**Chapter 04. Node의 기본 기능 알아보기**

* 주소 문자열과 요청 파라미터
* 서버에서는 클라이언트가 요청한 사이트 주소 정보(URL객체)를 처리하면서 ‘사이트 주소’와 ‘요청 파라미터’를 구분하여 처리해야 한다.
* node에서는 ‘url모듈’을 사용하여 문자열 🡨🡪 URL 객체로의 변환을 쉽게 진행한다.
* 주소 문자열 : <https://www.google.co.kr/?gws_rd=ssl#newwindow=1&q=actor>’
* Protocol : https  
  host : [www.google.co.kr](http://www.google.co.kr)  
  query : [gws\_rd=ssl#newwindow=1&q=actor](https://www.google.co.kr/?gws_rd=ssl#newwindow=1&q=actor) (요청 파라미터 구분을 위해 앞에 ? 붙임)
* 주소 문자열 URL 객체 변환

NodeExample1>ch04\_test1.js

var url = require('url');

// 주소 string을 URL 객체로 변환

var curURL = url.parse('https://m.search.naver.com/search.naver?query=steve+jobs&where=m&sm=mtp\_hty');

// URL 객체를 주소 string으로 변환

var curStr = url.format(curURL);

console.log('주소 문자열 : %s', curStr);

// URL 객체를 호출

console.dir(curURL);

* 실행 결과



즉, URL 객체로 주소string 변환 시, 객체 내에 각각의 속성이 분류되어 표시된다.

* 상기 결과에서 query 속성에서 요청 파라미터 정보를 확인할 수 있는데, 웹 서버에서는 주로 query 속성 내의 요청 파라미터를 받아 처리하므로, query 속성에 있는 string을 다시 각 **요청 파라미터로 분리해야 한다. 각각의 요청 파라미터는 ‘&’ 로 구분한다.**

var url = require('url');

// 주소 string을 URL 객체로 변환

var curURL = url.parse('https://m.search.naver.com/search.naver?query=steve+jobs&where=m&sm=mtp\_hty');

// URL 객체를 주소 string으로 변환

var curStr = url.format(curURL);

console.log('주소 문자열 : %s', curStr);

// URL 객체를 호출

console.dir(curURL);

// 요청 파라미터 구분

var querystring = require('querystring');

var param = querystring.parse(curURL.query);

console.log('요청 파라미터 중 query의 값 : %s', param.query);

// steve jobs

console.log('원본 요청 파라미터 : %s', querystring.stringify(param));

// query=steve%20jobs&where=m&sm=mtp\_hty

querystring.parse( )를 사용해 URL객체 curURL의 query를 요청 파라미터 객체 param으로 파싱한다. 이 요청 파라미터 객체 param.query를 출력하면 검색어에 입력한 ‘steve jobs’만이 출력되고, param 객체를 다시 stringify 하면 원래 query 텍스트가 다시 출력된다.

* **이벤트 이해하기**
* Node 객체는 ‘EventEmiter’를 상속 받은 후 on( ), emit( ) 등을 사용해 이벤트를 관리할 수 있다.
* on(event, listener) : on( ) 메소드는 이벤트가 전달될 객체에 **이벤트 리스너를 설정**한다.

process.on('exit', function(){

console.log('exit 이벤트 발생');

});

setTimeout(function(){

console.log('2초 후에 시스템 종료 시도함.');

process.exit();

}, 2000);

* process 객체(EventEmitter를 이미 상속받음)에 on( )을 통해 ‘exit’이벤트 발생에 대한 리스너를 설정해 준다. ‘2초 후에…’출력 후 ‘exit 이벤트 발생’이 출력된다.
* 이벤트 직접 생성해 처리해보기

[ch04\_test2.js]

process.on('tick', function(count){

console.log('tick 이벤트 발생 : %s', count);

});

setTimeout(function(){

console.log('2초 후에 tick 이벤트를 전달 시도');

process.emit('tick', '2');

}, 2000);

* ‘tick’이라는 이벤트를 process 객체에 on( )을 통해 리스너 설정을 해준다. 이후 setTimeout에서 **emit( )**을 통해 ‘tick’이라는 이벤트와 ‘2’라는 string을 process 객체에 전달하고, ‘2’는 count 파라미터로 전달된다.
* 계산기 객체 모듈로 만들기

[calc3.js]

var util = require('util');

var EventEmitter = require('events').EventEmitter;

var Calc = function(){

var self = this;

this.on('stop', function(){

console.log('Calc에 stop event 전달됨');

});

};

util.inherits(Calc, EventEmitter);

Calc.prototype.add = function(a,b){

return a + b;

};

module.exports = Calc;

module.exports.title = 'calculator';

* Calc는 function을 통해 Prototype Object로 만들고, 자신(this)에게 on( )을 통해 stop 이벤트에 대한 리스너를 설정한다. 그리고 prototype 객체 속성으로 add 함수를 추가한다.

[ch04\_test4.js]

var Calc = require('./calc3');

var calc = new Calc();

calc.emit('stop');

console.log(Calc.title + '에 stop 이벤트를 전달함');

* **파일 다루기**
* 파일 읽어들이기

동기식 or 비동기식 IO

var fs = require('fs');

// 파일을 동기식 IO로 읽는다

var data = fs.readFileSync('./package.json', 'utf8');

console.log(data);

* 동기식 I/O는 메소드 끝에 Sync가 붙으며, 파일을 전부 다 읽은 이후에 출력이 된다.

var fs = require('fs');

// 파일을 비동기식 IO로 읽는다

fs.readFile('./package.json', 'utf8', function(err, data){

// 읽어들인 데이터 출력

console.log(data);

});

console.log('파일 package.json을 읽도록 요청했습니다');

* 비동기식 I/O를 통해 파일을 읽도록 한 후, 바로 출력 메시지를 보내고 파일 읽기가 완료되면 콜백 함수를 실행시켜 파일 내용을 출력한다.  
  (이 때 에러가 발생하면 err 파라미터에 전달되는데, 이상이 없다면 err의 값은 null이 된다. 주로 err 파라미터의 null 체크를 통해 이상 유무를 확인한다)
* 파일 쓰기

파일의 내용을 일부분만 읽어들일 경우, fs.open( ) 메소드를 사용한다.   
 •open(path, flags, [mode], [callback])  
 🡪 flags 종류   
 1. ‘r’ : 읽기. 파일 없을 경우 예외처리  
 2. ‘w’ 쓰기. 파일이 없으면 만들어지고, 있으면 이전 내용을 모두 삭제  
 3. ‘w+’ 일기와 쓰기. 파일이 없으면 만들어지고, 있으면 이전 내용을 모두 삭제  
 4. ‘a+’ 읽기와 추가. 파일이 없으면 만들어지고, 있으면 이전 내용을 모두 삭제

[ch04\_test8.js]

var fs = require('fs');

// 파일에 데이터 쓰기

fs.open('./output.txt', 'w', function(err, fd){

if(err) throw err;

var buf = new Buffer('안녕!!\n');

fs.write(fd, buf, 0, buf.length, null, function(err, written, buffer){

if(err) throw err;

console.log(err, written, buffer);

fs.close(fd, function(){

console.log('파일을 열고 데이터를 쓴 후 파일 닫기 완료.');

});

});

});

* fs.open( )을 통해 파일을 생성해 열면, 우선 파일을 fd 파라미터로 전달된다.   
  open( )메소드의 콜백으로 텍스트 버퍼를 만들고, 동시에 fs.write( )를 실행해 해당 파일, 즉 fd에 원하는 텍스트 buf를 쓰고 fs.write( )의 콜백으로서 fs.close( )를 실행해 최종적으로 파일 쓰기를 마무리 한다.
* 파일 읽기

var fs = require('fs');

// 파일을 오픈해서 데이터를 읽는다

fs.open('./output.txt', 'r', function(err, fd){

if(err) throw err;

var buf = new Buffer(10); // byte 데이터의 크기

console.log('버퍼 타입 : %s', Buffer.isBuffer(buf)); // true

// output.txt를 오픈한 후, 10byte의 버퍼 객체를 생성했다.

// fs.read()를 통해 output.txt(fd)를 읽은 후 버퍼(buf)에 대입한다.

// bytesRead 파라미터는 읽어들인 버퍼의 byte크기를 의미함.

// buf 크기 설정 시 '안녕!!'보다 적게

// 설정 하면 정상적으로 읽히지 않는다.

fs.read(fd, buf, 0, buf.length, null, function(err, bytesRead, buffer){

if(err) throw err;

// buf는 buffer 파라미터로 전달되고, 이 전달된 buffer 파라미터를

// String 변환한다.

var inStr = buffer.toString('utf8', 0, bytesRead);

console.log('파일에서 읽은 데이터 : %s', inStr);

console.log(err, bytesRead, buffer);

fs.close(fd, function(){

console.log('output.txt 파일 열고 읽기 완료 후 닫음');

});

});

});

* 버퍼를 다루는 방법

빈 버퍼를 만들고 String을 대입하거나, 버퍼를 만들면서 문자열을 파라미터로 전달.  
String은 동일하게 들어가지만 전자의 경우 빈 공간이 남기도 한다.

버퍼 여부는 isBuffer( ), 두 개를 붙여 하나의 새로운 버퍼 객체 생성시에는 concat( ) 사용.

[ch04\_test10.js]

// 버퍼 객체를 크기만 지정한 후 문자열 대입

var output = '안녕!!1';

var buffer1 = new Buffer(10);

var len = buffer1.write(output, 'utf8');

console.log('첫 번째 버퍼의 문자열 : %s', buffer1.toString());

// 버퍼 객체를 String 대입하며 생성

var buffer2 = new Buffer('안녕!!2', 'utf8');

console.log('두 번째 버퍼의 문자열 : %s', buffer2.toString());

// 타입 확인

console.log('버퍼 객체의 타입 : %s', Buffer.isBuffer(buffer1));

// 버퍼 객체에 들어 있는 문자열 데이터를 문자열 변수로 변환

var byteLen = Buffer.byteLength(output);

var str1 = buffer1.toString('utf8', 0, byteLen);

var str2 = buffer2.toString('utf8');

// 첫 번째 버퍼 객체의 문자열을 두 번째 버퍼 객체로 복사

buffer1.copy(buffer2, 0, 0, len);

console.log('두 번째 버퍼에 복사한 후의 문자열 : %s', buffer2.toString('utf8'));

// 두 개의 버퍼를 붙이기

var buffer3 = Buffer.concat([buffer1, buffer2]);

console.log('두 개의 버퍼를 붙인 후의 문자열 : %s', buffer3.toString('utf8'));

* 스트림 단위로 파일 읽고 쓰기

※ (JAVA기준)파일 Stream은 [프로그램] 기준으로 입력을 받는 Input Stream과 출력을 내보내는 Output Stream이 존재한다. 스트림은 **‘단일 방향’ I/O** 시스템이므로, 반드시 프로그램은 Input Stream과 Output Stream을 가져야 한다. 이 챕터에서는 읽기/쓰기 스트림 사용

[ch04\_test11.js] – output.txt를 읽어 output2.txt 파일을 쓰는 케이스

var fs = require('fs');

// 읽기 스트림과 쓰기 스트림 생성

var infile = fs.createReadStream('./output.txt', {flags: 'r'});

var outfile = fs.createWriteStream('./output2.txt', {flags: 'w'});

// 읽기 스트림에 'data'이벤트 리스너를 설정해, 데이터가 들어오면 쓰기 스트림에 해당 데이터를 전달해 작성하게 만든다.

infile.on('data', function(data){

console.log('읽어들인 데이터', data);

outfile.write(data);

});

infile.on('end', function(){

console.log('파일 읽기 종료');

outfile.end(function(){

console.log('파일 쓰기 종료');

});

});

* pipe( ) 메소드는 두 개의 스트림을 연결한다. ReadStream 객체와 WriteStream 객체를 연결해 스트림 간 데이터를 알아서 전달한다.

var fs = require('fs');

var inname = './output.txt';

var outname = './output2.txt';

fs.exists(outname, function(exists){

if(exists){

fs.unlink(outname, function(err){

if(err) throw err;

console.log('기존파일 [' + outname + '] 삭제함');

});

}

var infile = fs.createReadStream(inname, {flags: 'r'});

var outfile = fs.createWriteStream(outname, {flags: 'w'});

infile.pipe(outfile);

console.log('파일 복사 [' + inname + '] -> [' + outname + ']');

});

* 기존 ‘data’이벤트로 infile에 outfile을 연결하던 부분을 infile.pipe(outfile)로 축약함.
* http 모듈로 요청받은 파일 내용 읽고 응답하기

var fs = require('fs');

var http = require('http');

var server = http.createServer(function(req, res){

// 파일을 읽어 응답 스트림과 pipe()로 연결

var instream = fs.createReadStream('./output.txt');

instream.pipe(res);

});

server.listen(7001, '127.0.0.1');

* fs모듈로 디렉터리 만들고 삭제하기

var fs = require('fs');

fs.mkdir('./docs', 0666, function(err){

if(err) throw err;

console.log('새로운 docs 폴더를 생성');

fs.rmdir('./docs', function(err){

if(err) throw err;

console.log('docs 폴더를 삭제했습니다.');

});

});

* **로그 파일 남기기**
* **에러 나는 부분 확인…**