

ch04. 노드의 기본 기능 알아보기

04-01 주소문자열과 요청 파라미터 다루기

- 일반 문자열을 URL 객체로 만들거나 URL 객체를 일반 문자열로 변환

주소 문자열

`https://www.google.co.kr/?gws_rd=ssl#newwindow=1&q=actor`

url 모듈

URL 객체



protocol : 'https'



host : 'www.google.co.kr'



query : 'gws_rd=ssl#newwindow=1&q=actor'



...

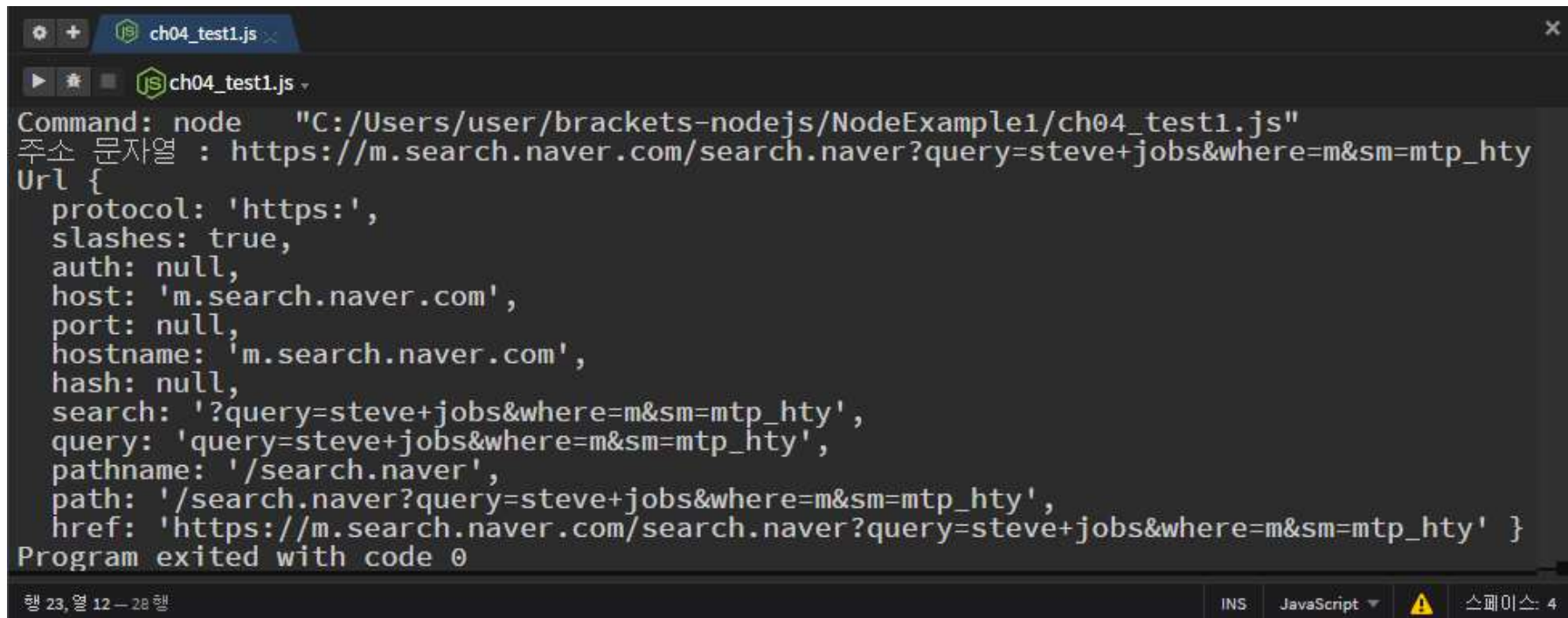
- parse 와 format 사용

메소드 이름	설명
parse()	주소 문자열을 파싱하여 URL 객체를 만들어 줍니다.
format()	URL 객체를 주소 문자열로 변환합니다.

ch04_test01.js

```
var url = require('url');  
var curURL = url.parse('https://m.search.naver.com/search.naver?  
                        query=steve+jobs&where=m&sm=mtp_h ty');  
var curStr = url.format(curURL);  
  
console.log('주소 문자열 : %s', curStr);  
console.dir(curURL);
```

- 파싱된 결과 화면



```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch04_test1.js"
주소 문자열 : https://m.search.naver.com/search.naver?query=steve+jobs&where=m&sm=mtp_h ty
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'm.search.naver.com',
  port: null,
  hostname: 'm.search.naver.com',
  hash: null,
  search: '?query=steve+jobs&where=m&sm=mtp_h ty',
  query: 'query=steve+jobs&where=m&sm=mtp_h ty',
  pathname: '/search.naver',
  path: '/search.naver?query=steve+jobs&where=m&sm=mtp_h ty',
  href: 'https://m.search.naver.com/search.naver?query=steve+jobs&where=m&sm=mtp_h ty' }
Program exited with code 0
```

행 23, 열 12 - 28 행

INS JavaScript 스페이스: 4

- & 기호로 구분되는 요청 파라미터를 분리하는 데 사용
- require 메소드로 모듈을 불러온 후 parse와 stringify 메소드 사용

ch04_test01.js - 추가

...중략

```
var querystring = require('querystring');  
var param = querystring.parse(curURL.query);  
console.log('요청 파라미터 중 query의 값 : %s', param.query);  
console.log('원본 요청 파라미터 : %s', querystring.stringify(param));
```

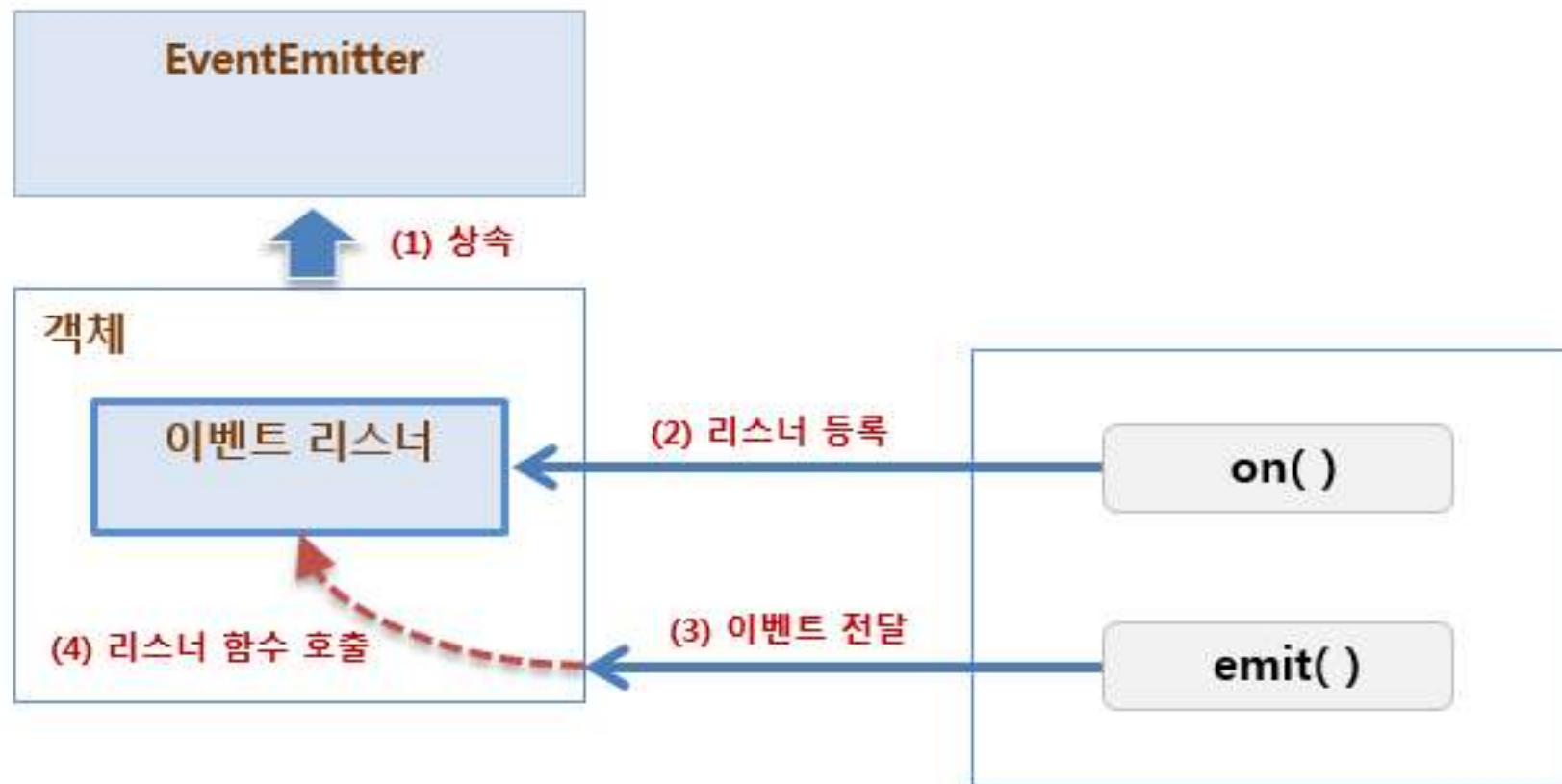
- parse 와 stringify 사용

메소드 이름	설명
parse()	요청 파라미터 문자열을 파싱하여 요청 파라미터 객체를 만들어 줍니다.
stringify()	요청 파라미터 객체를 문자열로 변환합니다.

```
ch04_test1.js
ch04_test1.js
hostname: 'm.search.naver.com',
hash: null,
search: '?query=steve+jobs&where=m&sm=mtp_h ty',
query: 'query=steve+jobs&where=m&sm=mtp_h ty',
pathname: '/search.naver',
path: '/search.naver?query=steve+jobs&where=m&sm=mtp_h ty',
href: 'https://m.search.naver.com/search.naver?query=steve+jobs&where=m&sm=mtp_h ty' }
요청 파라미터 중 query의 값 : steve jobs
원본 요청 파라미터 : query=steve%20jobs&where=m&sm=mtp_h ty
Program exited with code 0
행 27, 열 1 - 28 행
INS JavaScript 스페이스: 4
```

04-02 이벤트 이해하기

- 비동기 방식으로 처리하기 위해 한 쪽에서 다른 쪽으로 데이터 전달
- EventEmitter 사용
 - 한 쪽에서 이벤트를 emit으로 보내고 다른 쪽에서 리스너를 등록하여 on으로 받음



- on 으로 리스너 등록, emit으로 이벤트 전송

메소드 이름	설명
on(event, listener)	지정한 이벤트의 리스너를 추가합니다.
once(event, listener)	지정한 이벤트의 리스너를 추가하지만 한 번 실행한 후에 자동으로 리스너가 제거됩니다.
removeListener(event, listener)	지정한 이벤트에 대한 리스너를 제거합니다.
emit(event, param)	이벤트를 전송합니다.

ch04_test03.js

```
process.on('tick', function(count) {  
  console.log('tick 이벤트 발생함 : %s', count);  
});  
setTimeout(function() {  
  console.log('2초 후에 tick 이벤트 전달 시도함.');
```

```
  process.emit('tick', '2');  
}, 2000);
```

- 별도의 모듈 파일을 만들고 이벤트 처리
 - process 객체에서 처리 시 동일 이름의 이벤트 간에 충돌 가능성 있음
- 계산기 객체가 EventEmitter를 상속하면 emit과 on 메소드 사용 가능

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.');
```

← EventEmitter 는 event 모듈에 정의되어 있음

← Calc 생성자 함수 정의

← stop 이벤트와 리스너 등록

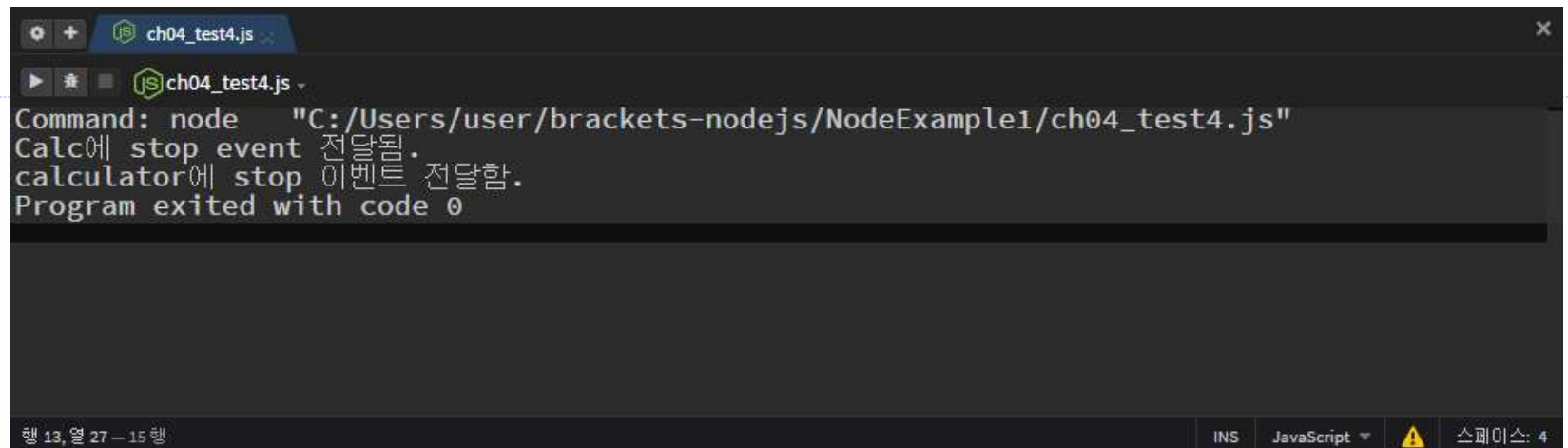
```
  });
};
util.inherits(Calc, EventEmitter);
Calc.prototype.add = function(a, b) {
  return a + b;
}
module.exports = Calc;
module.exports.title = 'calculator';
```

← 이벤트를 처리하도록 EventEmitter 객체를 Calc 객체의 상위 객체로 chaining 구성

- emit 으로 이벤트 전송

ch04_test04.js

```
var Calc = require('./calc3');  
var calc = new Calc();  
calc.emit('stop');  
console.log(Calc.title + '에 stop 이벤트 전달함.');
```



```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch04_test4.js"  
Calc에 stop event 전달됨.  
calculator에 stop 이벤트 전달함.  
Program exited with code 0
```

행 13, 열 27 - 15 행

INS JavaScript 스페이스: 4

04-03 파일 다루기

- fs 모듈 사용
- 동기식 IO와 비동기식 IO 모두 제공
- 동기식 IO
 - 파일 작업이 끝날 때까지 대기한다는 점에 주의
 - 동기식 IO 메소드에서는 Sync 라는 단어가 붙음

ch04_test05.js

```
var fs = require('fs');
```

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

- readFile 메소드 사용하면서 콜백 함수를 파라미터로 전달

ch04_test06.js

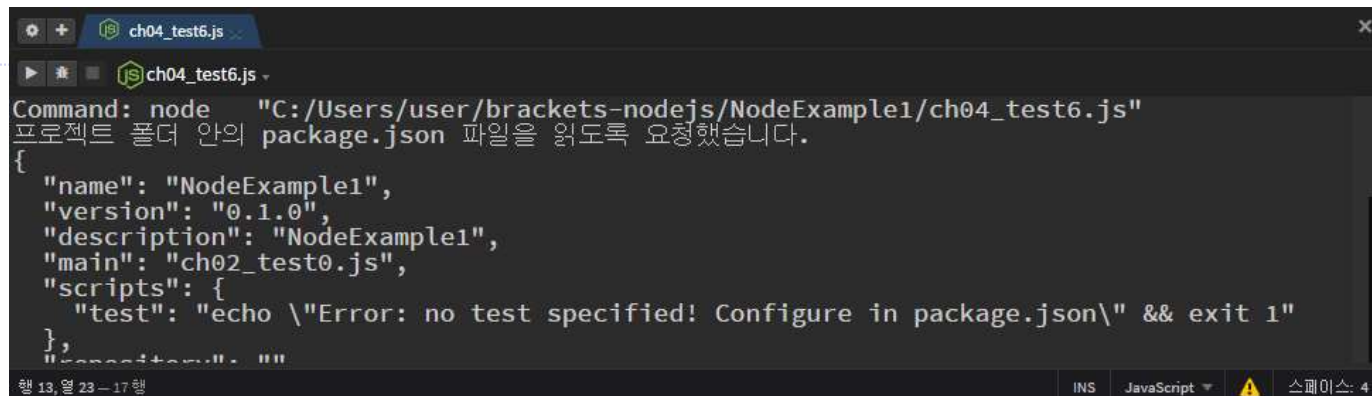
```
var fs = require('fs');
```

```
fs.readFile('./package.json', 'utf8', function(err, data) {  
    console.log(data);  
});
```

```
console.log('프로젝트 폴더 안의 package.json 파일을 읽도록 요청했습니다.');
```

error 객체

- 에러가 있으면 에러데이터 있음
- 에러가 없으면 null

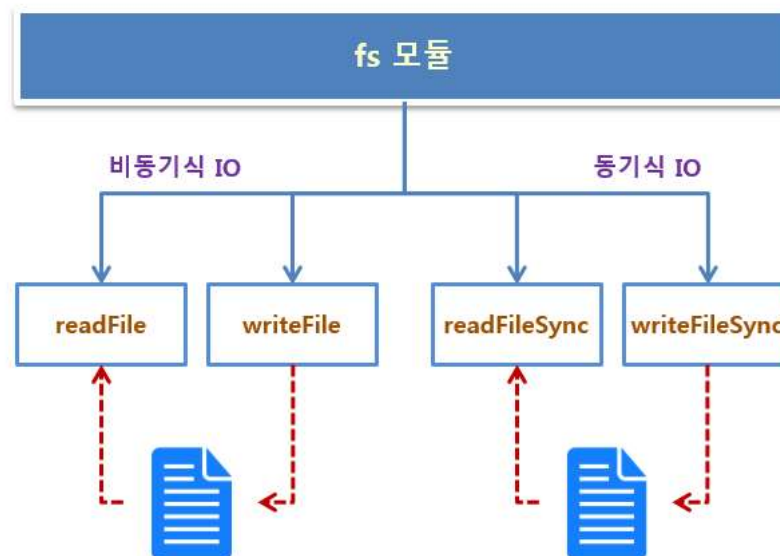


```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch04_test6.js"  
프로젝트 폴더 안의 package.json 파일을 읽도록 요청했습니다.  
{  
  "name": "NodeExample1",  
  "version": "0.1.0",  
  "description": "NodeExample1",  
  "main": "ch02_test0.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified! Configure in package.json\\\" && exit 1"  
  },  
  "repository": ""  
}
```

행 13, 열 23 - 17 행

- readFile로 읽고 writeFile로 쓰기

메소드 이름	설명
<code>readFile(filename, [encoding], [callback])</code>	비동기식 IO로 파일을 읽어 들입니다.
<code>readFileSync(filename, [encoding])</code>	동기식 IO로 파일을 읽어 들입니다.
<code>writeFile(filename, data, encoding='utf8', [callback])</code>	비동기식 IO로 파일을 씁니다.
<code>writeFileSync(filename, data, encoding='utf8')</code>	동기식 IO로 파일을 씁니다.



- readFile 메소드 사용하면서 콜백 함수를 파라미터로 전달

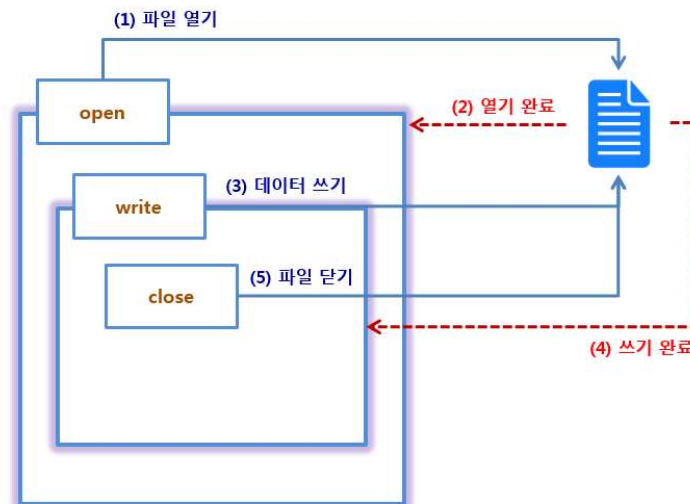
ch04_test07.js

```
var fs = require('fs');
```

```
fs.writeFile('./output.txt', 'Hello World!', function(err) {  
  if(err) {  
    console.log('Error : ' + err);  
  }  
  console.log('output.txt 파일에 데이터 쓰기 완료.');
```

- bulk read/write
 - readFile, readFileSync, writeFile, writeFileSync
- open, read, write, close 등의 메소드
 - 파일의 일부분 read/write, 다양한 파일 작업 필요한 경우 사용

메소드 이름	설명
open(path, flags [, mode] [, callback])	파일을 엽니다.
read(fd, buffer, offset, length, position [, callback])	지정한 부분의 파일 내용을 읽어 들입니다.
write(fd, buffer, offset, length, position [, callback])	파일의 지정한 부분에 데이터를 씁니다.
close(fd [, callback])	파일을 닫아 줍니다.



- open 으로 열고 write로 쓰기

ch04_test08.js

```
var fs = require('fs');
```

----- file flag: w, r, a, w+, r+, a+
↓

```
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() {
```

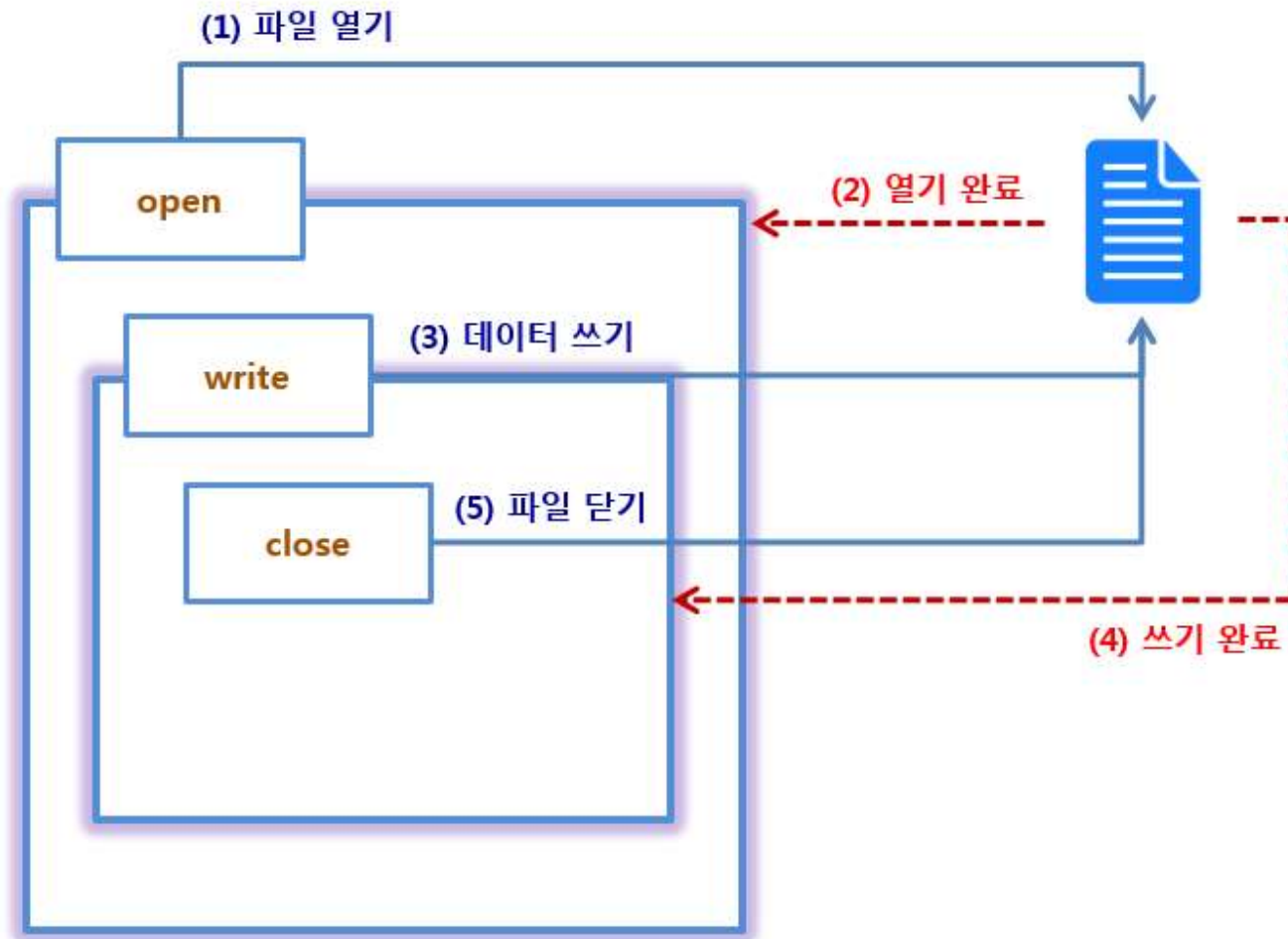
```
    console.log('파일 열고 데이터 쓰고 파일 닫기 완료.');
```

```
  });
```

```
});
```

```
});
```

- open 으로 열고 write로 쓰는 과정



- open 으로 열고 read로 읽기, 버퍼 사용

ch04_test09.js

```
var fs = require('fs');

fs.open('./output.txt', 'r', function(err, fd) {
  if(err) throw err;
  var buf = new Buffer(10);
  console.log('버퍼 타입 : %s', Buffer.isBuffer(buf));
  fs.read(fd, buf, 0, buf.length, null, function(err, bytesRead, buffer) {
    if(err) throw err;
    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 파일을 열고 읽기 완료.');
```

- Buffer 객체

- 주로 바이너리 데이터 읽고 쓰는데 사용
- new로 만들면서 바이트 크기 지정
- Buffer.isBuffer(), Buffer.concat() 등의 메소드 사용 가능

ch04_test10.js

```
var output = '안녕 1!';
var buffer1 = new Buffer(10);
var len = buffer1.write(output, 'utf8');
console.log('첫 번째 버퍼의 문자열 : %s', buffer1.toString());

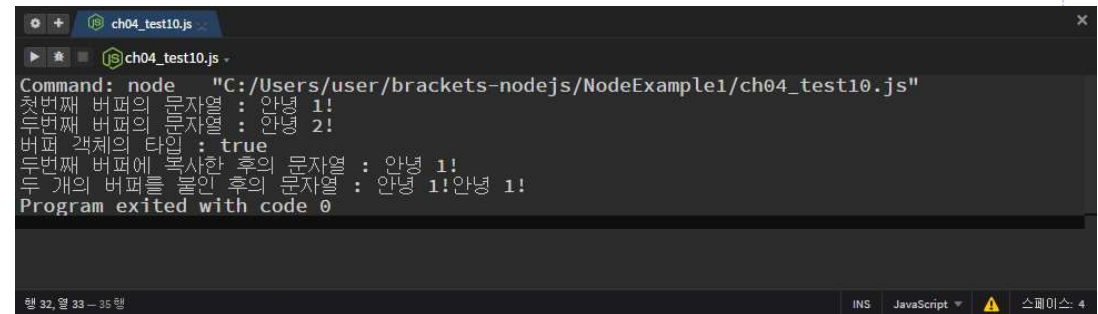
var buffer2 = new Buffer('안녕 2!', 'utf8');
console.log('두 번째 버퍼의 문자열 : %s', buffer2.toString());

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

var byteLen = Buffer.byteLength(output);
var str1 = buffer1.toString('utf8', 0, byteLen);
var str2 = buffer2.toString('utf8');

buffer1.copy(buffer2, 0, 0, len);
console.log('두 번째 버퍼에 복사한 후의 문자열 : %s', buffer2.toString('utf8'));

var buffer3 = Buffer.concat([buffer1, buffer2]);
console.log('두개의 버퍼를 붙인 후의 문자열 : %s', buffer3.toString('utf8'));
```



```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch04_test10.js"
첫번째 버퍼의 문자열 : 안녕 1!
두번째 버퍼의 문자열 : 안녕 2!
버퍼 객체의 타입 : true
두번째 버퍼에 복사한 후의 문자열 : 안녕 1!
두 개의 버퍼를 붙인 후의 문자열 : 안녕 1!안녕 1!
Program exited with code 0
```

- 읽기/쓰기 스트림 만들기

- `createReadStream(path, [options])`
- `createWriteStream(path, [options])`
- `options`
 - `flags`, `encoding`, `autoClose` 등이 있음

ch04_test11.js

```
var fs = require('fs');
```

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

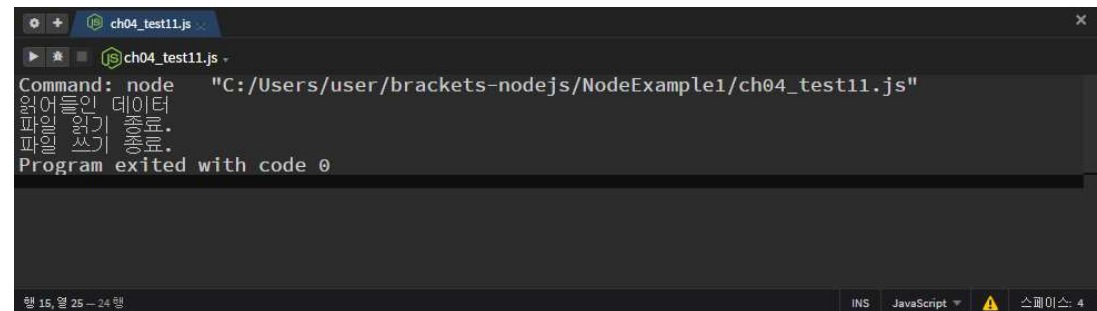
```
var outfile = fs.createWriteStream('./output2.txt', {flags: 'w'});
```

```
infile.on('data', function(data) {  
  console.log('읽어 들인 데이터', data);  
  outfile.write(data);  
});
```

```
infile.on('end', function() {  
  console.log('파일 읽기 종료.');
```

```
  outfile.end(function() {  
    console.log('파일 쓰기 종료.');
```

```
  });
```



```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch04_test11.js"  
읽어들인 데이터  
파일 읽기 종료.  
파일 쓰기 종료.  
Program exited with code 0
```

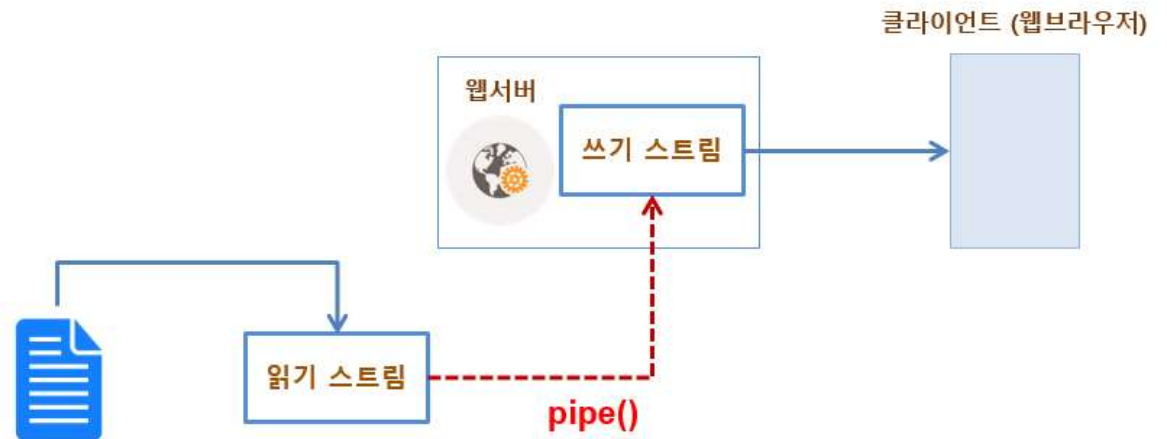
- pipe 메소드
 - 두 개의 스트림을 붙여줌
 - ReadStream과 WriteStream 객체를 붙이면 스트림간에 데이터를 자동 전달
- http 모듈에 대해서는 다시 살펴볼 것임
- 스트림으로 읽어 pipe로 연결함

ch04_test13.js

```
var fs = require('fs');
var http = require('http');

var server = http.createServer(function(req, res) {
  // 파일을 읽어 응답 스트림과 pipe()로 연결합니다.
  var instream = fs.createReadStream('./output.txt');
  instream.pipe(res);
});

server.listen(7001, '127.0.0.1');
```



- mkdir로 만들고 rmdir로 삭제

ch04_test14.js

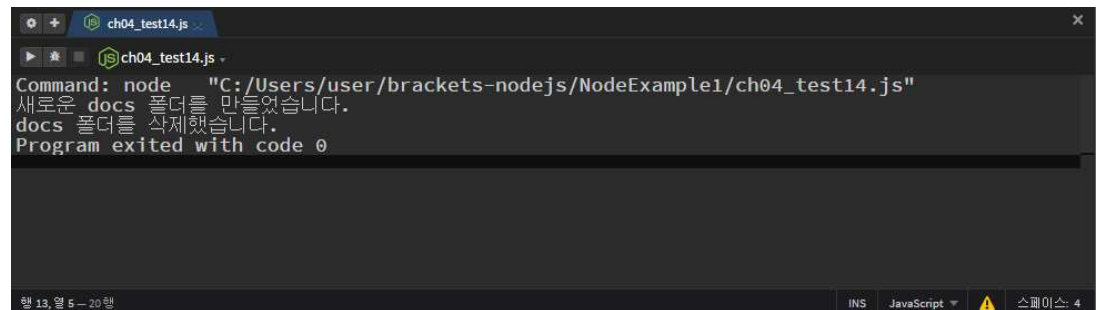
```
var fs = require('fs');  
fs.mkdir('./docs', 0666, function(err) {  
    if(err) throw err;  
    console.log('새로운 docs 폴더를 만들었습니다.');
```



```
fs.rmdir('./docs', function(err) {  
    if(err) throw err;  
    console.log('docs 폴더를 삭제했습니다.');
```



```
});  
});
```



```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch04_test14.js"  
새로운 docs 폴더를 만들었습니다.  
docs 폴더를 삭제했습니다.  
Program exited with code 0
```

04-04 로그파일 남기기

- 로그를 남길 수 있도록 도와주는 모듈에는 여러 가지가 있음
- 실무에서는 일자별로 로그를 남기는 것이 좋음
- winston 모듈
 - 다양한 설정 가능

ch04_test15.js

```
var winston = require('winston');    // 로그 처리 모듈
var winstonDaily = require('winston-daily-rotate-file');
                                // 로그 일별 처리 모듈
var moment = require('moment');      // 시간 처리 모듈

function timeStampFormat() {
    return moment().format('YYYY-MM-DD HH:mm:ss.SSS ZZ');
    // '2016-05-01 20:14:28.500 +0900'
};
```

- 로그 파일이 남는 위치 등 설정 가능

ch04_test15.js - 연속

```
var logger = new (winston.Logger)({
  transports: [
    new (winstonDaily)({
      name: 'info-file',
      filename: './log/server',
      datePattern: '_yyyy-MM-dd.log',
      colorize: false,
      maxsize: 50000000,
      maxFiles: 1000,
      level: 'info',
      showLevel: true,
      json: false,
      timestamp: timeStampFormat
    }),
    new (winston.transports.Console)({
      name: 'debug-console',
      colorize: true,
      level: 'debug',
      showLevel: true,
      json: false,
      timestamp: timeStampFormat
    })
  ],
});
```

- 로그 파일이 남는 위치 등 설정 가능

ch04_test15.js - 연속

```
exceptionHandlers: [  
  new (winstonDaily)({  
    name: 'exception-file',  
    filename: './log/exception',  
    datePattern: '_yyyy-MM-dd.log',  
    colorize: false,  
    maxsize: 50000000,  
    maxFiles: 1000,  
    level: 'error',  
    showLevel: true,  
    json: false,  
    timestamp: timeStampFormat  
  }),  
  new (winston.transports.Console)({  
    name: 'exception-console',  
    colorize: true,  
    level: 'debug',  
    showLevel: true,  
    json: false,  
    timestamp: timeStampFormat  
  })  
]  
});
```

- winston 모듈로 만드는 로거(logger)
 - transports 속성
 - name: 'info-file' 설정
 - 매일 새로운 파일에 로그를 기록하도록 설정
 - level: info 수준의 로그만 기록하도록 설정
 - debug:0 > info:1 > notice:2 > warning:3 > error:4 > crit:5 > alert:6 > emerg:
 - maxsize: 파일의 최대 크기, 넘어가는 경우에 새 파일 생성
 - maxFiles: 생성 가능한 최대 파일의 수
 - name: 'debug-console' 설정
 - 콘솔 창에 출력되는 로그 설정
 - colorsize: 컬러 출력 여부 설정

- 파일 복사하는 코드에서 logger.info() 메소드로 로그 남기기

ch04_test15.js - 연속

```
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;
      logger.info('기존 파일 [' + outname + '] 삭제함.');
```

```
    });
  }

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

  infile.pipe(outfile);
  logger.info('파일 복사 [' + inname + '] -> [' + outname + ']');
```

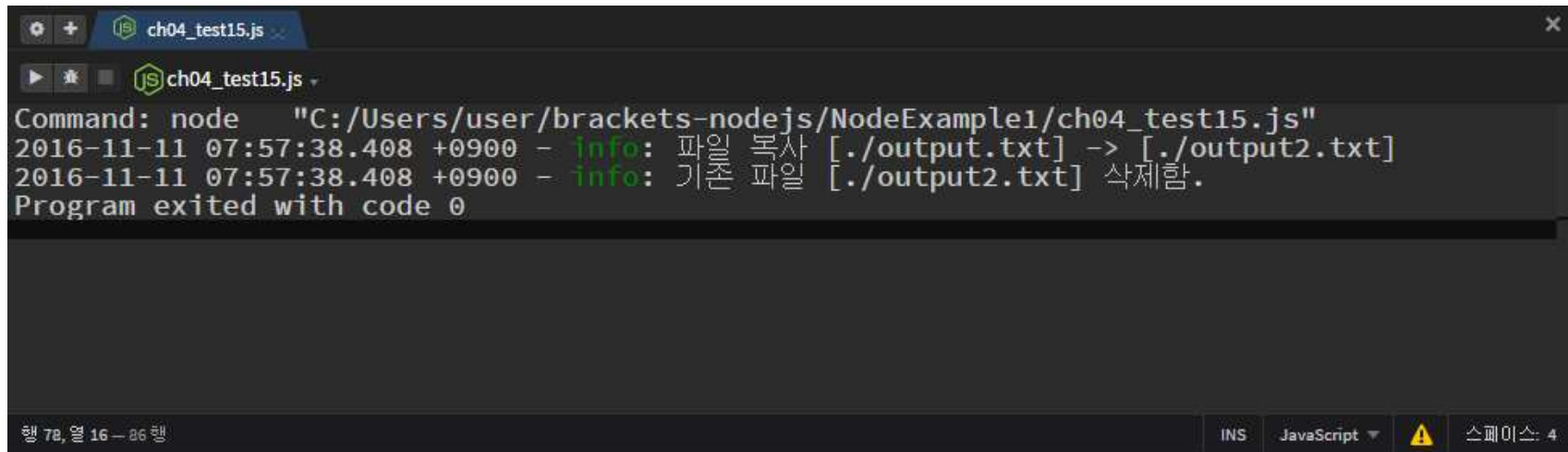
```
});
```

- 외장 모듈 설치

% npm install winston --save

% npm install winston-daily-rotate-file --save

% npm install moment --save



The screenshot shows a code editor window with a file named `ch04_test15.js`. The command being executed is `node "C:/Users/user/brackets-nodejs/NodeExample1/ch04_test15.js"`. The output in the console shows two log messages: `2016-11-11 07:57:38.408 +0900 - info: 파일 복사 [./output.txt] -> [./output2.txt]` and `2016-11-11 07:57:38.408 +0900 - info: 기존 파일 [./output2.txt] 삭제함.`. The program exited with code 0. The status bar at the bottom indicates line 78, column 16, and 86 lines in total. The language is set to JavaScript.

```
Command: node "C:/Users/user/brackets-nodejs/NodeExample1/ch04_test15.js"
2016-11-11 07:57:38.408 +0900 - info: 파일 복사 [./output.txt] -> [./output2.txt]
2016-11-11 07:57:38.408 +0900 - info: 기존 파일 [./output2.txt] 삭제함.
Program exited with code 0
```