENmIV25fDVs3SB
7g==
```