

# 1. SSR 개요(1)

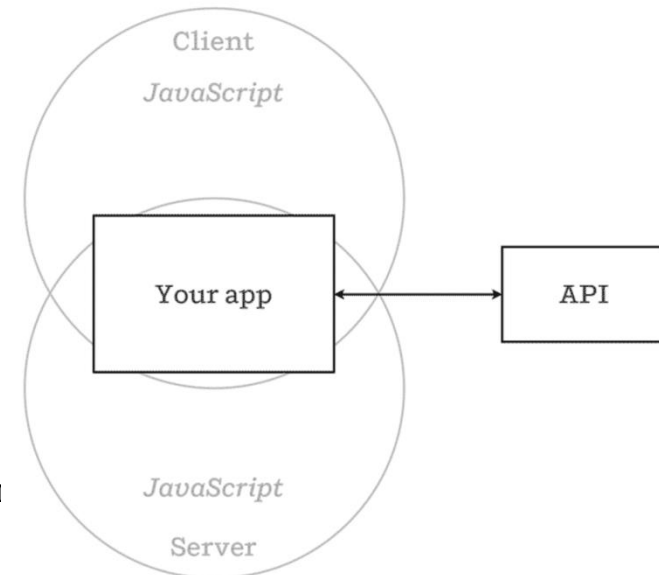


## ■ SPA(Single Page Application)

- JS 코드를 이용해 동적 애플리케이션 기능 구현
- 브라우저가 JS 코드를 내려받아 실행할 때까지 화면이 나타나지 않음
- CSR : Client Side Rendering
- SSR : Server Side Rendering

## ■ Isomorphic JS Application

- 코드가 전체 또는 부분적으로 클라이언트와 서버 애플리케이션
- JS 코드를 서버에서 실행
  - SSR(Server Side Rendering)
  - 브라우저로 전달되기 전에 미리 화면을 구성해줄 수 있음
- PHP, Spring Framework, .NET 등에서도 가능
  - <https://facebook.github.io/react/docs/environments.htm>



# 1. SSR 개요(2)



## ■ SSR(Server Side Rendering)

### ■ 사용 메서드

- `import ReactDOMServer, { renderToString } from 'react-dom/server';`
- `import React from 'react';`
- `const html = renderToString(AppInstance);`

### ■ 장점

- 빠른 로딩 속도, 빠른 렌더링
- SEO(search engine optimization) 기능 지원
- JS 코드 오류가 발생해도 애플리케이션이 중단되지 않음.
- Full URL Navigation
- 더 좋은 보안 접근성

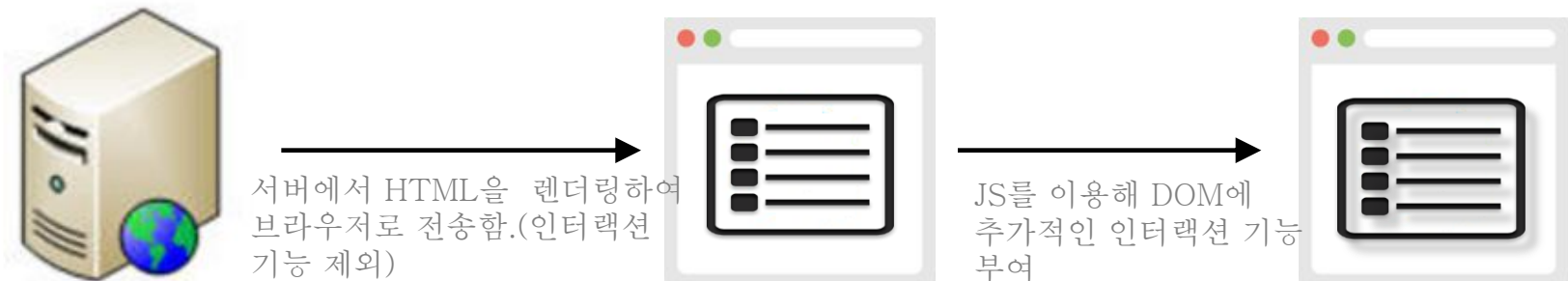
# 1. SSR 개요(3)



## ■ CSR(Client-Side Rendering)



## ■ SSR(Server-Side Rendering)



## 2. Node.js + express 기초(1)



### ■ Node.js

- 서버측 자바스크립트 기술
- React 애플리케이션을 서버에서 실행하기 위해 필요함.

### ■ Express

- Node.js 웹 애플리케이션 서버 프레임워크

### ■ EJS

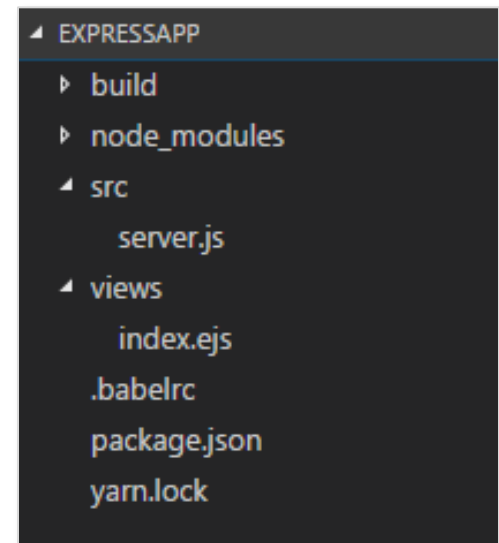
- Express에서 사용할 수 있는 자바스크립트 기반의 템플릿 엔진

## 2. Node.js + express 기초(2)



### ■ 간단한 예제

- 프로젝트 초기화
  - mkdir expressapp
  - cd expressapp
  - yarn init
  - yarn add express ejs
  - yarn add -D babel-cli babel-preset-env nodemon npm-run-all
- 디렉터리 구조 : 오른쪽 그림 참조
- .babelrc 파일 추가



```
{  
  "presets": [ "env" ]  
}
```

## 2. Node.js + express 기초(3)



### ■ src/server.js 작성

```
import express from 'express';
const app = express();

app.set('view engine', 'ejs');
app.use(express.static(__dirname + '/public'));

const port = process.env.PORT || 8080;

app.get('/', (req, res) => {
  res.render('index', {
    title: 'React.js',
    htmldata: `

<h3>Declarative</h3>
      <p>React makes it painless to create interactive UIs. (중략)</p>
    </div>`
  })
})

app.listen(8080, () => {
  console.log(`${port}에서 서버 실행중!`);
})


```

## 2. Node.js + express 기초(4)



### ■ views/index.ejs 작성

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title><%=title %></title>
</head>
<body>
  <h1><%=title %></h1>
  <div>
    <%=htmldata %>
  </div>
</body>
</html>
```

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

## 2. Node.js + express 기초(5)



### ■ package.json 변경

```
{
  "name": "expressapp",
  "version": "1.0.0",
  "main": "server.js",
  "license": "MIT",
  "scripts": {
    "build": "babel src -d build",
    "runssr": "node build/server.js",
    "server": "npm-run-all --serial build runssr",
    "build:watch": "babel src --out-dir build --watch",
    "run:watch": "nodemon build/server.js",
    "server:watch": "npm run build && npm-run-all --parallel build:watch run:watch"
  },
  "dependencies": {
    "ejs": "^2.5.7",
    "express": "^4.16.2"
  },
  "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-preset-env": "^1.6.1",
    "nodemon": "^1.14.11",
    "npm-run-all": "^4.1.2"
  }
}
```

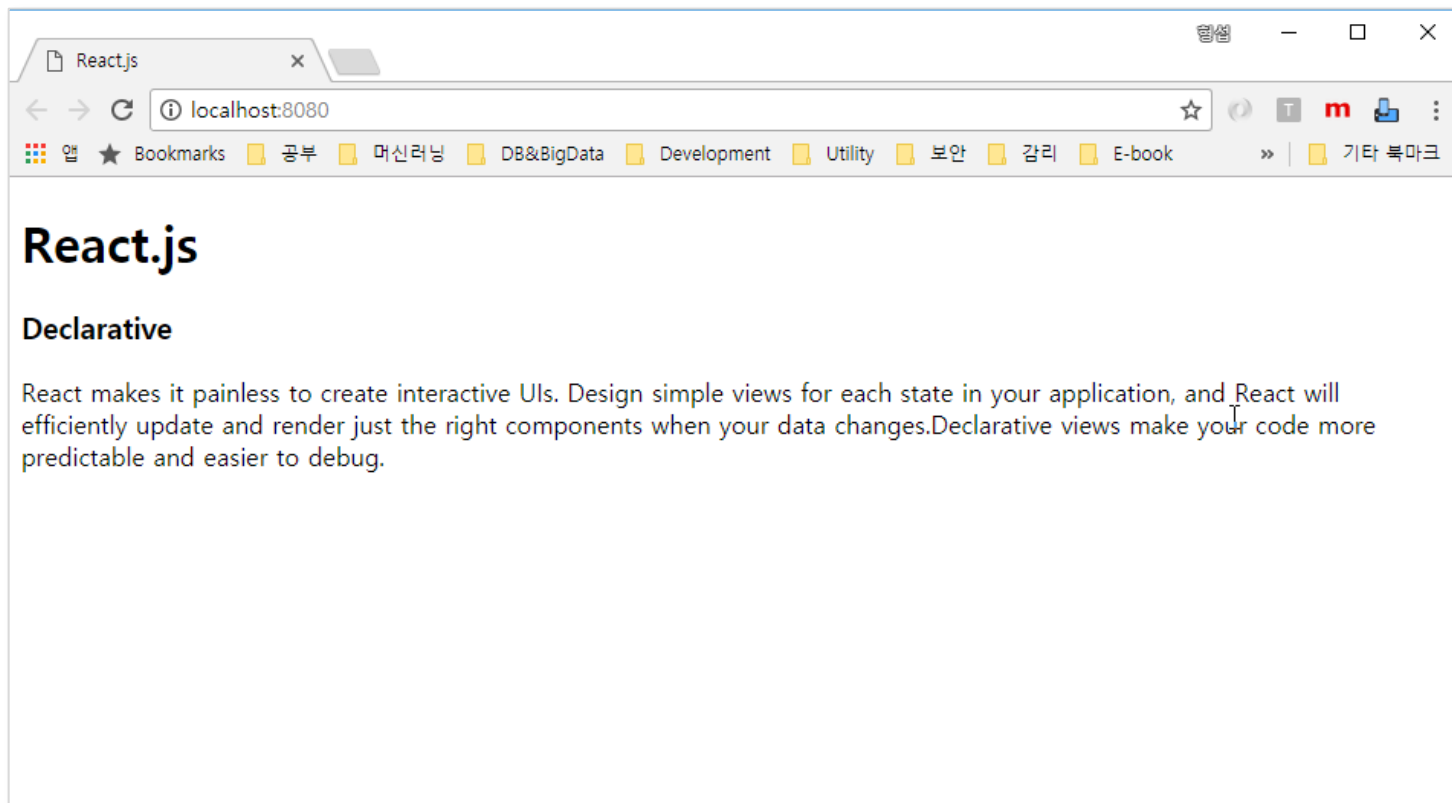


## 2. Node.js + express 기초(6)



### ■ 실행 결과

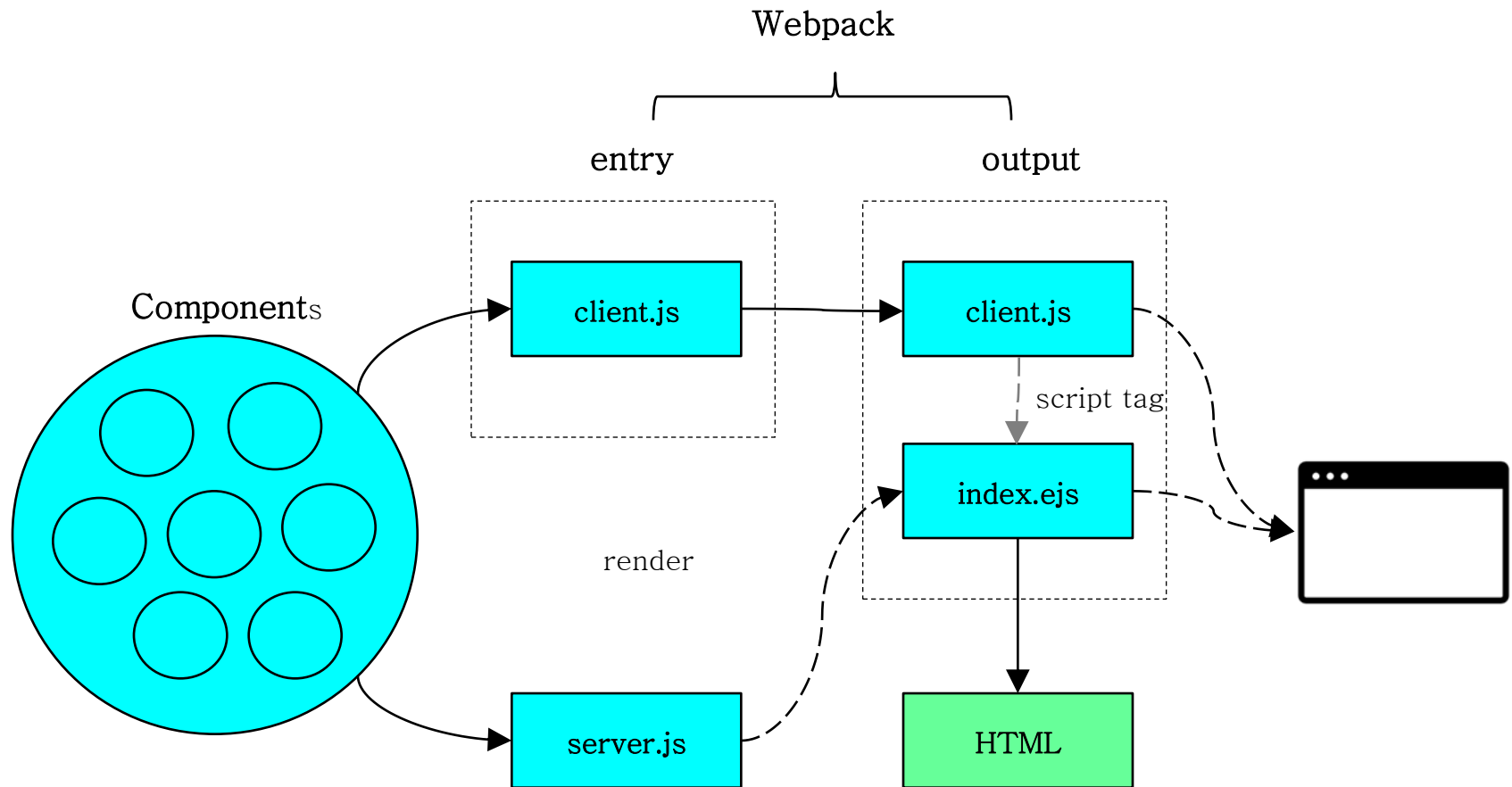
- yarn server:watch 또는 yarn server



### 3. React SSR 기초(1)



#### ■ 전체 개요도



### 3. React SSR 기초(2)



#### ■ expressapp 예제에 React 컴포넌트 렌더링 기능 추가

##### ■ 패키지 추가 참조

- yarn add react react-dom
- yarn add -D babel-loader babel-preset-react webpack

##### ■ client/App.js 추가

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      title : "Hello React SSR",
      members: [
        { id:1, name:"홍길동" },
        { id:2, name:"이몽룡" },
        { id:3, name:"성춘향" }
      ]
    }
  }
}
```

(다음 페이지에 이어짐)

### 3. React SSR 기초(3)



#### ■ client/App.js 추가(이어서)

```
render() {  
  let list = this.state.members.map((member)=>{  
    return <li key={member.id}>{member.name}</li>  
  })  
  
  return (  
    <div>  
      <h2>{this.state.title}</h2>  
      <ul>  
        {list}  
      </ul>  
    </div>  
  );  
}  
}  
  
export default App;
```

### 3. React SSR 기초(4)



#### ■ src/server.js 변경

```
import express from 'express';
import App from '../client/App';
import ReactDOMServer, { renderToString } from 'react-dom/server';
import React from 'react';

const app = express();

app.set('view engine', 'ejs');
app.use(express.static(__dirname + '/public'));

const port = process.env.PORT || 8080;

app.get('/', (req, res) => {
  let html = renderToString(<App title="React SSR Test!!" />)
  res.render('index', {
    title: 'React.js',
    htmldata: html
  })
})

app.listen(8080, () => {
  console.log(`${port}에서 서버 실행중!!!`);
})
```

### 3. React SSR 기초(5)



#### ■ webpack.ssr.config.js 작성

```
module.exports = {
  entry: {
    server: __dirname + '/src/server.js',
  },
  externals: ['express'],
  output: {
    path: __dirname + '/build',
    filename: '[name].js',
    libraryTarget: "commonjs2"
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /(node_modules|bower_components)/,
        use: {
          loader: 'babel-loader',
          options: { presets: ['env', 'react'] }
        }
      }
    ]
  },
  "target": "node"
};
```

### 3. React SSR 기초(6)



#### ■ package.json 변경

```
{
  .....
  "scripts": {
    "build": "webpack --config ./webpack.ssr.config.js",
    "runssr": "node build/server.js",
    "server": "npm-run-all --serial build runssr",
    "build:watch": "webpack --config ./webpack.ssr.config.js --watch",
    "run:watch": "nodemon build/server.js",
    "server:watch": "npm run build && npm-run-all --parallel build:watch run:watch"
  },
  "dependencies": {
    "ejs": "^2.5.7",
    "express": "^4.16.2",
    "react": "^16.2.0",
    "react-dom": "^16.2.0"
  },
  "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-preset-env": "^1.6.1",
    "babel-loader": "^7.1.2",
    "babel-preset-react": "^6.24.1",
    "nodemon": "^1.14.11",
    "npm-run-all": "^4.1.2",
    "webpack": "^3.10.0"
  }
}
```

### 3. React SSR 기초(7)



#### ■ 실행 결과

- yarn server:watch 또는 npm run server:watch

A screenshot of a web browser window. The address bar shows 'view-source:localhost:8080'. The browser's bookmark bar is visible with various categories like '공부', '머신러닝', 'DB&BigData', etc. The main content area displays the source code of an HTML document. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>React.js</title>
8 </head>
9 <body>
10   <h1>React.js</h1>
11   <div>
12     <div data-reactroot=""><h2>Hello React SSR</h2><ul><li>홍길동1</li>
13 <li>홍길동2</li><li>홍길동3</li></ul></div>
14 </div>
15 </body>
16 </html>
```



## 4. 연락처 앱(1)



### ■ 연락처 앱에 React SSR 기능 추가

- 핵심은 URL 경로에 따라 다른 Route 컴포넌트를 포함하여 renderToString 수행
- StaticRouter를 이용하면서 Provider를 통해 Store 객체를 전달해야 하

```
//Express 기반으로 실행되는 server.js
app.get('*', (req, res) => {
  console.log(html)
  let context = {};

  const html = renderToString(
    <Provider store={ContactStore}>
      <StaticRouter location={req.url} context={context} >
        <App/>
      </StaticRouter>
    </Provider>
  )

  return res.render('index', { html:html })
})
```

## 4. 연락처 앱(2)



### ■ 예제 작성

- 연락처 앱에 기능 추가
- 패키지 참조
  - yarn add ejs express
  - yarn add -D babel-cli cross-env nodemon npm-run-all
- src/server.js 추가

```
import express from 'express'
import fs from 'fs';
import path from 'path'
import React from 'react'
import ReactDOMServer, { renderToString } from 'react-dom/server'
import { StaticRouter } from 'react-router-dom'
import App from './src/components/App'
import routes from './src/routes'
import ContactStore from './src/store/ContactStore';
import { Provider } from 'react-redux';
```

```
const app = express()
const viewPath = process.env.DEVELOPMENT ? 'view' : 'build'
```

(다음 페이지에 이어짐)

## 4. 연락처 앱(3)



### ■ src/server.js 추가(이어서)

```
app.set('view engine', 'ejs')
app.set('views', path.join(__dirname, viewPath))
app.use(express.static(path.join(__dirname, 'build')))

app.get('*', (req, res) => {
  let context = {};
  const html = renderToString(
    <Provider store={ContactStore}>
      <StaticRouter location={req.url} context={context} >
        <App/>
      </StaticRouter>
    </Provider>
  )
  if(context.url) {
    res.writeHead(301, {Location: context.url})
    res.end()
  }
  return res.render('index', { html:html })
})

const port = process.env.PORT || 3000
app.listen(port, err => {
  if (err) return console.error(err)
  console.log(`Server listening at http://localhost:${port}`)
})
```

## 4. 연락처 앱(4)



### ■ scripts/buildssr.js 추가

```
const path = require('path');
const fs = require('fs');
const htmlPath = path.resolve('build/index.html');
const ejsPath = path.resolve('build/index.ejs');

// index.html 파일을 읽어서...EJS 표현식을 추가한다.
let html = fs.readFileSync(htmlPath, 'utf8');
let ejs = html.replace('<div id="root">', '<div id="root"><%- html %>');

// EJS 표현식을 포함한 문자열을 다시 build/index.ejs 파일에 기록한다.
fs.writeFileSync(ejsPath, ejs, 'utf8');

// index.html은 삭제한다.
fs.unlinkSync(htmlPath);
console.log("index.html을 index.ejs로 변환 완료!!WrWn");
```

### ■ .babelrc 추가

```
{
  "presets": [ "env", "react" ]
}
```

## 4. 연락처 앱(5)



### ■ package.json 변경

```
{
  "name": "contactsapp",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    .....
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build && node scripts/buildssr.js",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject",
    "server": "nodemon server.js --watch server.js --watch src --exec babel-node",
    "build:server": "npm-run-all --serial build server"
  },
  "devDependencies": {
    .....
  }
}
```

## 4. 연락처 앱(6)



### ■ 실행

- yarn build:server
- 실행 결과 확인 : 브라우저에서 페이지 소스 보기

```
1 <!DOCTYPE html><html lang="en"><head><meta charset="utf-8"><meta name="viewport"
content="width=device-width,initial-scale=1,shrink-to-fit=no"><meta name="theme-
color" content="#000000"><link rel="manifest" href="/manifest.json"><link
rel="shortcut icon" href="/favicon.ico"><title>React App</title><link
href="/static/css/main.94a15a82.css" rel="stylesheet"></head><body><noscript>You
need to enable JavaScript to run this app.</noscript><div id="root"><div
class="container" data-reactroot=""><div><div class="well"><div class="col-xs-1">
</div><div class="title col-xs-10">:: 연락처 앱</div><div class="col-xs-1"><a
class="btn btn-warning btn-circle" href="/add"><span class="glyphicon glyphicon-
plus"></span></a></div><div class="clearfix"></div></div><div class="panel panel-
default panel-borderless"><div class="panel-body"><div class="row"><div
class="col"><div class="input-group"><input type="text" id="name" class="form-
control" name="name" placeholder="Type name and Enter!" value=""><span
class="btn btn-primary input-group-addon">검<!-- --> <!-- -->색</span></div>
</div></div></div><div><ul class="list-group" id="contact-list"></ul></div>
</div></div></div><div id="modal-area"></div><script type="text/javascript"
src="/static/js/main.c6de31ee.js"></script></body></html>
```

## 5. Router Context 추가(1)



### ■ SSR 도중 Router로 임의의 Route Context를 추가하려면?

- context는 Router에 주입하는 임의의 값

```
<StaticRouter location={req.url} context={context} >  
  <App/>  
</StaticRouter>
```

### ■ 예제 분석 : booklistapp

- route 정보

```
const loadData = (match) => {  
  return fetch('http://localhost:3000/books.json')  
    .then(res => res.json())  
}  
const routes = [  
  { path: '/', exact: true, component: Home },  
  { path: '/book', component: Book,  
    routes: [  
      { path: '/book', exact: true, component: BookAll, loadData:loadData },  
      { path: '/book/:slug', component: BookSingle, loadData:loadData }  
    ]  
  },  
  .....  
]
```

## 5. Router Context 추가(2)



### ■ server.js 예시

```
app.get('*', (req, res) => {  
  //URL 경로에 매칭되는 Route 컴포넌트만을 뽑아냄. 리턴값 Route Array  
  const branch = matchRoutes(routes, req.url)  
  const promises = []  
  //Route 컴포넌트 갯수만큼 반복하여 route에 전달된 loadData 함수를 실행하고  
  // 리턴받은 Promise 객체를 promises 배열에 저장함.  
  branch.forEach( ({route, match}) => {  
    if (route.loadData)  
      promises.push(route.loadData(match))  
  })  
  //promises 배열 내의 모든 Promise가 성공적으로 수행되면 한꺼번에 결과를 받음  
  //수신한 배열(data)를 차곡차곡 모아서 context 객체 생성하고 이것을 StaticRouter의 context로 전달함  
  Promise.all(promises).then(data => {  
    const context = data.reduce( (context, data) => {  
      return Object.assign(context, data)  
    }, {})  
    const html = renderToString(  
      <StaticRouter location={req.url} context={context} >  
        <App/>  
      </StaticRouter>  
    )  
    if(context.url) {  
      res.writeHead(301, {Location: context.url})  
      res.end()  
    }  
    return res.render('index', {html})  
  })  
})
```



## 5. Router Context 추가(3)



- 전달한 context를 컴포넌트에서 로딩하는 방법은?

```
class BookAll extends Component {  
  constructor(props) {  
    super(props)  
    this.state = this.props.staticContext || { books: [] }  
  }  
  .....  
}  
  
export default BookAll
```

- 전체 코드는 booklistapp 코드 참조

## 6. Spring + SSR(1)



### ■ Java Spring Framework를 이용한 SSR 기법

- Java로 Javascript 코드를 실행할 수 있는 기능을 이용함.
- React가 만들어내는 요소를 서버에서 생성할 수 있어야 함.
- Nashorn
  - JDK8 이상에서만 제공 (JDK8u66 이상)
  - JSR-223 Scripting API(JSR : Java Specification Request)
    - "스크립트 언어가 Java 플랫폼 내부의 정보에 접근하는 방법을 명시하고, Java의 서버측 어플리케이션에서 스크립트 언어가 쓰일 수 있는 방법을 제공하는 것"
  - Spring Framework 4.2 + Spring Boot 1.3 부터 Nashorn 지원 라이브러리 탑재
    - ScriptTemplateConfigurer
    - ScriptTemplateView : EJS, HandleBars, Jade 등 지원
- J2V8
  - 상대적으로 빠른 실행 속도
  - 운영체제마다 다른 런타임 제공
  - JDK6부터 지원

## 6. Spring + SSR(2)



### ■ Nashorn 간단 샘플

#### ■ hello.js

```
function render(name) {  
    return "<h1>Hello " + name + "</h1>";  
}
```

#### ■ Util.java

```
public class Util {  
    private ThreadLocal<NashornScriptEngine> engineHolder =  
        new ThreadLocal<NashornScriptEngine>() {  
            @Override  
            protected NashornScriptEngine initialValue() {  
                NashornScriptEngine nashornScriptEngine =  
                    (NashornScriptEngine) new ScriptEngineManager().getEngineByName("nashorn");  
                try {  
                    nashornScriptEngine.eval(read("js/hello.js"));  
                } catch (ScriptException e) {  
                    throw new RuntimeException(e);  
                }  
                return nashornScriptEngine;  
            }  
        };  
};
```

(다음 페이지에 이어서)

## 6. Spring + SSR(3)



### ■ (이어서)

#### ■ Util.java

```
public String renderTest() {
    try {
        Object html = engineHolder.get().invokeFunction("render", "홍길동");
        return String.valueOf(html);
    }
    catch (Exception e) {
        throw new IllegalStateException("failed to render react component", e);
    }
}

private Reader read(String path) {
    InputStream in = getClass().getClassLoader().getResourceAsStream(path);
    return new InputStreamReader(in);
}
}
```

#### ■ 기능 실행

```
public static void main(String[] args) {
    Util util = new Util();
    String html = util.renderTest();
    System.out.println(html);
}
```

## 6. Spring + SSR(4)



### ■ J2V8

- nashorn 보다 빠른 실행 속도
- ES6의 일부 기능 제공
  - hello.js : 예) Template String, Arrow Function

```
var render2 = (name) => {  
    return `

# 

}
```

### ■ 샘플

- Util.java

```
public class Util {  
    public String getJSFileString(String path) throws IOException {  
        InputStream in = getClass().getClassLoader().getResourceAsStream(path);  
        Scanner scanner = new Scanner(in);  
        scanner.useDelimiter(System.getProperty("line.separator"));  
        StringBuilder sb = new StringBuilder();  
        while (scanner.hasNext()) {  
            sb.append(scanner.next() + "Wn");  
        }  
        scanner.close();  
        return sb.toString();  
    }  
}
```

```
<dependency>  
    <groupId>com.eclipsesource.j2v8</groupId>  
    <artifactId>j2v8_win32_x86_64</artifactId>  
    <version>4.6.0</version>  
</dependency>
```

## 6. Spring + SSR(5)



### ■ (이어서)

#### ■ 기능 실행

```
public static void main(String[] args) throws IOException {  
    V8 v8 = V8.createV8Runtime();  
    Util util = new Util();  
  
    String js = util.getJSFileString("js/hello.js");  
    v8.executeVoidScript(js);  
    V8Array parameters = new V8Array(v8).push("홍길동");  
    String result = v8.executeStringFunction("render2", parameters);  
    System.out.println(result);  
}
```

## 6. Spring + SSR(6)



### ■ Spring + Nashorn

- ScriptTemplateConfigurer
- ScriptTemplateViewResolver

```
@Configuration
public class ViewConfig {
    .....

    @Bean
    public ViewResolver reactViewResolver() {
        ScriptTemplateViewResolver viewResolver = new ScriptTemplateViewResolver();
        viewResolver.setPrefix("templates/");
        viewResolver.setSuffix(".ejs");
        return viewResolver;
    }

    @Bean
    public ScriptTemplateConfigurer reactConfigurer() {
        ScriptTemplateConfigurer configurer = new ScriptTemplateConfigurer();
        configurer.setEngineName("nashorn");
        configurer.setScripts(scripts);
        configurer.setRenderFunction("render");
        configurer.setSharedEngine(false);
        return configurer;
    }
}
```

## 6. Spring + SSR(7)



### ■ Controller

```
@Controller
public class ReactController {
    .....
    @RequestMapping(value = "/app/**")
    public String index(Model model, HttpServletRequest request) {
        model.addAttribute("requestPath", request.getRequestURI());
        model.addAttribute("personList", PERSON_LIST);
        return "react";
    }
    .....
}
```

### ■ react.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    .....
</head>
<body>
    <div id="root"><%= ReactApp.render(requestPath, personList) %></div>
    <script type="text/javascript" src="/react-app/dist/main.min.js"></script>
</body>
</html>
```



## 6. Spring + SSR(8)



### ■ SSR을 위해 webpack.config.js에서...

- webpack은 내부 함수를 모두 캡슐화함.
  - 외부에서 접근할 객체명을 지정해주어야 함.
  - 외부로 공개할 함수나 변수는 export!!

```
export function render( ... ) {  
  //html을 생성하기 위해 renderToString!! → 생성된 html을 리턴함  
  .....  
  return html;  
}
```

```
module.exports = {  
  entry: {  
    main: path.join(__dirname, 'app/main.js'),  
    server: path.join(__dirname, 'app/main-server.js')  
  },  
  output: {  
    path: path.join(__dirname, '/dist'),  
    filename: '[name].min.js',  
    publicPath: '/',  
    library: 'ReactApp'  
  },  
  .....  
}
```

## 6. Spring + SSR(9)



### ■ nashorn + Spring Sample

- <https://github.com/arrwhidev/nashorn-webpack-react-redux-boilerplate>

## 7. Next.js(1)



### ■ Next.js란?

- Next.js is a minimalistic framework for server-rendered React applications.
  - 기존의 SSR은 복잡하다!! -> Next.js 는 간편함
- <https://learnnextjs.com/>
- <https://github.com/zeit/next.js/>

### ■ 간단한 사용

- `mkdir nextapp`
- `cd nextapp`
- `yarn init`
- `yarn add react react-dom next`
- `pages` 디렉토리에 컴포넌트파일 생성!!

## 7. Next.js(2)



### ■ pages/index.js

```
import React, { Component } from 'react';

class Index extends Component {
  render() {
    console.log("## Render Index Component")
    var dt = new Date();
    return (
      <div>
        <a href="/hello">hello 페이지로</a>
        <h1>Index 페이지 : { dt.getTime() }</h1>
      </div>
    )
  }
}

export default Index;
```

## 7. Next.js(3)



### ■ pages/Hello.js

```
import React, { Component } from 'react';

class Hello extends Component {
  render() {
    console.log("## Render Hello Component")
    return (
      <div>
        <div>Hello!!</div>
        <a href="/">Home으로</a>
      </div>
    );
  }
}

export default Hello;
```

## 7. Next.js(4)



### ■ package.json 변경

#### ■ 시작 script 추가

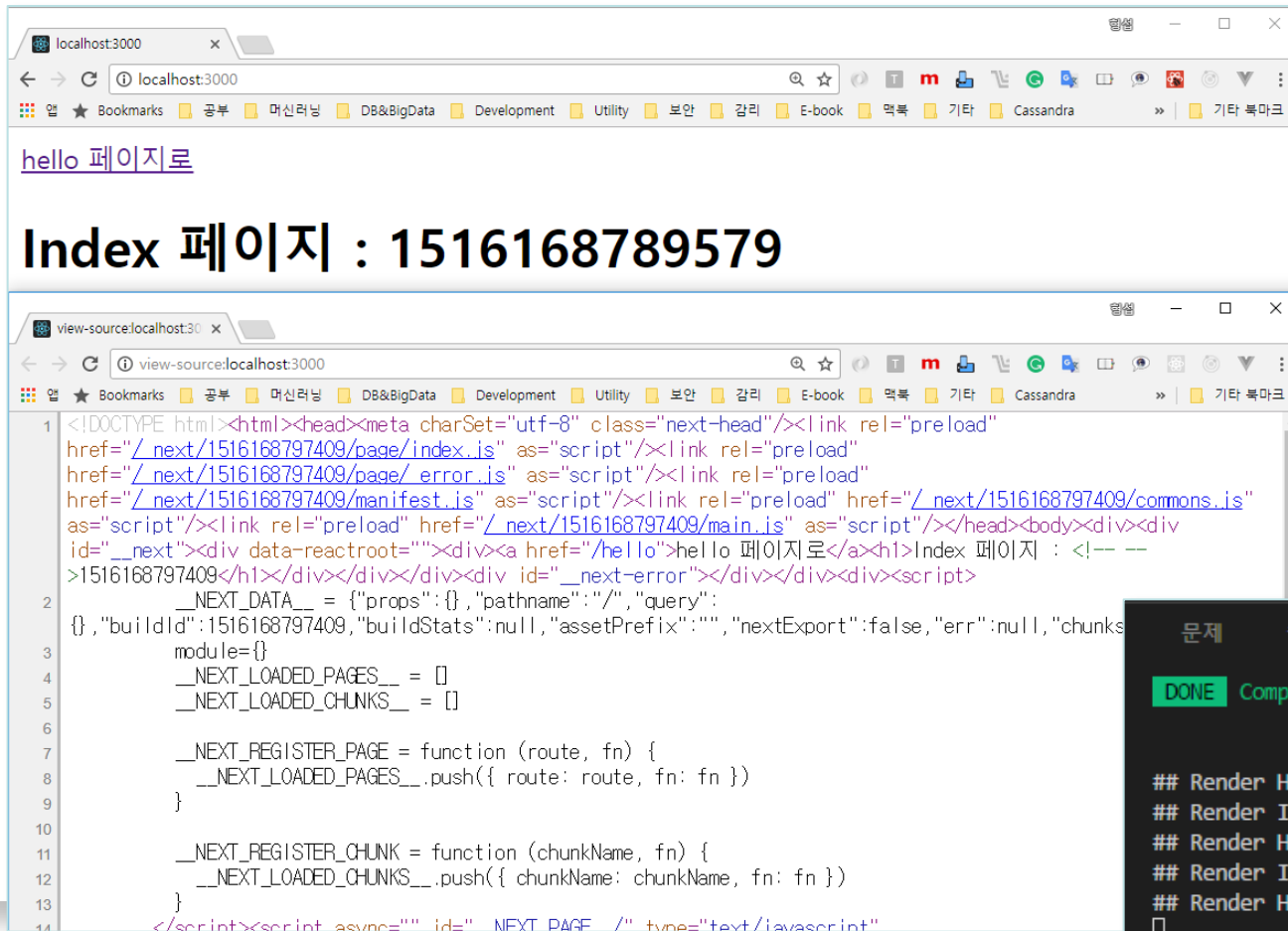
```
{  
  "name": "nextapp",  
  "version": "1.0.0",  
  "main": "index.js",  
  "license": "MIT",  
  "scripts": {  
    "dev": "next",  
    "build": "next build",  
    "start": "next start"  
  },  
  "dependencies": {  
    "next": "^4.2.3",  
    "react": "^16.2.0",  
    "react-dom": "^16.2.0"  
  }  
}
```

# 7. Next.js(5)



## ■ 실행

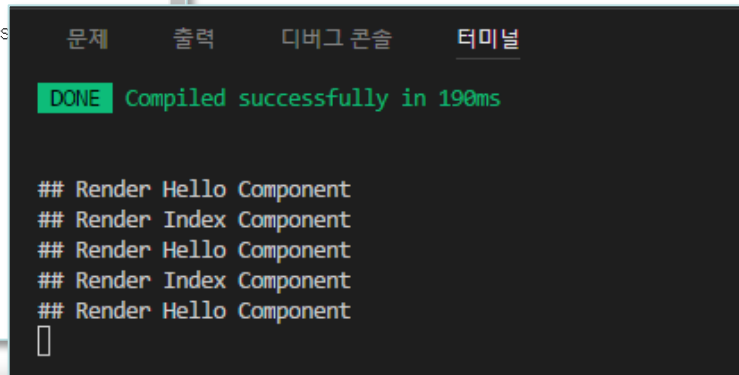
- yarn dev



```
<!DOCTYPE html><html><head><meta charSet="utf-8" class="next-head"/><link rel="preload"
href="/_next/1516168797409/page/index.js" as="script"/><link rel="preload"
href="/_next/1516168797409/page/_error.js" as="script"/><link rel="preload"
href="/_next/1516168797409/manifest.js" as="script"/><link rel="preload" href="/_next/1516168797409/commons.js"
as="script"/><link rel="preload" href="/_next/1516168797409/main.js" as="script"/></head><body><div><div
id="__next"><div data-reactroot=""><div><a href="/hello">hello 페이지로</a><h1>Index 페이지 : <!-- --
>1516168797409</h1></div></div></div></div></div><script>
  __NEXT_DATA__ = {"props": {}, "pathname": "/", "query":
  {}, "buildId": "1516168797409", "buildStats": null, "assetPrefix": "", "nextExport": false, "err": null, "chunks
  module= {}
  __NEXT_LOADED_PAGES__ = []
  __NEXT_LOADED_CHUNKS__ = []

  __NEXT_REGISTER_PAGE = function (route, fn) {
    __NEXT_LOADED_PAGES__.push({ route: route, fn: fn })
  }

  __NEXT_REGISTER_CHUNK = function (chunkName, fn) {
    __NEXT_LOADED_CHUNKS__.push({ chunkName: chunkName, fn: fn })
  }
</script><script async="" id="__NEXT_PAGE" /></script>
```



```
문제    출력    디버그 콘솔    터미널

DONE Compiled successfully in 190ms

## Render Hello Component
## Render Index Component
## Render Hello Component
## Render Index Component
## Render Hello Component
[]
```