

Node.js

html 랜더링 및 기타 기능 - 김근형 강사

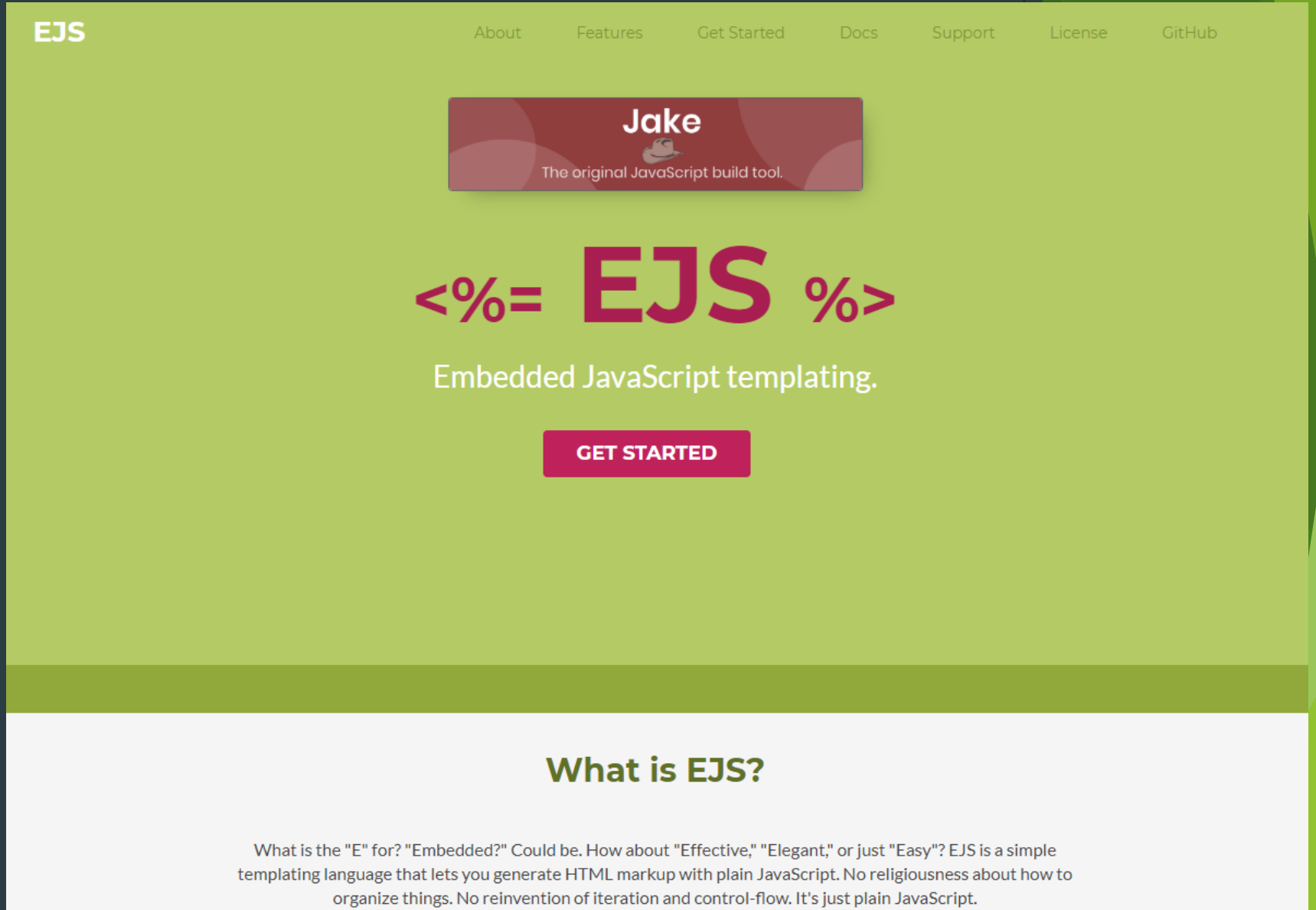
HTML 랜더링

▶ HTML 랜더링

- ▶ **html** 데이터를 문자열로 만들어서 **res.write**나 **res.send**에 담아서 전송하려면 생산성에 문제가 생길 수 있다.
- ▶ **express** 모듈은 외부 파일의 데이터를 읽어와 **html** 코드로 만든 다음 클라이언트에게 전달하는 기능을 제공하는데 이를 랜더링이라 부른다.
- ▶ **express** 에서 랜더링을 위해서는 다른 모듈을 사용해야 하는데 두가지 모듈인 **pug(jade)**와 **ejs**가 존재한다.
- ▶ **pug**는 **html** 코드를 그대로 쓸 수 없으므로 **ejs**를 사용하는 편이 디자이너들과 협업하기에도 좋다.

EJS

► EJS



The screenshot shows the EJS website homepage. At the top, there is a navigation bar with links: About, Features, Get Started, Docs, Support, License, and GitHub. Below the navigation bar is a banner for Jake, described as 'The original JavaScript build tool.' The main content area features the EJS logo, which is the text '<%= EJS %>' in a large, bold, pink font. Below the logo, the text 'Embedded JavaScript templating.' is displayed. A pink button labeled 'GET STARTED' is positioned below the text. The bottom section of the page has a white background and contains the heading 'What is EJS?' followed by a paragraph explaining the acronym 'E' and the purpose of the templating language.

EJS

About Features Get Started Docs Support License GitHub

Jake
The original JavaScript build tool.

<%= EJS %>

Embedded JavaScript templating.

GET STARTED

What is EJS?

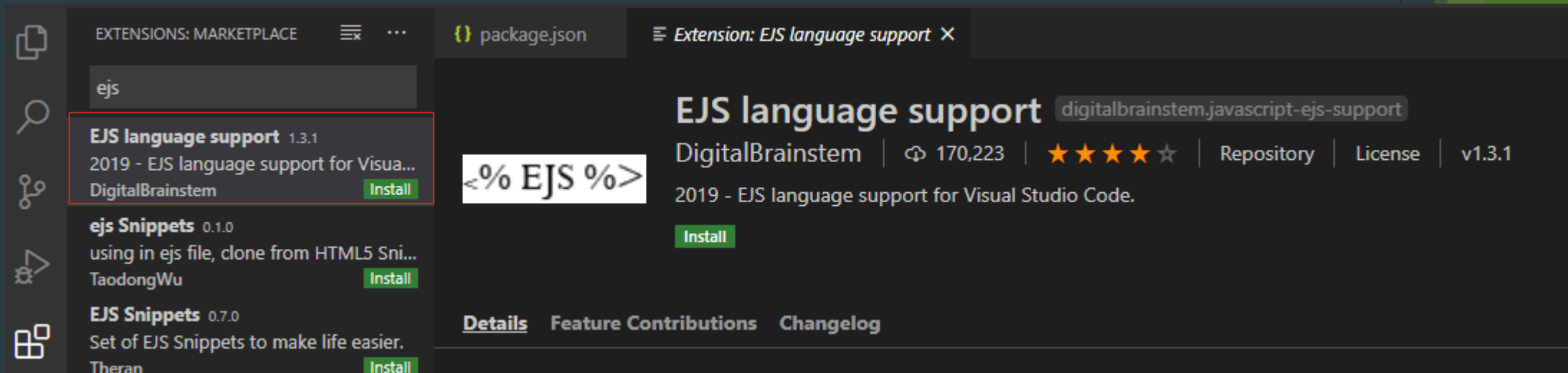
What is the "E" for? "Embedded?" Could be. How about "Effective," "Elegant," or just "Easy"? EJS is a simple templating language that lets you generate HTML markup with plain JavaScript. No religiousness about how to organize things. No reinvention of iteration and control-flow. It's just plain JavaScript.

EJS

- ▶ EJS(Embedded JavaScript)
 - ▶ 자바스크립트가 내장되어 있는 `html` 파일
 - ▶ 자바스크립트를 마치 `html` 태그처럼 삽입할 수 있다.
 - ▶ 페이지를 동적으로 짜기 위해 최적의 환경을 만들어준다.
 - ▶ 또한 서버에서 세팅한 변수를 가져다 쓸 수 있다는 장점이 있다.

EJS

- ▶ vscode package 세팅
 - ▶ EJS language support 설치



EJS

▶ package.json 설정

```
{  
  "name": "EJSProject",  
  "version": "0.0.0",  
  "private": true,  
  "dependencies": {  
    "express": "*",  
    "ejs": "*"   
  }  
}
```

EJS

▶ ejs 스크립트 없이 html 파일 랜더링 만으로 앱 만들기

```
> node_modules
├── router
│   └── JS router1.js
├── views
│   ├── index.html
│   └── test.html
├── JS 01_main1.js
├── {} package.json
└── {} package-lock.json
```

```
<body>
  <h1>index.html</h1>
</body>
```

```
<body>
  <h1>test.html</h1>
</body>
```

```
module.exports = function(app){
  app.get("/", function(req,res){
    res.render("index.html");
  });

  app.get("/test", function(req, res){
    res.render("test.html");
  });
}
```

```
var express = require('express');
var ejs = require("ejs");

var app = express();

app.set("views", __dirname + "/views");
app.set("view engine", "ejs");
app.engine("html", ejs.renderFile);

var router1 = require('./router/router1')(app)

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```

localhost:2000

index.html

localhost:2000/test

test.html

EJS

▶ 정적 파일 사용하기

- ▶ html 문서에는 css, js, image, 동영상, 사운드 등의 파일들을 사용할 때는 정적 파일이 위치하는 폴더를 지정하여 사용할 수 있다.

```
var express = require('express');
var ejs = require("ejs");


var app = express();

app.set("views", __dirname + "/views");
app.set("view engine", "ejs");
app.engine("html", ejs.renderFile);

app.use(express.static("public"));

var router1 = require('./router/router1')(app)

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```



```
> node_modules
▼ public
  ▼ css
    # aaa.css
  > js
  > router
  > views
  JS 01_main1.js
  JS 02_main2.js
  {} package.json
  {} package-lock.json
```


EJS

▶ 정적 파일 사용하기 예제

```
> node_modules
  public
    css
      # aaa.css
    img
      node.png
    js
      JS bbb.js
  router
    JS router1.js
  views
    < index.html
    < red.html
    < test.html
  JS 01_main1.js
  JS 02_main2.js
  {} package.json
  {} package-lock.json
```

```
@charset "utf-8";

.c1 {
  color : ■ red;
}
```

```
function bbb_func(){
  alert("자바 스크립트 코드가 실행되었습니다.");
}
```

```
module.exports = function(app){
  app.get("/", function(req,res){
    res.render("index.html");
  });

  app.get("/test", function(req, res){
    res.render("test.html");
  });

  app.get("/red", function(req, res){
    res.render("red.html");
  });
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="css/aaa.css"/>
  <script src="js/bbb.js"></script>
</head>
<body>
  <h1 class="c1">red.html</h1>
  <button onclick="javascript:bbb_func()">버튼</button>
  
</body>
</html>
```

```
var express = require('express');
var ejs = require('ejs');

var app = express();

app.set("views", __dirname + "/views");
app.set("view engine", "ejs");
app.engine("html", ejs.renderFile);

app.use(express.static("public"));

var router1 = require('./router/router1')(app);

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```

EJS

▶ 정적 파일 사용하기 예제



localhost:2000 내용:

자바 스크립트 코드가 실행되었습니다.

확인

EJS

- ▶ EJB를 활용하여 동적 웹 페이지 만들기
 - ▶ EJS에서 자바스크립트를 사용하기 위해 필요한 문법

```
<%  
    자바스크립트 소스코드 사용 가능  
%>
```

```
<%= 자바스크립트 변수 사용 가능 %>
```

EJS

▶ EJB를 활용하여 동적 웹페이지 만들기 예제 - 1

```
app.get("/", function(req,res){  
  res.render("index.ejs");  
});
```

index.ejs

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Document</title>  
</head>  
<body>  
  <h1>index.ejs</h1>  
</body>  
</html>
```

EJS

▶ EJB를 활용하여 동적 웹 페이지 만들기 예제 - 3

- ▶ 보내고 싶은 데이터가 존재할 경우 객체 형태로 데이터를 만들어 **ejs** 화면에 뿌려줄 수 있다.

```
app.get("/test2", function(req, res){  
  var date = new Date();  
  
  var render_data = {  
    str : "문자열입니다",  
    number : 100,  
    date : date  
  }  
  
  res.render("test2.ejs", render_data);  
});
```

test2.ejs

str : 문자열입니다

number : 100

date : Sun Aug 09 2020 17:36:22 GMT+0900 (GMT+09:00)

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Document</title>  
</head>  
<body>  
  <h1>test2.ejs</h1>  
  <p>str : <%=str%></p>  
  <p>number : <%=number%></p>  
  <p>date : <%=date%></p>  
</body>  
</html>
```

EJS

▶ EJB를 활용하여 동적 웹페이지 만들기 예제 - 1

```
app.get("/test1", function(req, res){  
  res.render("test1.ejs");  
});
```

test1.ejs

date : Sun Aug 09 2020 17:34:24 GMT+0900 (GMT+09:00)

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Document</title>  
</head>  
<body>  
  <h1>test1.ejs</h1>  
  <%  
    var date = new Date();  
  %>  
  <p>date : <%=date%></p>  
</body>  
</html>
```

Express 파라미터 요청

▶ Express 에서 파라미터 요청

- ▶ 파라미터는 클라이언트가 서버에 요청할 때 전달하는 데이터를 의미한다.
- ▶ `express` 에서 파라미터는 `request` 객체를 통해 파라미터 추출이 가능하다.
- ▶ `get` 방식의 경우 `query`라는 객체안에 모두 들어있다.
- ▶ `post` 방식의 경우 `bodyParser` 모듈을 이용해야 파라미터를 추출할 수 있다.

Express 파라미터 요청

- ▶ Express 에서 파라미터 요청 예제 - 1
 - ▶ package.json 설정

```
{  
  "name": "EJSProject",  
  "version": "0.0.0",  
  "private": true,  
  "dependencies": {  
    "express": "*",  
    "ejs": "*",  
    "body-parser": "*"   
  }  
}
```


Express 파라미터 요청

▶ Express 에서 파라미터 요청 예제 - 1

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>get.ejs</h1>
  <a href="parameter?data1=aaa&data2=bbb">파라미터 출력으로 이동</a>
  <form action="parameter" method="get">
    <label>data1 : <input type="text" name="data1"></label><br>
    <label>data2 : <input type="text" name="data2"></label><br>
    <button type="submit">입력</button>
  </form>
  <form action="parameter" method="post">
    <label>data1 : <input type="text" name="data1"></label><br>
    <label>data2 : <input type="text" name="data2"></label><br>
    <button type="submit">입력</button>
  </form>
</body>
</html>
```

get.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>parameter.ejs</h1>
  <p>data1 : <%=data1%></p>
  <p>data2 : <%=data2%></p>
</body>
</html>
```

parameter.ejs

Express 파라미터 요청

- ▶ Express 에서 파라미터 요청 예제 - 1
 - ▶ router 세팅

get.ejs

파라미터 출력으로 이동

data1 :

data2 :

입력

data1 :

data2 :

입력

parameter.ejs

data1 : abcd

data2 : abcd

parameter.ejs

data1 : qwer

data2 : qwer

```
var bodyParser = require("body-parser");
var urlencodedParser = bodyParser.urlencoded({extended:false});
```

```
module.exports = function(app){
  app.get("/", function(req,res){
    res.render("index.ejs");
  });

  app.get("/get", function(req,res){
    res.render("get.ejs");
  });

  app.get("/parameter", function(req, res){
    var render_data = {
      data1 : req.query.data1,
      data2 : req.query.data2
    };
    res.render("parameter.ejs",render_data)
  });
};
```

```
app.post("/parameter", urlencodedParser, function(req, res){
  var render_data = {
    data1 : req.body.data1,
    data2 : req.body.data2
  };
  res.render("parameter.ejs",render_data)
});
```

```
}
```

쿠키(cookie)

- ▶ 쿠키(cookie)
 - ▶ 쿠키란 클라이언트 측에 저장되는 데이터를 의미한다.
 - ▶ 클라이언트가 서버에 요청할 때 쿠키 정보를 전부 전달하게 된다. 이를 통해 서버에서 사용자 컴퓨터에 저장된 쿠키 정보를 사용할 수 있다.
 - ▶ 쿠키는 사용자 컴퓨터에 저장되므로 브라우저를 닫아도 데이터가 유지된다.
 - ▶ `express`에서 쿠키를 관리할 때는 `cookie-parser` 모듈을 사용한다.

쿠키(cookie)

- ▶ 쿠키(cookie) 예제 - 1
 - ▶ package.json 설정

```
{  
  "name": "EJSProject",  
  "version": "0.0.0",  
  "private": true,  
  "dependencies": {  
    "express": "*",  
    "ejs": "*",  
    "body-parser": "*",  
    "cookie-parser": "*"   
  }  
}
```

쿠키(cookie)

- ▶ 쿠키(cookie) 예제 - 1
 - ▶ main 및 router 설정

```
var express = require('express');
var ejs = require("ejs");
var cookieParser = require("cookie-parser");

var app = express();

app.set("views", __dirname + "/views");
app.set("view engine", "ejs");
app.engine("ejs", ejs.renderFile);
app.use(cookieParser());

app.use(express.static("public"));

var router1 = require('./router/router4')(app);

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```

```
var bodyParser = require("body-parser");
var urlencodedParser = bodyParser.urlencoded({extended:false});

module.exports = function(app){
  app.get("/", function(req,res){
    res.render("index.ejs");
  });

  app.get("/cookie", function(req,res){
    res.render("cookie.ejs");
  });

  app.get("/save_cookie", function(req,res){
    var op = {
      maxAge : 365 * 24 * 60 * 60
    }
    // 쿠키저장
    res.cookie("cookie1", "aaaaa", op);
    res.render("save_cookie.ejs");
  });

  app.get("/load_cookie", function(req,res){
    var render_data = {
      cookie1 : req.cookies.cookie1
    }
    // 쿠키저장
    res.render("load_cookie.ejs",render_data);
  });
}
```

쿠키(cookie)

- ▶ 쿠키(cookie) 예제 - 1
 - ▶ cookie, save_cookie, load_cookie 설정

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>cookie.ejs</h1>
  <a href="save_cookie">쿠키저장</a>
  <a href="load_cookie">쿠키로드</a>
</body>
</html>
```

cookie.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>sava_cookie.ejs</h1>
  <h2>쿠키가 저장되었습니다.</h2>
</body>
</html>
```

save_cookie.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>load_cookie.ejs</h1>
  <p> cookie1 : <%=cookie1%></p>
</body>
</html>
```

load_cookie.ejs

쿠키(cookie)

▶ 쿠키(cookie) 예제 실행

cookie.ejs

[쿠키저장](#) [쿠키로드](#)

sava_cookie.ejs

쿠키가 저장되었습니다.

load_cookie.ejs

cookie1 : aaaaa

세션(session)

- ▶ 세션(session)
 - ▶ 서버 메모리에 데이터를 저장하는 방식으로 클라이언트 하나당 하나의 공간이 할당된다.
 - ▶ 브라우저를 닫으면 세션은 삭제된다.
 - ▶ `express` 에서 세션을 관리할 때 `express-session` 모듈을 사용한다.

세션(session)

- ▶ 세션(session) 예제 - 1
 - ▶ package.json 세팅

```
{  
  "name": "EJSProject",  
  "version": "0.0.0",  
  "private": true,  
  "dependencies": {  
    "express": "*",  
    "ejs": "*",  
    "body-parser": "*",  
    "cookie-parser": "*",  
    "express-session": "*"   
  }  
}
```

세션(session)

▶ 세션(session) 예제 - 1

▶ main과 router 세팅

```
var express = require('express');
var ejs = require("ejs");
var cookieParser = require("cookie-parser");
var session = require("express-session");

var app = express();

app.set("views", __dirname + "/views");
app.set("view engine", "ejs");
app.engine("ejs", ejs.renderFile);
app.use(cookieParser());
app.use(session({
  secret : "abcdefg", // 이 데이터를 이용해서 암호화를 하게 된다.
  resave : false, // 세션의 정보를 다시 저장하는지 여부
  saveUninitialized : true // 초기화 값을 저장할지 안할지 여부
}));

app.use(express.static("public"));

var router1 = require('./router/router5')(app);

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```

```
var bodyParser = require("body-parser");
var urlencodedParser = bodyParser.urlencoded({extended:false});

module.exports = function(app){
  app.get("/", function(req,res){
    res.render("index.ejs");
  });

  app.get("/session", function(req,res){
    res.render("session.ejs");
  });

  app.get("/save_session", function(req,res){
    req.session.data1 = 100;
    req.session.data2 = "안녕하세요";

    res.render("save_session.ejs");
  });

  app.get("/load_session", function(req,res){
    var render_data = {
      data1 : req.session.data1,
      data2 : req.session.data2
    }

    res.render("load_session.ejs",render_data);
  });
}
```

세션(session)

▶ 세션(session) 예제 - 1

- ▶ session.ejs, save_session.ejs, load_session.ejs

```
<body>
  <h1>session.ejs</h1>
  <a href="/save_session">세션 저장</a><br>
  <a href="/load_session">세션 로드</a><br>
</body>
```

session.ejs

[세션 저장](#)
[세션 로드](#)

```
<body>
  <h1>sava_session.ejs</h1>
  <h3>세션이 저장되었습니다.</h3>
</body>
```

sava_session.ejs

세션이 저장되었습니다.

```
<body>
  <h1>load_session.ejs</h1>
  <p>data1 : <%=data1%></p>
  <p>data2 : <%=data2%></p>
</body>
```

load_session.ejs

data1 : 100

data2 : 안녕하세요