

네트워크서비스 프로토콜

# 6주차 2

» 소프트웨어학부

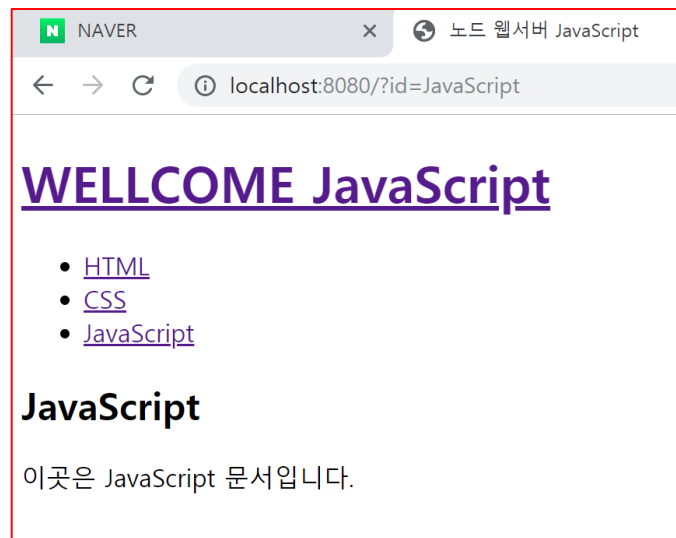
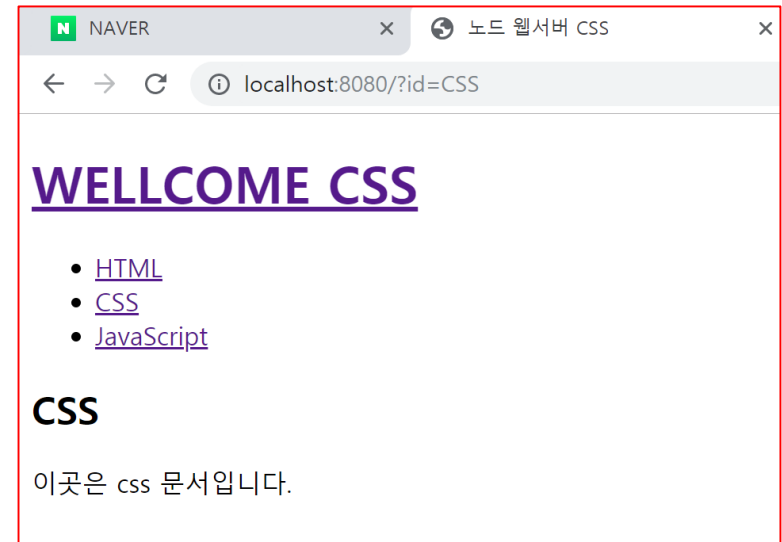
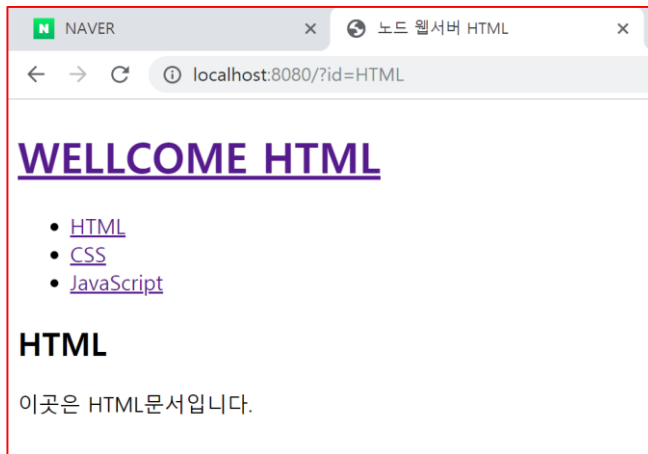
» 김형균 교수



# 수업에 들어가며

- 복습
  - 본문 내용도 동적으로 변하는 웹페이지
- 오늘 학습내용
  - fs.readdir
  - fs.readdir 활용 글목록 출력
  - 학습 참여도 평가
  - querystring
  - 이벤트 이해하기
  - 버퍼와 스트림 이해하기
  - Buffer의 메서드
  - 버퍼 사용하기
  - 파일 읽는 스트림 사용하기
  - 폼(form)
  - get방식, post방식

# 복습 본문 내용도 동적으로 변하는 웹페이지





# 본문 내용도 동적으로 변하는 웹페이지


## » fs 파일 시스템에 접근하는 모듈


- 파일/폴더 생성, 삭제, 읽기, 쓰기 가능
- 웹 브라우저에서는 제한적이었으나 노드는 권한을 가지고 있음
- 형식) `fs.readFile(filename, [options], callback);`


## » 본문의 내용을 다음과 같이 파일형태로 구분 저장

C > 로컬 디스크 (C:) > test > data

이름

 javascript

 css

 html

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

var app = http.createServer(function(request, response){
  var _url = request.url;
  var queryData = url.parse(_url, true).query;
  var pathname = url.parse(_url, true).pathname;

  if(pathname == '/'){
    if(queryData.id == undefined){ ...
    } else {
      fs.readFile(`data/${queryData.id}`, 'utf8', function(err, description){ ...
      });
    }
  } else {
    response.writeHead(404);
    response.end('Not found');
  }
});
app.listen(3000);
```



```
if(pathname == '/'){
  if(queryData.id == undefined){
    var title = 'Home';
    var description = 'Hello, Node.js';
    var template = `
      <!doctype html>
      <html>
        <head>
          <title>노드 웹서버 ${title}</title>
          <meta charset="utf-8">
        </head>
        <body>
          <h1><a href="/">WELLCOME ${title}</a></h1>
          <ul>
            <li><a href="/?id=HTML">HTML</a></li>
            <li><a href="/?id=CSS">CSS</a></li>
            <li><a href="/?id=JavaScript">JavaScript</a></li>
          </ul>
          <h2>${title}</h2>
          <p>${description}</p>
        </body>
      </html>
    `;
    response.writeHead(200);
    response.end(template);
  }
```

```
  } else {
    fs.readFile(`data/${queryData.id}`, 'utf8', function(err, description){
      var title = queryData.id;
      var template = `
        <!doctype html>
        <html>
          <head>
            <title>노드 웹서버 ${title}</title>
            <meta charset="utf-8">
          </head>
          <body>
            <h1><a href="/">WELLCOME ${title}</a></h1>
            <ul>
              <li><a href="/?id=HTML">HTML</a></li>
              <li><a href="/?id=CSS">CSS</a></li>
              <li><a href="/?id=JavaScript">JavaScript</a></li>
            </ul>
            <h2>${title}</h2>
            <p>${description}</p>
          </body>
        </html>
      `;
      response.writeHead(200);
      response.end(template);
    });
  }
}
```



# fs.readdir

» 디렉토리에 있는 아이템(파일, 디렉토리 등)의 이름들을 리턴해준다.

» `fs.readdir(path[, options], callback)`

- `path` `<string>` | `<Buffer>` | `<URL>`
- `options` `<string>` | `<Object>`
  - `encoding` `<string>` Default: 'utf8'
  - `withFileTypes` `<boolean>` Default: false
- `callback` `<Function>`
- `err` `<Error>`
- `files` `<string[]>` | `<Buffer[]>` | `<fs.Dirent[]>`

```
var testFolder = './data';  
var fs = require('fs');  
  
fs.readdir(testFolder, function(error, filelist){  
  console.log(filelist);  
})
```

```
C:\Program Files\nodejs\node.exe --inspect-brk=28481 readdir.js  
Debugger listening on ws://127.0.0.1:28481/4c4587a4-9e8b-4aa7-9a  
For help, see: https://nodejs.org/en/docs/inspector  
Debugger attached.
```

```
> Array(3) ["css", "html", "javascript"]
```





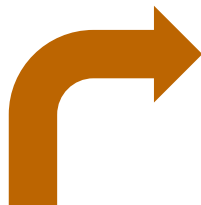
# fs.readdir 활용 글목록 출력

» data 디렉토리에 있는 파일들의 이름을 이용해서 글 목록(ul태그의 <li> 목록)을 js코드로 생성하는 기능을 구현

```
> Array(3) ["css", "html", "javascript"]
```

```
<ul>
  <li><a href="/?id=HTML">HTML</a></li>
  <li><a href="/?id=CSS">CSS</a></li>
  <li><a href="/?id=JavaScript">JavaScript</a></li>
</ul>
```

# 전체 구조의 변화



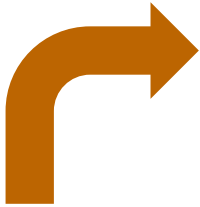
```
var app = http.createServer(function(request,response){
  var _url = request.url;
  var queryData = url.parse(_url, true).query;
  var pathname = url.parse(_url, true).pathname;
  if(pathname === '/'){
    if(queryData.id === undefined){
      fs.readdir('./data', function(error, filelist){...
    })
  } else {
    fs.readdir('./data', function(error, filelist){...
  });
}
} else {
  response.writeHead(404);
  response.end('Not found');
}
});
app.listen(3000);
```

```
var app = http.createServer(function(request,response){
  var _url = request.url;
  var queryData = url.parse(_url, true).query;
  var pathname = url.parse(_url, true).pathname;

  if(pathname === '/'){
    if(queryData.id == undefined){...
  } else {
    fs.readFile(`data/${queryData.id}`, 'utf8', function(err, description){...
    });
  }
} else {
  response.writeHead(404);
  response.end('Not found');
}
});
app.listen(3000);
```




# 글목록을 list 변수화 처리하기



```
<body>
  <h1><a href="/">WELLCOME ${title}</a></h1>
  ${list}
  <h2>${title}</h2>
  <p>${description}</p>
</body>
```

```
<body>
<h1><a href="/">WELLCOME ${title}</a></h1>
<ul>
  <li><a href="/?id=HTML">HTML</a></li>
  <li><a href="/?id=CSS">CSS</a></li>
  <li><a href="/?id=JavaScript">JavaScript</a></li>
</ul>
<h2>${title}</h2>
<p>${description}</p>
</body>
```



```
var app = http.createServer(function(request,response){
  var _url = request.url;
  var queryData = url.parse(_url, true).query;
  var pathname = url.parse(_url, true).pathname;
  if(pathname === '/'){
    if(queryData.id === undefined){
      var title = 'Home';
      var description = 'Hello, Node.js';
      var template = `
```

```
var app = http.createServer(function(request,response){
  var _url = request.url;
  var queryData = url.parse(_url, true).query;
  var pathname = url.parse(_url, true).pathname;

  if(pathname === '/'){
    if(queryData.id == undefined){
      var title = 'Home';
      var description = 'Hello, Node.js';
      var template = `
```

```
if(pathname === '/'){\n  if(queryData.id === undefined){
```

```
} else {
```

```
} else {
```



```
fs.readFile(`data/${queryData.id}`, 'utf8', function(err, description){
  var title = queryData.id;
  var template = `
<!doctype html>
<html>
  <head>
    <title>노드 웹서버 ${title}</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1><a href="/">WELLCOME ${title}</a></h1>
    ${list}
    <h2>${title}</h2>
    <p>${description}</p>
  </body>
</html>
`;
  response.writeHead(200);
  response.end(template);
});
```

```
});
```

```
}
```

```
} else {
```

# 학습 참여도 평가 문제

## - 함수를 이용해서 정리 정돈하기



```
var app = http.createServer(function(request,response){
  var _url = request.url;
  var queryData = url.parse(_url, true).query;
  var pathname = url.parse(_url, true).pathname;
  if(pathname == '/'){
    if(queryData.id == undefined){
      fs.readdir('./data', function(error, filelist){
        var title = 'Home';
        var description = 'Hello, Node.js';
        var list = templateList(filelist);
        var template = templateHTML(title, list, `

## ${title}</h2>${description}`); response.writeHead(200); response.end(template); }) } else { fs.readdir('./data', function(error, filelist){ fs.readFile(`data/${queryData.id}`, 'utf8', function(err, description){ var title = queryData.id; var list = templateList(filelist); var template = templateHTML(title, list, `${title}</h2>${description}`); response.writeHead(200); response.end(template); }); }); } } else { response.writeHead(404); response.end('Not found'); } }); app.listen(3000);


```



```
function templateHTML(title, list, body){
```



```
}
```





```
function templateList(filelist){
```

```
}
```



## 12. querystring

» 기존 노드 방식에서는 url querystring을 querystring 모듈로 처리

- querystring.parse(쿼리): url의 query 부분을 자바스크립트 객체로 분해해줍니다.
- querystring.stringify(객체): 분해된 query 객체를 문자열로 다시 조립해줍니다.

```
1  const url = require('url');
2  const querystring = require('querystring');
3
4  const parsedUrl = url.parse('https://www.kookmin.ac.kr/site/search/search.htm?keyword=복지관&x=20&y=12');
5  const query = querystring.parse(parsedUrl.query);
6  console.log('querystring.parse():', query);
7  console.log('querystring.stringify():', querystring.stringify(query));
8  |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

C:\Program Files\nodejs\node.exe --inspect-brk=36201 nodejs-book-master\ch3\3.5\querystring.js

Debugger listening on ws://127.0.0.1:36201/093d81dd-f823-40ad-ad59-974432366890

For help, see: <https://nodejs.org/en/docs/inspector>

Debugger attached.

querystring.parse():

Object {keyword: "복지관", x: "20", y: "12"}

querystring.stringify(): keyword=%EB%B3%B5%EC%A7%80%EA%B4%80&x=20&y=12

que

que

que



# 이벤트 이해 하기

» Node 에서 이벤트 처리 하는 EventEmitter 라는것이 만들어져 있음.

» 이벤트 보내고 받기

- 노드의 객체는 EventEmitter를 상속 받을 수 있으며, 상속 후에는 EventEmitter 객체의 on() 과 emit() 메소드 사용 가능
- on() 메소드 = 이벤트가 전달될 객체에 이벤트 리스너를 설정하는 역할. 보통은 노드 내부에서 미리 만들어 제공하는 이벤트를 받아 처리하지만, 필요할 때는 직접 이벤트를 만들어 전달 할 수도 있다.
- once() 메소드 = 이벤트 리스너 함수가 한번이라도 실행하고 나면 자동으로 제거 되므로 이벤트를 딱 한번만 받아서 처리 할 수 있음.
- emit() 메소드 = 이벤트를 다른쪽으로 전달하고 싶을 경우

» 이벤트를 처리하는 EventEmitter의 주요 메소드

- on(event, 콜백)
  - 지정한 이벤트의 리스너를 추가합니다.
  - 지정한 Event 생성하고 on 메소드를 이용하여 이벤트를 등록하면
  - 이 메소드를 호출하면서 파라미터로 전달한 이벤트가 발생했을 때
  - 그다음에 나오는 콜백 함수가 실행 된다.



```
process.on('exit',function(){
  console.log('exit 이벤트 발생함');
});
// process 객체는 내부적으로 EventEmitter 를 상속받도록 만들어져 있어
// on() emit()메소드 바로 사용 가능

setTimeout(function(){
  console.log('2초 후에 시스템 종료 시도함');
  process.exit();
},2000);
```

# 7. 버퍼와 스트림 이해하기

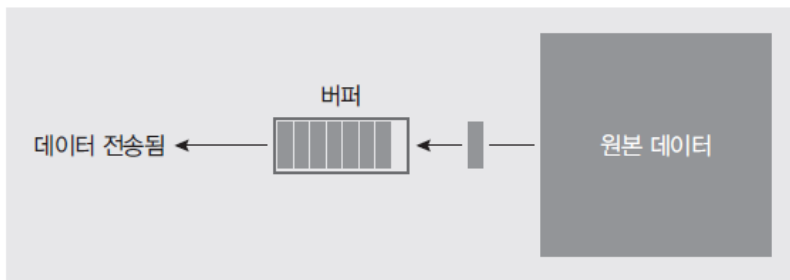
## » 버퍼: 일정한 크기로 모아두는 데이터

- 일정한 크기가 되면 한 번에 처리
- 버퍼링: 버퍼에 데이터가 찰 때까지 모으는 작업

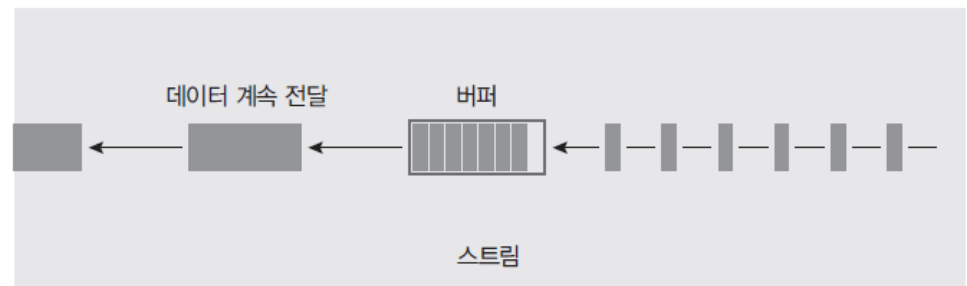
## » 스트림: 데이터의 흐름

- 일정한 크기로 나눠서 여러 번에 걸쳐서 처리
- 버퍼(또는 청크)의 크기를 작게 만들어서 주기적으로 데이터를 전달
- 스트리밍: 일정한 크기의 데이터를 지속적으로 전달하는 작업

▼ 그림 3-11 버퍼



▼ 그림 3-12 스트림





## 8. Buffer의 메서드

» 노드에서는 Buffer 객체 사용

- from(문자열)
  - 문자열을 버퍼로 바꿀 수 있습니다.
  - length 속성은 버퍼의 크기를 알려줍니다. 바이트 단위입니다.
- toString(버퍼)
  - 버퍼를 다시 문자열로 바꿀 수 있습니다.
  - 이때 base64나 hex를 인자로 넣으면 해당 인코딩으로도 변환할 수 있습니다.
- concat(배열)
  - 배열 안에 든 버퍼들을 하나로 합칩니다.
- alloc(바이트)
  - 빈 버퍼를 생성합니다.
  - 바이트를 인자로 지정해주면 해당 크기의 버퍼가 생성됩니다.



# 9. 버퍼 사용하기

## » 노드에서는 Buffer 객체 사용

buffer.js

```
const buffer = Buffer.from('저를 버퍼로 바꿔보세요');
console.log('from():', buffer);
console.log('length:', buffer.length);
console.log('toString():', buffer.toString());

const array = [Buffer.from('띄엄 '), Buffer.from('띄엄 '), Buffer.from('띄어쓰기')];
const buffer2 = Buffer.concat(array);
console.log('concat():', buffer2.toString());

const buffer3 = Buffer.alloc(5);
console.log('alloc():', buffer3);
```

콘솔

```
$ node buffer
from(): <Buffer ec a0 80 eb a5 bc 20 eb b2 84 ed 8d bc eb a1 9c 20 eb b0 94 ea bf 94 eb
b3 b4 ec 84 b8 ec 9a 94>
length: 32
toString(): 저를 버퍼로 바꿔보세요
concat(): 띄엄 띄엄 띄어쓰기
alloc(): <Buffer 00 00 00 00 00>
```

# 10. 파일 읽는 스트림 사용하기

## » fs.createReadStream

- createReadStream에 인자로 파일 경로와 옵션 객체 전달
- highWaterMark 옵션은 버퍼의 크기(바이트 단위, 기본값 64KB)
- **data**(chunk 전달), **end**(전달 완료), **error**(에러 발생) 이벤트 리스너와 같이 사용
- data 이벤트 : 파일의 일부를 리턴한다.
- end 이벤트 : 읽기가 완료되었을때 호출된다.
- error 이벤트 : 에러가 발생했을때 호출된다.





## 10. 파일 읽는 스트림 사용하기

readme3.txt

저는 조금씩 조금씩 나눠서 전달됩니다. 나눠진 조각을 chunk라고 부릅니다.

createReadStream.js

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

```
const readStream = fs.createReadStream('./readme3.txt', { highWaterMark: 16 });
```

```
const data = [];
```

```
readStream.on('data', (chunk) => {  
  data.push(chunk);  
  console.log('data :', chunk, chunk.length);  
});
```

```
readStream.on('end', () => {  
  console.log('end :', Buffer.concat(data).toString());  
});
```

```
readStream.on('error', (err) => {  
  console.log('error :', err);  
});
```

콘솔

```
$ node createReadStream
```

```
data : <Buffer ec a0 80 eb 8a 94 20 ec a1 b0 ea b8 88 ec 94 a9> 16
```

```
data : <Buffer 20 ec a1 b0 ea b8 88 ec 94 a9 20 eb 82 98 eb 88> 16
```

```
data : <Buffer a0 ec 84 9c 20 ec a0 84 eb 8b ac eb 90 a9 eb 8b> 16
```

```
data : <Buffer 88 eb 8b a4 2e 20 eb 82 98 eb 88 a0 ec a7 84 20> 16
```

```
data : <Buffer ec a1 b0 ea b0 81 ec 9d 84 20 63 68 75 6e 6b eb> 16
```

```
data : <Buffer 9d bc ea b3 a0 20 eb b6 80 eb a6 85 eb 8b 88 eb> 16
```

```
data : <Buffer 8b a4 2e> 3
```

```
end : 저는 조금씩 조금씩 나눠서 전달됩니다. 나눠진 조각을 chunk라고 부릅니다.
```



# 폼(form)

» 폼이란 사용자의 데이터를 서버에 저장할 때 사용하는 방법이다. 보통 로그인, 회원가입, 게시판에 글을 작성하거나 파일을 전송할 때 사용하게 된다.

» 문법

```
<form action="URL" method="데이터를 전송하는 방법">
```

```
<!--텍스트,라디오,체크박스과 같은 컨트롤을 생성-->
```

```
</form>
```

- action : 사용자가 입력한 데이터를 전송할 서버의 URL
- method : 사용자가 입력한 데이터를 전송하는 방법으로 대표적으로 get방식과 post방식이 있음.
  - get방식 : action에 입력한 URL에 파라미터 형태로 전송(생략가능)
  - post방식 : header의 body에 포함해서 전송



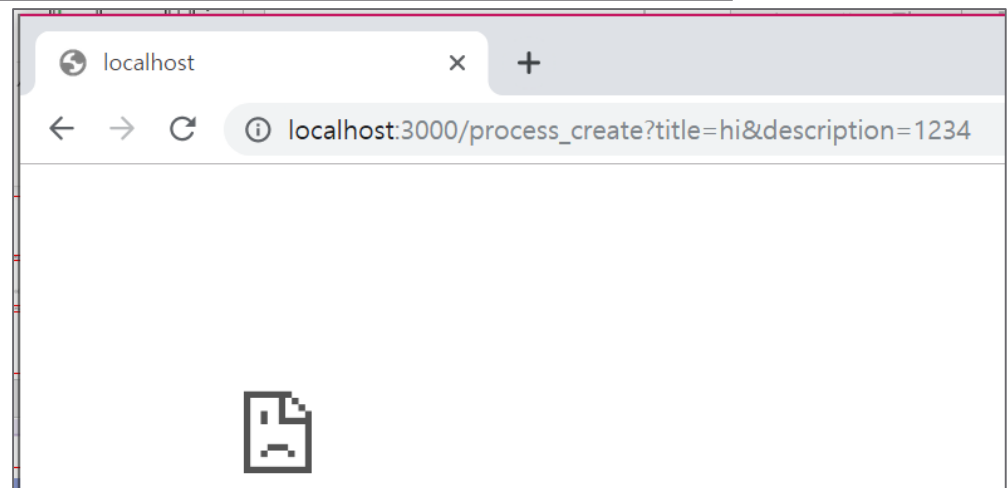
# get방식

- » 클라이언트로가 입력한 query의 이름과 값이 결합되어 스트링 형태로 서버에 전달됩니다.
- » 한번 요청시 전송 데이터 양은 주소값+파라미터로 255자(HTTP/1.1 인 2048자)로 제한됩니다.
- » DB에 추가로 정보를 처리하지않고, 저장된 Data를 단순 요청하는 정도로 사용합니다.
- » URL에 그대로 query의 이름과 값이 같이 연결되어 표현됩니다.



# get방식

```
<form action="http://localhost:3000/process_create" >
  <p><input type="text" name="title"></p>
  <p>
    <textarea name="description"></textarea>
  </p>
  <p>
    <input type="submit">
  </p>
</form>
```





# post방식

- » 클라이언트와 서버 간에 인코딩하여 서버로 전송합니다.
- » 헤더를 통해 요청이 전송되는 방식입니다.
- » 한번 요청시 데이터 양은 제한이 없습니다.
- » DB에 추가로 서버에서 갱신 작업을 할때, 서버에서 정보가 가공되어 응답하는 경우에 사용합니다.
- » POST 방식 : 클라이언트에서 데이터를 인코딩 -> 서버측에서 디코딩 해서 사용합니다.
- » Query는 body 안에 들어가 있어서 보안에 조금 유리함이 있습니다.



# post방식

```
<form action="http://localhost:3000/process_create" method="post">
  <p><input type="text" name="title"></p>
  <p>
    <textarea name="description"></textarea>
  </p>
  <p>
    <input type="submit">
  </p>
</form>
```

