

---

## 4장. 노드의 기본 기능 알아보기

# 강의 주제 및 목차

## 강의 주제

### 노드의 기본 기능 알아보기

#### 목 차



1

주소 문자열과 요청 파라미터 다루기

2

이벤트 이해하기

3

파일 다루기

1.

## 주소 문자열과 요청 파라미터 다루기

# 주소 문자열을 구분할 때 사용하는 url 모듈

- 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'



...

# url 모듈의 주요 메소드

- parse 와 format 사용

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

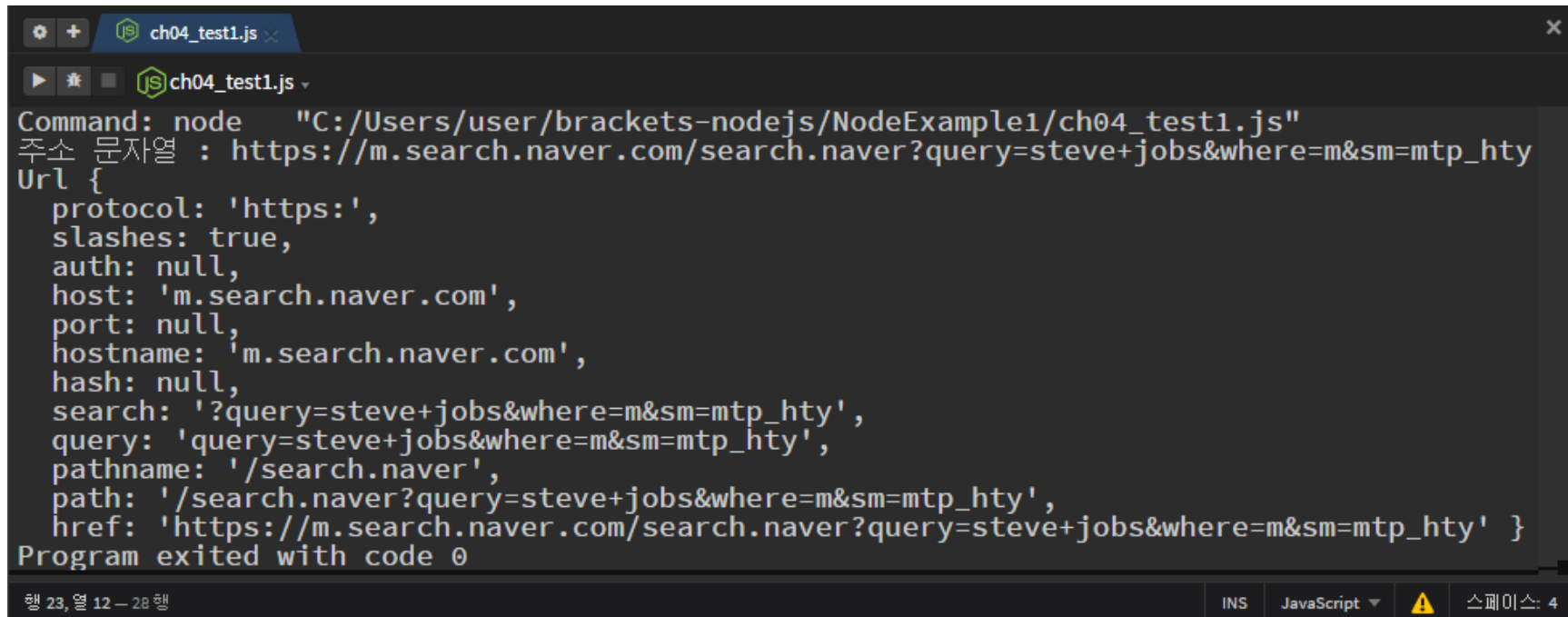
```
var url = require('url');
var curURL = url.parse('https://m.search.naver.com/search.naver?
                        query=steve+jobs&where=m&sm=mtp_h ty');
var curStr = url.format(curURL);

console.log('주소 문자열 : %s', curStr);
console.dir(curURL);
```

ch04\_test1.js~

# parse 메소드로 URL 파싱하기

- 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

# 요청 파라미터를 확인할 때 사용하는 querystring 모듈

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

...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));
```

## ch04\_test1.js url 모듈 사용하기

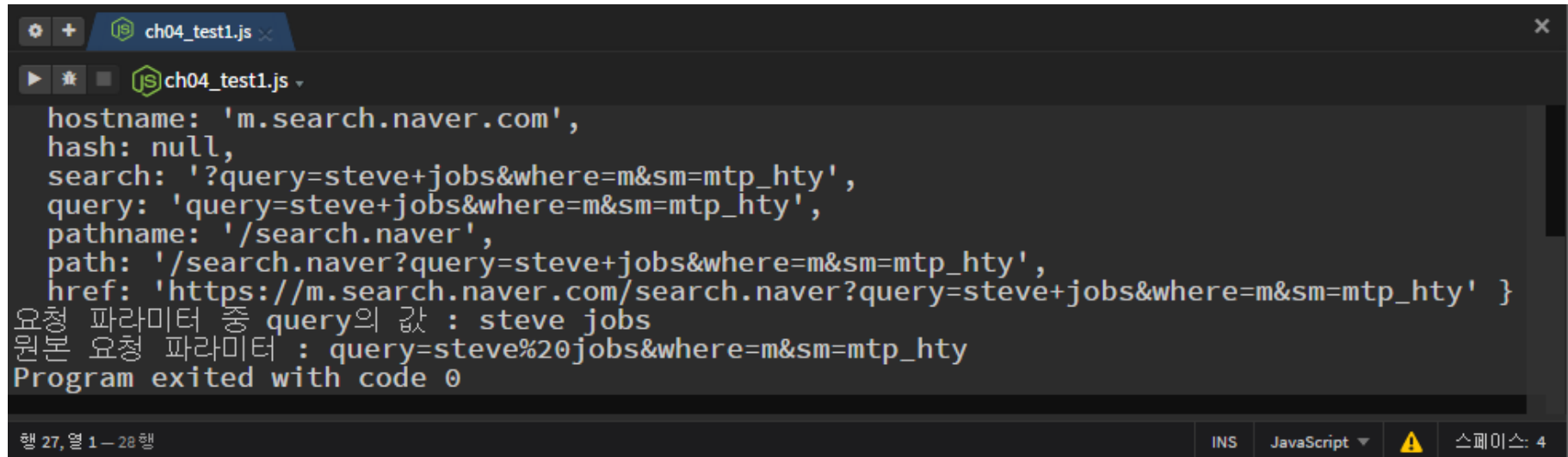
```
var url = require('url');  
// 주소 문자열을 URL 객체로 만들기  
var curURL =  
url.parse('https://search.naver.com/search.naver?sm=top_hy&fbm=1&ie=utf8&query=popcorn');  
  
// URL 객체를 주소 문자열로 만들기  
var curStr = url.format(curURL);  
console.log('주소 문자열 : %s', curStr);  
console.dir(curURL);  
  
// 요청 파라미터 구분하기  
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));
```



# querystring 모듈의 메소드

- 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
>
hostname: 'm.search.naver.com',
hash: null,
search: '?query=steve+jobs&where=m&sm=mtp_hty',
query: 'query=steve+jobs&where=m&sm=mtp_hty',
pathname: '/search.naver',
path: '/search.naver?query=steve+jobs&where=m&sm=mtp_hty',
href: 'https://m.search.naver.com/search.naver?query=steve+jobs&where=m&sm=mtp_hty' }
요청 파라미터 중 query의 값 : steve jobs
원본 요청 파라미터 : query=steve%20jobs&where=m&sm=mtp_hty
Program exited with code 0
```

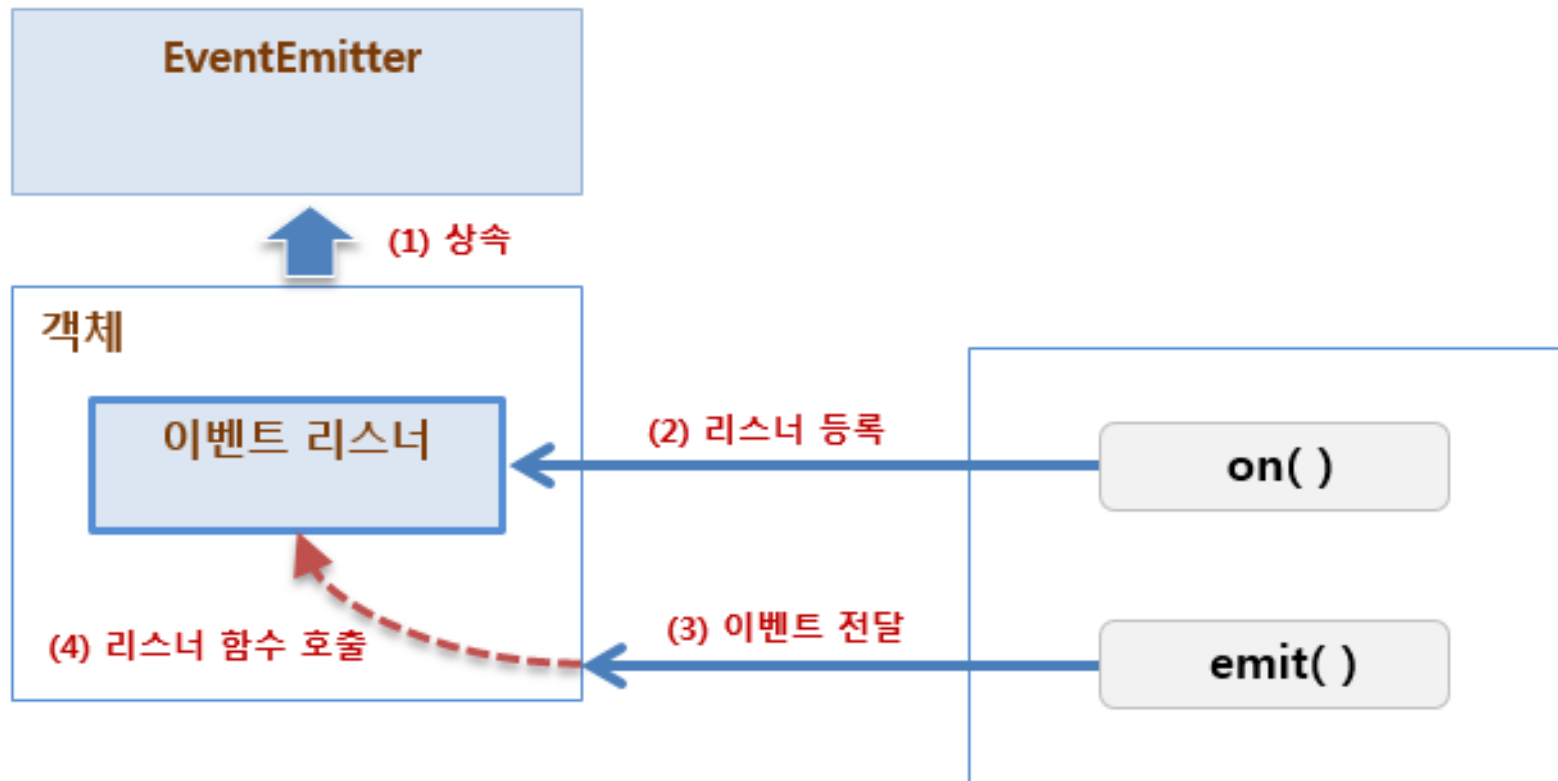
---

2.

이벤트 이해하기

# 이벤트란?

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



# 이벤트 보내고 받기

- 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.

```
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);
```

ch04\_test2.js, ch04\_test3.js

## ch04\_test2.js event 사용하기 1

```
process.on('exit', function() {  
    console.log('exit 이벤트 발생함.');});  
  
setTimeout(function() {  
    console.log('5초 후에 시스템 종료 시도함.');  
    process.exit();  
}, 5000);  
  
console.log('5초 후에 실행될 것임');
```

## ch04\_test3.js event 사용하기 2

```
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);
```

# 계산기 객체를 모듈로 구성

- 계산기 객체가 EventEmitter를 상속하면 emit과 on 메소드 사용 가능

```
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';
```

ch04\_test4.js

## ch04\_test4.js   프로토타입 객체를 만들고 EventEmitter를 상속하도록 하기

---

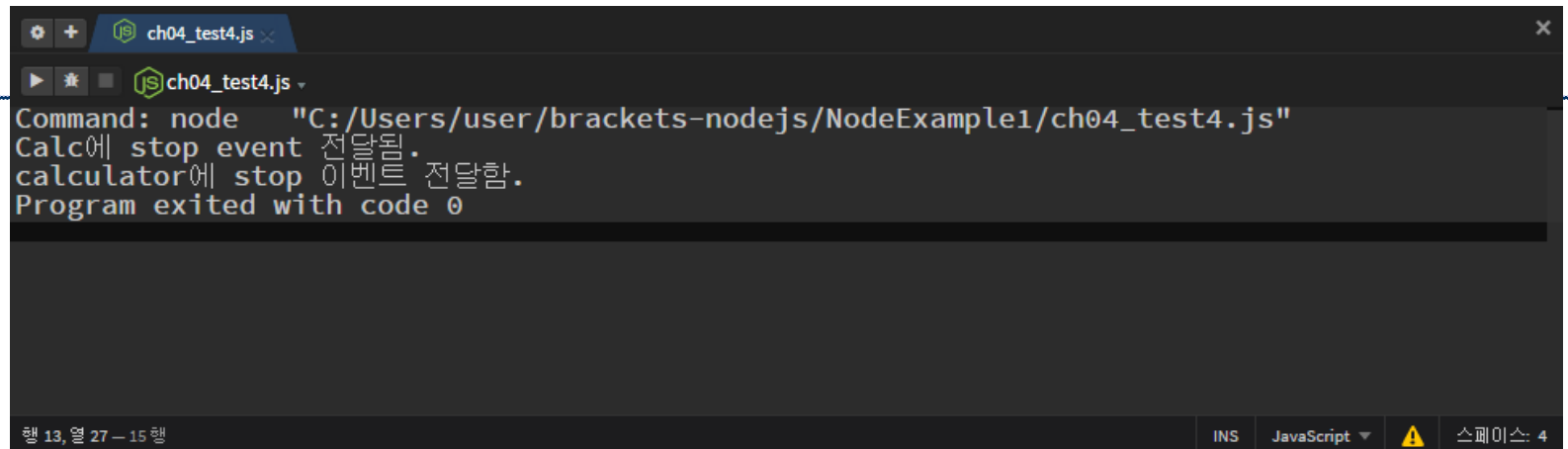
```
var Calc = require('./calc3');  
  
var calc = new Calc();  
calc.emit('stop');  
  
console.log(Calc.title + '에 stop 이벤트 전달함.');
```



# 메인 파일에서 계산기 객체 사용

- emit 으로 이벤트 전송

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



The screenshot shows a terminal window with the following content:

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

At the bottom of the terminal, there is a status bar showing "행 13, 열 27 - 15 행", "INS", "JavaScript", a warning icon, and "스페이스: 4".

calc3.js

## calc3.js 모듈 - 더하기 함수가 들어있는 calc3 모듈

```
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';
```

---

3.

파일 다루기

# 노드의 파일 시스템

---

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

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

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

## ch04\_test5.js FS 사용하기

```
var fs = require('fs');  
  
// 파일을 동기식 IO 방식으로 읽어 들입니다.  
var data = fs.readFileSync('./package.json', 'utf8');  
  
// 읽어 들인 데이터를 출력합니다.  
console.log(data);
```

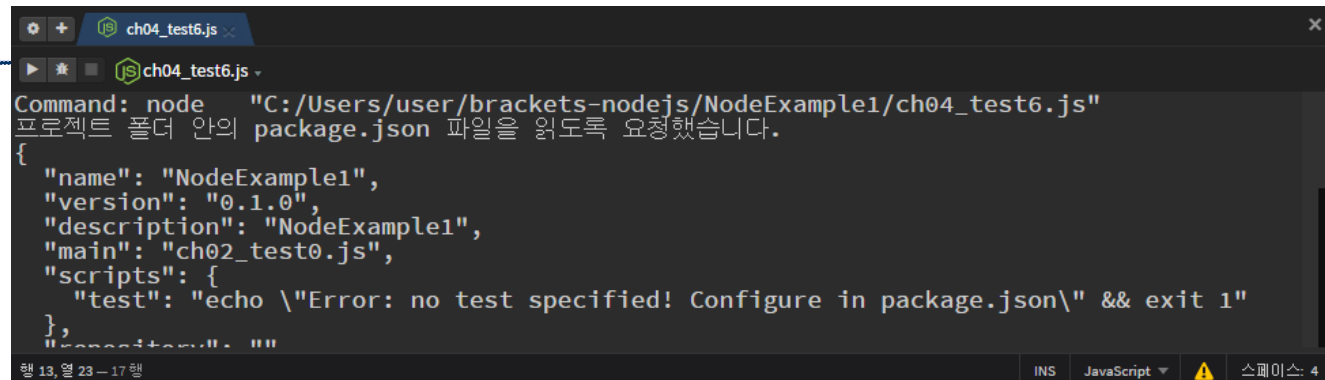
# 비동기식으로 파일 읽기

- readFile 메소드 사용하면서 콜백 함수를 파라미터로 전달

```
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.');
```



```
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": ""  
}
```

ch04\_test6.js

## ch04\_test6.js FS 사용하기 : Non-Blocking IO

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

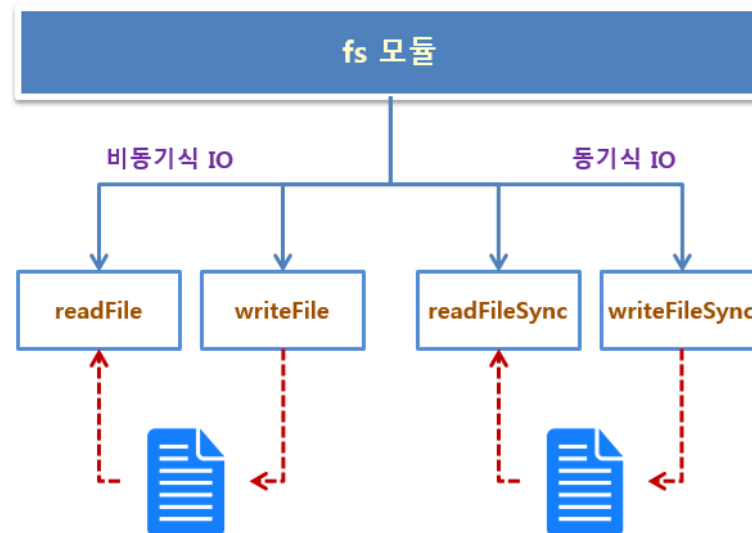
//파일을 비동기식 IO 방식으로 읽어 들입니다.
fs.readFile('./package.json', 'utf8', function(err, data) {
    // 읽어 들인 데이터를 출력합니다.
    console.log(data);
});

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

# fs 모듈의 주요 메소드

- 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.





# 비동기식으로 파일 쓰기

- readFile 메소드 사용하면서 콜백 함수를 파라미터로 전달

```
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.');
```

```
});
```

## ch04\_test7.js FS 사용하기 : 파일 쓰기

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

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

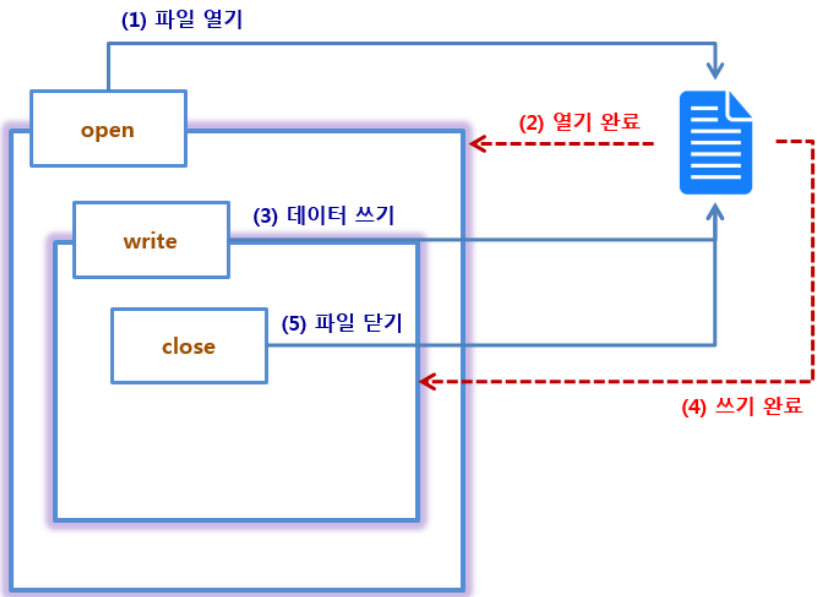
    console.log('output.txt 파일에 데이터 쓰기 완료.');
```

```
});
```

# 파일을 직접 열고 닫으면서 읽거나 쓰기

- open, read, write, close 등의 메소드가 사용됨

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.



# 파일을 직접 열고 데이터 쓰기

- open 으로 열고 write로 쓰기

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

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.');
```

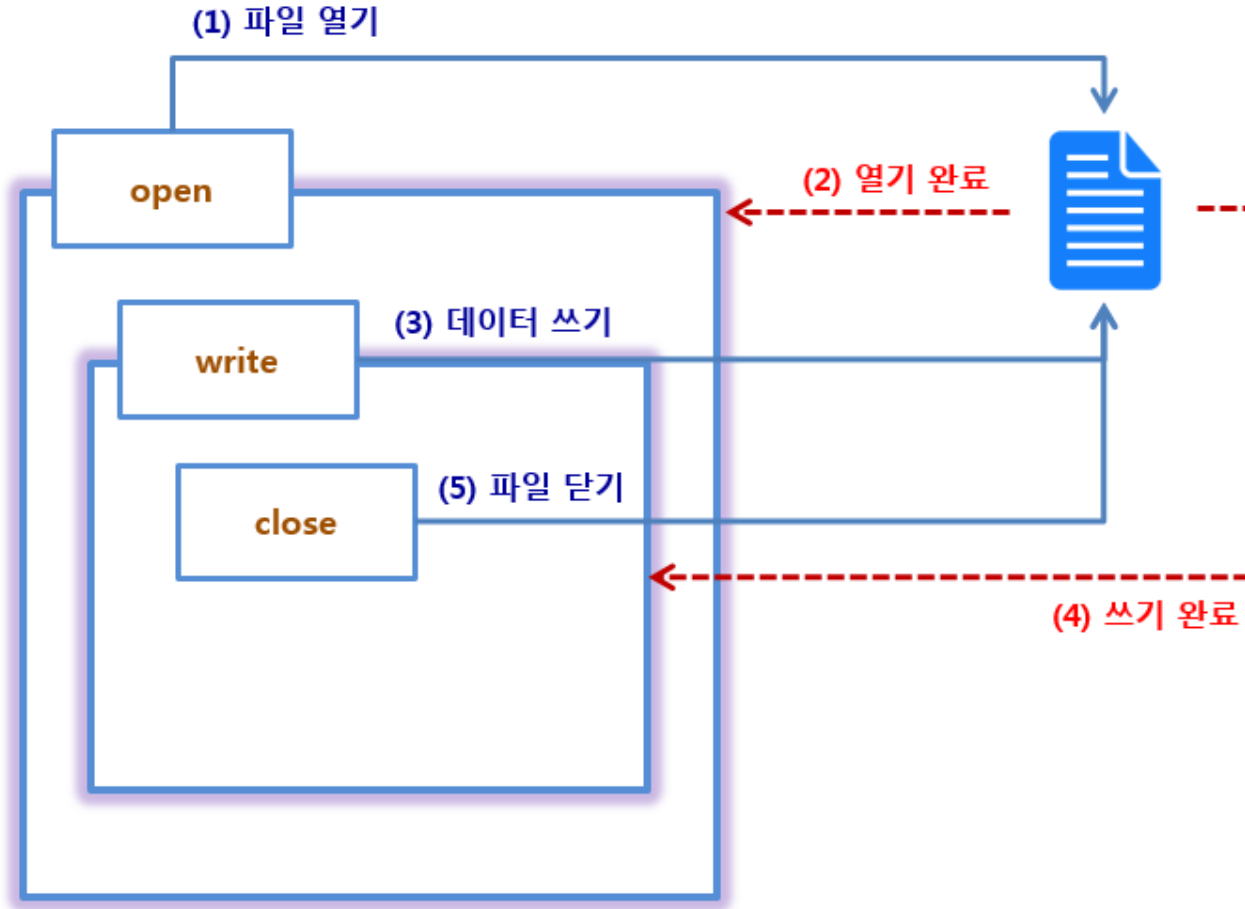
ch04\_test8.js

## ch04\_test8.js FS 사용하기 : 파일을 열어 데이터를 쓰고 파일 닫기

```
var fs = require('fs');  
  
//파일에 데이터를 씁니다.  
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.');        });  
    });  
});
```

# 파일 처리 플로우

- open 으로 열고 write로 쓰는 과정



# 파일 직접 열고 읽기

- open 으로 열고 read로 읽기, 버퍼 사용

```
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.');
```

ch04\_test9.js

## ch04\_test9.js FS 사용하기 : 파일의 일부 내용 읽어들이기

```
var fs = require('fs');
//파일에서 데이터를 읽어 들입니다.
fs.open('./output.txt', 'r', function(err, fd) {
    if(err) throw err;

    var buf = new Buffer(10);
    console.log('버퍼 타입 : ', 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.');
```



# 버퍼 사용하기

- new로 만들고 Buffer.isBuffer(), Buffer.concat() 등의 메소드 사용 가능

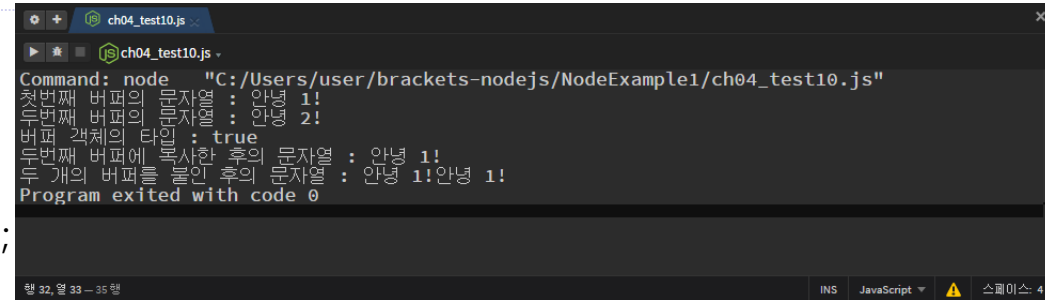
```
var output = '안녕 1!';  
var buffer1 = new Buffer(10);  
var len = buffer1.write(output, 'utf8');  
console.log('첫 번째 버퍼의 문자열 : %s', buffer1.toString());
```

```
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'));
```



```
ch04_test10.js  
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  
행 32, 열 33 - 35 행  
INS JavaScript 스킴이스 4
```

ch04\_test10.js

## ch04\_test10.js FS 사용하기 : Buffer 객체 이해하기

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

// 버퍼 객체를 문자열을 이용해 만듭니다.
var buffer2 = new Buffer('안녕 2!', 'utf8');
console.log('두번째 버퍼의 문자열 : %s', buffer2.toString());

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

// 버퍼 객체에 들어있는 문자열 데이터를 문자열 변수로 만듭니다.
var byteLen = Buffer.byteLength(output);
var str1 = buffer1.toString('utf8', 0, byteLen);
var str2 = buffer2.toString('utf8');

// 두번째 버퍼 객체의 문자열을 첫 번째 버퍼 객체로 복사합니다.
buffer1.copy(buffer2, 0, 0, len);
console.log('두번째 버퍼에 복사한 후의 문자열 : %s', buffer2.toString('utf8'));

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

# 스트림 단위로 파일 읽고 쓰기

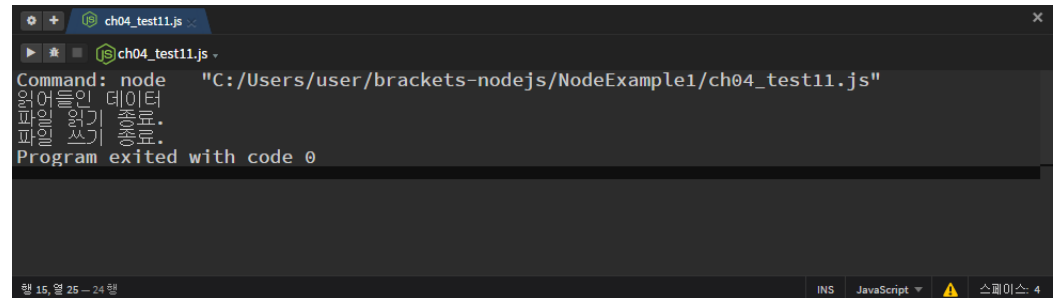
- createReadStream으로 읽기 위해 열고, createWriteStream으로 쓰기 위해 열기

```
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.');
```



ch04\_test11.js

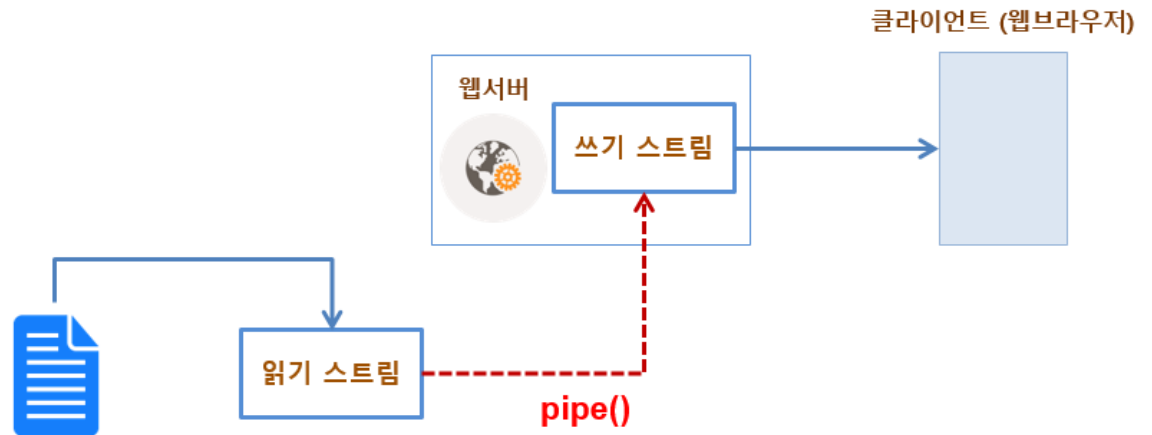
## ch04\_test11.js FS 사용하기 : 한 파일을 스트림으로 읽어 다른 파일에 쓰기

```
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('읽어들인 데이터', data);  
    outfile.write(data);  
});  
  
infile.on('end', function() {  
    console.log('파일 읽기 종료.');    outfile.end(function() {  
        console.log('파일 쓰기 종료.');    });  
});
```

# http 모듈로 요청받은 파일 내용을 읽고 응답하기

- http 모듈에 대해서는 다시 살펴볼 것임
- 스트림으로 읽어 pipe로 연결함

```
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');
```



ch04\_test12.js

## ch04\_test12.js FS 사용하기 : 파이프 사용하기

```
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('기존 파일 [' + 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 + ']');  
});
```

# 새 디렉터리 만들고 삭제하기

- mkdir로 만들고 rmdir로 삭제

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

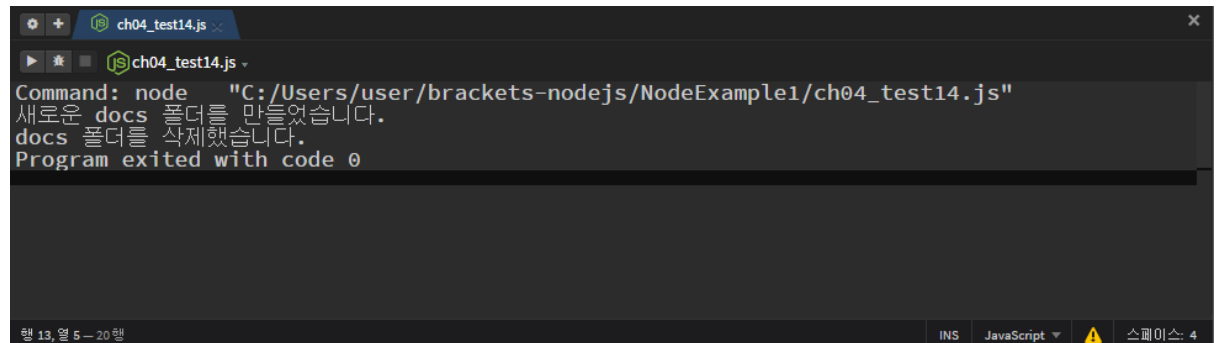
```
fs.rmdir('./docs', function(err) {
  if(err) throw err;
  console.log('docs 폴더를 삭제했습니다.');
```

```
});
```

```
});
```



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

ch04\_test13js

## ch04\_test13.js FS 사용하기 : 한 파일을 스트림으로 읽어 다른 파일에 쓰기

```
var fs = require('fs');  
  
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.');    });*/  
});
```



