Ch.2 자바스크립트

▼ 🌟🌟🌟렌더링

렌더링이란 HTML, CSS, JS로 작성된 문서를 해석해서 브라우저에 시각적으로 출력하는 것을 말한다.

때로는 서버에서 데이터를 HTML로 변환해서 브라우저에게 전달하는 과정을(SSR) 가리키기도 한다.

▼ 🌟🌟🌟 CSR vs SSR

클라이언트 사이드 렌더링 (Client-Side Rendering, CSR)와 서버 사이드 렌더링 (Server-Side Rendering, SSR)은 웹 애플리케이션의 HTML 콘텐츠를 렌더링하는 두 가지 주요 방식입니다.

클라이언트 사이드 렌더링 (CSR)

- **렌더링 위치**: 사용자 브라우저에서 렌더링
- **작동 방식**: 서버는 최소한의 HTML과 자바스크립트 파일을 클라이언트에게 전달하고, 브라우저가 이를 받아서 실행하여 화면을 구성
- **CSR**은 사용자의 인터랙션이 많은 애플리케이션에 적합하며, 풍부한 사용자 경험을 제공하지만 초기 로딩 시간이 길고 SEO에 불리합니다.

장점

1. **빠른 사용자 인터페이스**: 최초 로드 후에는 동적인 페이지 전환이 빠르며, 애플리케이션의 반응 속도가 향상됨.
2. **부하 분산**: 서버의 부담이 줄어들고, 많은 사용자 트래픽을 감당할 수 있음.
3. **풍부한 사용자 경험**: 복잡한 UI와 동적인 인터페이스를 구현하는 데 유리함.

단점

1. **초기 로딩 시간**: 초기 페이지 로딩 시 모든 자바스크립트 파일을 다운로드하고 실행해야 하므로 시간이 더 걸릴 수 있습니다.
2. **SEO 최적화 어려움**: 검색 엔진 봇이 자바스크립트를 제대로 실행, 처리하지 못할 경우, 페이지의 콘텐츠를 크롤링하기 어려워, SEO에 불리할 수 있습니다.
3. **자바스크립트 의존성**: 브라우저가 자바스크립트를 지원하고 실행할 수 있어야 합니다. 자바스크립트가 비활성화된 환경에서는 웹사이트가 제대로 동작하지 않을 수 있습니다.

서버 사이드 렌더링 (SSR)

- **렌더링 위치**: 서버에서 렌더링
- **작동 방식**: 서버가 요청에 따라 완전한 HTML 페이지를 생성하여 클라이언트에 전달
- **SSR**은 초기 로딩 속도가 빠르고 SEO에 유리하지만, 서버 부하가 크고 페이지 전환이 느릴 수 있습니다.

장점

1. **빠른 초기 로딩**: 초기 페이지 로딩 속도가 빠르며, 완전한 HTML을 받아서 바로 렌더링 가능.
2. **SEO 최적화**: 검색 엔진이 HTML 콘텐츠를 쉽게 크롤링할 수 있어 SEO에 유리함.
3. **저사양 디바이스 호환성**: 클라이언트의 자바스크립트 처리 능력에 의존하지 않아 다양한 디바이스에서 잘 작동함.

단점

1. **서버 부하 증가**: 모든 요청에 대해 서버가 HTML을 렌더링해야 하므로 서버 부하가 증가할 수 있음.
2. **페이지 전환 느림**: 각 페이지 전환 시 서버로부터 새로운 HTML을 받아야 하므로 페이지 전환 속도가 느려짐.
3. **복잡한 상태 관리**: 클라이언트와 서버 간의 상태 관리가 복잡해질 수 있음.

프론트엔드는 클라이언트 사이드 렌더링(CSR) 방식과 서버 사이드 렌더링(SSR) 방식 모두 사용할 수 있습니다. 어떤 방식이 적합한지는 프로젝트의 요구 사항과 목표에 따라 달라집니다.

프론트엔드에서 주로 CSR을 사용하는 이유

1. **동적 애플리케이션**: CSR은 동적이고 복잡한 사용자 인터페이스를 구현하는 데 매우 적합합니다. 자바스크립트를 사용하여 페이지 내에서 동적으로 콘텐츠를 변경할 수 있습니다.
2. **빠른 사용자 경험**: 첫 로딩 이후에 페이지 전환이 매우 빠릅니다. 페이지 전환 시 서버에서 전체 HTML 페이지를 다시 가져오는 대신, 필요한 데이터만 가져와서 클라이언트 측에서 렌더링합니다.
3. **프론트엔드 프레임워크**: React, Vue.js, Angular와 같은 현대적인 프론트엔드 프레임워크들은 대부분 CSR을 기본으로 설계되어 있습니다. 이들 프레임워크는 상태 관리, 라우팅, 컴포넌트 기반 구조 등 CSR 방식에 최적화되어 있습니다.
4. **부하 분산**: 서버의 부하를 줄여주고, 많은 사용자를 동시에 처리할 수 있는 능력이 향상됩니다.

예시

React 애플리케이션을 예로 들면, ReactDOM.render() 함수를 사용하여 클라이언트 측에서 컴포넌트를 렌더링합니다. 이 경우, 서버는 최소한의 HTML을 전달하고, 나머지 UI는 클라이언트 측에서 렌더링됩니다.

```
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

▼ 🌟Ajax

Asynchronous JavaScript and XML

자바스크립트를 이용해 서버와 브라우저가 비동기적으로 데이터를 교환할 수 있는 통신 기능.

AJAX의 XMLHttpRequest의 등장으로 페이지를 새로 고침하지 않고, 즉 웹페이지에서 변경할 필요가 없는 부분은 다시 렌더링 하지 않고, 서버로부터 필요한 데이터만 전송받아 변경해야 하는 부분만 한정적으로 렌더링하는 방식이 가능해졌습니다.

이로써 웹 브라우저에서도 데스크톱 애플리케이션과 유사한 빠른 성능과 부드러운 화면 전환이 가능해졌습니다.

▼ XMLHttpRequest

1. **XMLHttpRequest 객체**: XMLHttpRequest는 브라우저 내장 객체로, JavaScript에서 서버와 비동기적으로 데이터를 주고받을 수 있게 합니다. 이 객체를 통해 서버에 요청을 보내고, 응답을 받아서 처리할 수 있습니다.
2. **이름의 의미**: XMLHttpRequest라는 이름은 객체가 처음 설계되었을 때, 주로 XML 데이터를 HTTP를 통해 비동기적으로 전송하는 데 사용되었기 때문에 붙여졌습니다. 그러나, AJAX는 XML뿐만 아니라 JSON, HTML, 일반 텍스트 등의 다양한 데이터 형식을 전송할 수 있습니다.

아래는 XMLHttpRequest를 사용하여 간단한 AJAX 요청을 보내는 예제입니다:

```
// XMLHttpRequest 객체 생성
var xhr = new XMLHttpRequest();

// 요청의 종류와 URL을 설정 (GET 요청, 'example.txt' 파일)
xhr.open('GET', 'example.txt', true);

// 요청이 완료되었을 때 호출될 콜백 함수
xhr.onload = function () {
    if (xhr.status === 200) { // HTTP 상태 코드가 200 (성공) 일 때
        console.log(xhr.responseText); // 응답 텍스트 출력
    } else {
        console.error('Request failed. Returned statu
```

```
s of ' + xhr.status);
    }
};

// 요청 보내기
xhr.send();
```

- XMLHttpRequest 객체는 AJAX 기술의 핵심 구성 요소로, 웹 페이지가 전체를 다시 로드하지 않고도 서버와 통신할 수 있게 합니다.
- XMLHttpRequest는 이름에서 알 수 있듯이 원래 XML 데이터를 전송하기 위해 설계되었지만, 다양한 데이터 형식을 처리할 수 있습니다.
- AJAX의 개념은 XMLHttpRequest 객체의 도입과 함께 발전했으며, 이를 통해 웹 애플리케이션의 성능과 사용자 경험을 크게 향상시킬 수 있었습니다.

▼ XML

XML(Extensible Markup Language)은 데이터를 구조적으로 저장하고 전송하기 위해 설계된 마크업 언어입니다. XML은 인간이 읽을 수 있으며, 또한 컴퓨터 프로그램이 처리할 수 있도록 설계되었습니다. 주로 데이터의 교환 및 저장 목적으로 사용되며, 다음과 같은 특징을 가지고 있습니다:

1. **자체 설명적 데이터**: 태그와 속성을 사용하여 데이터를 설명합니다.
2. **유연성**: 사용자가 직접 태그를 정의할 수 있어 다양한 데이터 구조를 표현할 수 있습니다.
3. **계층 구조**: 트리 구조로 데이터를 구성하여 계층적 관계를 쉽게 표현할 수 있습니다.
4. **플랫폼 독립성**: 다양한 시스템과 프로그램에서 동일한 방식으로 사용할 수 있습니다.
5. **표준화**: W3C(World Wide Web Consortium)에서 표준화되어 다양한 기술 및 도구에서 지원됩니다.

다음은 XML 문서의 간단한 예입니다:

```
<note>
```

```
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

이 XML 문서는 "note"라는 루트 엘리먼트를 포함하고 있으며, 그 안에 "to", "from", "heading", "body"와 같은 자식 엘리먼트들이 포함되어 있습니다. 이를 통해 "to", "from", "heading", "body" 각각의 데이터가 무엇을 의미하는지 명확하게 알 수 있습니다.

▼ XML vs HTML

XML(Extensible Markup Language)과 HTML(HyperText Markup Language)은 둘 다 마크업 언어이지만, 목적과 사용 방식에 있어서 여러 차이점이 있습니다.

XML과 HTML은 각각 데이터 저장 및 전송, 그리고 웹 페이지의 구조와 콘텐츠를 정의하는 데 특화된 언어입니다. XML은 데이터의 의미와 구조를 정의하는 데 중점을 두고 있으며, HTML은 웹 페이지를 구성하는 데 사용됩니다. 이 두 언어는 서로 다른 목적을 위해 설계되었으며, 각자의 영역에서 중요한 역할을 합니다.

1. 목적

- **XML**: 주로 데이터의 저장 및 전송을 위한 언어입니다. 데이터의 구조를 정의하고, 다른 시스템 간에 데이터를 교환할 수 있도록 설계되었습니다.
- **HTML**: 웹 페이지를 만들기 위한 언어입니다. 문서의 구조를 정의하고, 텍스트, 이미지, 링크 등을 포함하여 웹 페이지의 콘텐츠와 레이아웃을 표현하는 데 사용됩니다.

2. 태그 및 속성

- **XML**: 사용자가 태그와 속성을 자유롭게 정의할 수 있습니다. 태그 이름과 구조는 데이터의 의미에 따라 결정됩니다. 예를 들어:

```
<book>
    <title>XML Guide</title>
    <author>John Doe</author>
</book>
```

- **HTML**: 정해진 태그와 속성을 사용해야 합니다. 각 태그는 특정한 기능과 의미를 가지고 있습니다. 예를 들어:

```
<h1>Welcome to My Website</h1>
<p>This is a paragraph.</p>
```

3. 문법

- **XML**: 엄격한 문법을 따릅니다. 모든 태그는 짝을 이루어야 하며, 속성 값은 반드시 큰따옴표로 감싸야 합니다. 예를 들어:

```
<note>
    <to>Tove</to>
    <from>Jani</from>
</note>
```

- **HTML**: 비교적 유연한 문법을 가집니다. 일부 태그는 닫는 태그가 필요 없으며, 속성 값도 큰따옴표 없이 사용할 수 있습니다. 예를 들어:

```
<img src="image.jpg" alt=Picture>
```

4. 데이터 구조

- **XML**: 계층적 구조를 통해 데이터를 표현합니다. 복잡한 데이터 구조를 쉽게 표현할 수 있습니다.
- **HTML**: 문서의 구조를 표현합니다. 주로 웹 페이지의 레이아웃과 콘텐츠를 구성하는 데 사용됩니다.

▼ jQuery

jQuery는 웹 개발에서 자주 사용되는 **오픈 소스 자바스크립트 라이브러리**입니다.

주로 HTML 문서 탐색, 이벤트 처리, 애니메이션, Ajax 통신 등을 쉽게 처리할 수 있게 해주는 도구입니다.

jQuery는 다음과 같은 장점을 가지고 있습니다:

1. **간단한 문법**: 복잡한 자바스크립트 코드를 간단하게 작성할 수 있습니다. 예를 들어, DOM 요소를 선택하고 조작하는 것이 매우 직관적입니다.
2. **브라우저 호환성**: 다양한 웹 브라우저에서 동일하게 동작하도록 만들어져, 브라우저 간 호환성 문제를 줄여줍니다.
3. **플러그인 확장성**: 다양한 플러그인을 통해 기능을 확장할 수 있습니다. 커뮤니티에서 제공하는 수많은 플러그인을 활용할 수 있습니다.
4. **Ajax 지원**: Ajax 요청을 쉽게 만들고 처리할 수 있습니다.
5. **애니메이션**: HTML 요소에 애니메이션 효과를 쉽게 적용할 수 있습니다.

다음은 jQuery의 기본 사용 예제입니다:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>jQuery Example</title>
    <script src="https://code.jquery.com/jquery-3.6.0.mi
n.js"></script>
    <script>
        $(document).ready(function(){
            $("button").click(function(){
                $("p").hide();
            });
        });
    </script>
</head>
<body>
    <p>If you click on the "Hide" button, I will disappe
ar.</p>
    <button>Hide</button>
</body>
```

```
</html>
```

이 예제에서는 페이지가 로드되면 jQuery가 버튼 클릭 이벤트를 감지하고, 클릭 시 해당 페이지의 <p> 요소를 숨깁니다. jQuery를 사용하면 이러한 작업을 매우 간단하게 구현할 수 있습니다.

"다소 배우기가 까다로운 자바스크립트보다 쉽고 직관적인 jquery를 선호하는 개발자가 늘었었다"는 뜻?

jQuery가 등장했을 당시, 자바스크립트 자체는 매우 강력한 언어였지만 다소 복잡하고 다양한 브라우저 간의 호환성 문제가 있었습니다. 특히, DOM 요소를 선택하고 조작하는 것, 이벤트를 처리하는 것, Ajax 요청을 보내는 것 등이 기본 자바스크립트로는 코드가 길고 복잡해지는 경우가 많았습니다.

jQuery는 이러한 자바스크립트의 복잡성을 크게 줄여주었고, 다음과 같은 이유로 많은 개발자들이 선호하게 되었습니다:

1. **단순한 문법**: jQuery는 직관적이고 간단한 문법을 제공하여 코드 작성이 훨씬 쉬워졌습니다. 예를 들어, 특정 HTML 요소를 선택하고 조작하는 것이 jQuery를 사용하면 훨씬 간단해집니다.

   기본 자바스크립트:

   ```
   document.getElementById('myElement').style.display = 'none';
   ```

   jQuery:

   ```
   $('#myElement').hide();
   ```

2. **브라우저 호환성 문제 해결**: jQuery는 다양한 브라우저에서 동일하게 동작하도록 만들어져 브라우저 간 호환성 문제를 신경 쓰지 않아도 되었습니다. 이는 웹 개발자들이 크로스 브라우징 문제를 걱정하지 않고 개발에 집중할 수 있게 해주었습니다.

3. **풍부한 기능과 플러그인**: jQuery는 다양한 내장 기능을 제공할 뿐만 아니라, 수많은 플러그인들이 개발되어 쉽게 기능을 확장할 수 있었습니다. 이를 통해 개발자들은 복잡한 기능도 빠르게 구현할 수 있었습니다.

4. **짧은 학습 곡선**: jQuery는 배우기 쉽고 직관적이어서, 자바스크립트를 처음 접하는 개발자들도 쉽게 익힐 수 있었습니다.

이러한 이유들로 인해, 많은 개발자들이 자바스크립트의 복잡성을 피하고 jQuery를 사용하게 되었고, 이는 jQuery가 웹 개발의 주류 라이브러리로 자리잡게 된 이유입니다.

그러나 최근에는 자바스크립트 자체가 발전하고(예: ES6+), React, Vue.js 같은 프레임워크들이 등장하면서 jQuery의 사용이 줄어들고 있는 추세입니다.

▼ 오픈소스

오픈소스(Open Source)는 소프트웨어의 소스 코드를 공개하여 누구나 자유롭게 사용, 수정, 배포할 수 있도록 허용하는 소프트웨어를 의미합니다. 오픈소스 소프트웨어는 보통 다음과 같은 특징을 가지고 있습니다:

1. **소스 코드 공개**: 소프트웨어의 소스 코드를 누구나 열람할 수 있도록 공개합니다.
2. **자유로운 사용**: 누구나 소프트웨어를 자유롭게 사용하고 실행할 수 있습니다.
3. **수정 및 배포 가능**: 소프트웨어를 수정하거나 기능을 추가하여 재배포할 수 있습니다.
4. **커뮤니티 참여**: 오픈소스 프로젝트는 보통 여러 사람들의 협업으로 발전하며, 누구나 기여할 수 있습니다.

오픈소스 소프트웨어의 예로는 리눅스(Linux) 운영체제, 아파치(Apache) 웹 서버, 파이썬(Python) 프로그래밍 언어 등이 있습니다. 오픈소스 소프트웨어는 기업이나 개인 개발자들에게 큰 혜택을 제공하며, 기술 발전에 중요한 역할을 하고 있습니다.

▼ 🌟Node.js

Node.js는 구글의 V8 JavaScript 엔진을 기반으로 빌드된 자바스크립트 런타임 환경입니다.

Node.js는 자바스크립트를 서버 측에서 실행할 수 있는 런타임 환경을 제공합니다.

Node.js는 비동기식 이벤트 구동 구조를 가지고 있어, 높은 처리량과 낮은 지연 시간의 네트워크 애플리케이션을 작성하는 데 적합합니다. 또한, npm(Node Package

Manager)을 통해 다양한 모듈과 패키지를 쉽게 관리하고 설치할 수 있어 개발 생산성을 높이는 데 큰 도움을 줍니다.

▼ 빌드되다

"빌드되었다"는 소프트웨어 개발에서 **소스 코드를 컴파일하거나 다른 방식으로 변환하여 실행 가능한 프로그램이나 라이브러리로 만드는 과정**을 의미합니다. Node.js는 구글 V8 JavaScript 엔진을 사용하여 이러한 변환 과정을 거쳐 만들어졌습니다.

▼ 런타임 환경

런타임 환경(runtime environment)은 프로그램이 실행될 때 필요한 모든 소프트웨어와 하드웨어 리소스를 제공하는 환경을 말합니다. 이를 통해 프로그램이 제대로 동작할 수 있도록 지원합니다. 좀 더 구체적으로 설명하자면:

1. **라이브러리와 프레임워크**:
   - 프로그램이 실행되면서 필요로 하는 다양한 라이브러리와 프레임워크를 포함합니다. 예를 들어, 자바스크립트 프로그램이 특정 기능을 수행하기 위해 사용하는 코드 모듈들이 런타임 환경에 포함됩니다.
2. **가상 머신이나 인터프리터**:
   - 프로그래밍 언어에 따라, 코드가 실행되기 위해서는 가상 머신(VM)이나 인터프리터가 필요할 수 있습니다. 예를 들어, Java 프로그램은 Java Virtual Machine(JVM)에서 실행됩니다. Node.js는 구글 V8 엔진을 사용하여 자바스크립트를 실행합니다.
3. **시스템 자원 접근**:
   - 파일 시스템, 네트워크, 메모리 등 시스템 자원에 접근하고 이를 관리할 수 있는 기능을 제공합니다. 프로그램이 실행되면서 파일을 읽고 쓰거나 네트워크 요청을 처리하는 등의 작업을 할 수 있도록 돕습니다.
4. **환경 변수와 설정**:
   - 프로그램이 실행될 때 필요한 환경 변수와 설정 정보를 제공합니다. 예를 들어, 데이터베이스 연결 정보, API 키, 디버그 모드 설정 등이 포함될 수 있습니다.
5. **입출력 처리**:
   - 프로그램이 사용자로부터 입력을 받고, 결과를 출력할 수 있도록 입력과 출력(IO) 기능을 제공합니다.

Node.js의 런타임 환경은 서버 측에서 자바스크립트 코드를 실행할 수 있도록 위의 모든 기능을 제공하며, 특히 비동기식 이벤트 처리를 효율적으로 지원하여 고성능 네트워크 애플리케이션을 작성하는 데 적합합니다.

▼ 엔진

엔진(engine)은 프로그래밍 언어의 **코드를 해석하고 실행**하는 핵심 소프트웨어 구성 요소입니다. 구글 V8 엔진은 구글 크롬 브라우저에서 사용되는 고성능 자바스크립트 엔진으로, 자바스크립트 코드를 기계어로 변환하여 빠르게 실행할 수 있게 해줍니다. Node.js는 이 V8 엔진을 사용하여 자바스크립트를 서버 측에서도 빠르고 효율적으로 실행할 수 있게 합니다.

p46 **“Node.js는 비동기I/O를 지원하며 단일스레드 이벤트 루프를 기반으로 동작함으로써 요청 처리 성능이 좋다. 따라서 Node.js는 데이터를 실시간으로 처리하기 위해 I/O가 빈번하게 발생하는 SPA에 적합하다. 하지만 CPU사용률이 높은 애플리케이션에는 권장하지않는다. “**

Node.js는 단일 스레드 이벤트 루프를 기반으로 동작합니다. 이는 Node.js가 주로 I/O 작업에 최적화되어 있다는 것을 의미합니다. 파일 시스템 접근, 네트워크 요청, 데이터베이스 쿼리와 같은 작업은 비동기적으로 처리되며, 이벤트 루프는 이러한 작업들이 완료되면 콜백 함수를 호출하여 결과를 처리합니다.

이러한 구조는 I/O 작업에는 매우 효율적이지만, CPU 사용률이 높은 애플리케이션에는 적합하지 않을 수 있습니다. CPU를 많이 사용하는 작업(예: 복잡한 계산, 대규모 데이터 처리 등)은 단일 스레드에서 실행되기 때문에 이벤트 루프를 차단하고, 결과적으로 다른 작업들이 지연될 수 있습니다.

따라서, CPU 집약적인 작업이 많은 애플리케이션의 경우, Node.js를 사용하는 것보다는 멀티스레드를 지원하는 다른 기술을 사용하는 것이 더 나을 수 있습니다. 예를 들어, Node.js 애플리케이션에서는 CPU 집약적인 작업을 처리할 때, 워커 스레드(worker threads)나 클러스터링(cluster) 기능을 이용하여 작업을 분산시키는 방법을 사용할 수 있습니다. 이를 통해 일부 CPU 집약적인 작업을 처리하는 동안에도 메인 이벤트 루프가 원활하게 동작할 수 있도록 할 수 있습니다.

▼ 단일스레드

Node.js는 단일 스레드로 동작합니다. 이는 모든 코드가 하나의 스레드에서 실행된다는 것을 의미합니다. 단일 스레드는 자바스크립트의 특성 중 하나로, 이를 통해 코드 실행이 단순해지고, 멀티 스레딩에서 발생할 수 있는 복잡한 문제(예: 동기화, 데드락)를 피할 수 있습니다.

▼ 🌟🌟🌟**이벤트 루프**

이벤트 루프(Event Loop)는 JavaScript, 특히 Node.js와 같은 단일스레 환경에서 비동기 작업을 효율적으로 처리하기 위한 메커니즘입니다.

주요 역할은 비동기 작업(예: 타이머, 파일 I/O, 네트워크 요청 등)이 완료되었을 때 콜백 함수를 호출하여 결과를 처리하는 것입니다.

이벤트 루프의 구성 요소

1. **콜 스택 (Call Stack)**: 함수 호출이 쌓이는 스택 구조로, 현재 실행 중인 코드와 호출된 함수들이 저장되는 공간입니다.
2. **이벤트 큐 (Event Queue)**: 실행 대기 중인 비동기 작업(예: 콜백 함수, 이벤트 핸들러)이 저장되는 큐입니다.
3. **웹 API (Web APIs)**: 타이머, AJAX 요청, DOM 이벤트와 같은 비동기 작업을 처리하는 브라우저 제공 API입니다. Node.js에서는 파일 시스템 접근, 네트워크 요청 등 비동기 I/O 작업을 위한 자체 API를 사용합니다.

이벤트 루프의 작동 방식

1. **동기 작업 처리**: 자바스크립트 코드가 실행되면, 함수 호출이 콜 스택에 쌓이고 하나씩 실행됩니다.
2. **비동기 작업 처리**: 비동기 작업이 호출되면, 해당 작업은 웹 API나 Node.js의 비동기 API로 넘겨지고, 콜 스택에서 제거됩니다. 작업이 완료되면 콜백 함수가 이벤트 큐에 추가됩니다.
3. **이벤트 루프 실행**: 이벤트 루프는 콜 스택이 비어 있는지 확인합니다. 콜 스택이 비어 있으면, 이벤트 큐에서 대기 중인 콜백 함수를 콜 스택으로 이동시켜 실행합니다.

이 과정을 통해 Node.js는 단일 스레드 환경에서도 여러 비동기 작업을 효율적으로 처리할 수 있습니다.

예시

```
console.log('Start');
```

```
setTimeout(() => {
  console.log('Timeout callback');
}, 1000);

console.log('End');
```

위 코드의 실행 순서는 다음과 같습니다:

1. 'Start'가 콜 스택에 쌓여 실행됩니다.
2. **setTimeout 함수가 호출되면, 비동기 작업으로 웹 API로 넘겨집니다.**
3. 'End'가 콜 스택에 쌓여 실행됩니다.
4. **1초 후, 타이머가 만료되면 setTimeout의 콜백 함수가 이벤트 큐에 추가됩니다.**
5. **이벤트 루프는 콜 스택이 비어 있음을 확인한 후, 이벤트 큐에서 콜백 함수를 가져와 콜 스택에 쌓고 실행합니다.**

이렇게 이벤트 루프는 단일 스레드 환경에서도 비동기 작업을 효율적으로 관리하며, Node.js의 성능을 최적화합니다. 그러나 CPU 집약적인 작업은 단일 스레드에서 실행되므로 이벤트 루프를 차단하고, 다른 작업이 지연될 수 있으므로 주의해야 합니다.

▼ 크로스 플랫폼

크로스 플랫폼(Cross-Platform)은 **소프트웨어나 애플리케이션이 여러 운영체제(OS)에서 동작할 수 있도록 개발된 것**을 의미합니다. 크로스 플랫폼 소프트웨어는 윈도우, macOS, 리눅스, 안드로이드, iOS 등 다양한 환경에서 실행될 수 있습니다. 이를 통해 사용자들은 특정 운영체제에 구애받지 않고 동일한 소프트웨어를 사용할 수 있습니다.

크로스 플랫폼 개발의 주요 이점은 다음과 같습니다:

1. **개발 비용 절감**: 하나의 코드베이스로 여러 플랫폼에서 사용할 수 있어, 개별 플랫폼별로 개발할 필요가 없습니다.
2. **유지보수 용이**: 버그 수정이나 기능 업데이트를 한 번만 하면 모든 플랫폼에 적용되므로 유지보수가 용이합니다.
3. **시장 도달성 증가**: 다양한 플랫폼을 지원하므로 더 많은 사용자에게 도달할 수 있습니다.

주요 크로스 플랫폼 프레임워크 및 도구로는 다음과 같은 것들이 있습니다:

- **React Native**: 페이스북에서 개발한 모바일 애플리케이션 프레임워크로, 자바스크립트를 사용하여 iOS와 안드로이드 애플리케이션을 개발할 수 있습니다.
- **Flutter**: 구글에서 개발한 UI 툴킷으로, 하나의 코드베이스로 iOS와 안드로이드 앱을 개발할 수 있습니다.
- **Xamarin**: 마이크로소프트에서 개발한 크로스 플랫폼 프레임워크로, C#과 .NET을 사용하여 모바일 애플리케이션을 개발할 수 있습니다.
- **Electron**: 웹 기술(HTML, CSS, JavaScript)을 사용하여 데스크탑 애플리케이션을 개발할 수 있는 프레임워크로, Atom 에디터와 Visual Studio Code가 이 프레임워크로 만들어졌습니다.

▼ 🌟🌟SPA

SPA는 "Single Page Application"의 약자로, **한 번 웹 페이지를 로드한 후 페이지 전체를 다시 로드하지 않고, 사용자와의 상호작용을 통해 필요한 부분만 동적으로 업데이트하는 웹 애플리케이션을 의미합니다.** 이렇게 하면 사용자 경험이 더 부드럽고 빠르게 느껴집니다.

SPA의 주요 특징은 다음과 같습니다:

1. **빠른 응답 시간**: 페이지 전환 시 서버로부터 전체 페이지를 다시 로드하지 않고, 필요한 데이터만 가져와 갱신합니다.
2. **단일 HTML 페이지**: 애플리케이션 전체가 하나의 HTML 페이지에 로드됩니다.
3. **클라이언트 사이드 라우팅**: URL 변화에 따라 서버로 요청을 보내지 않고, 클라이언트 측에서 자바스크립트를 사용해 필요한 부분만 업데이트합니다.
4. **자바스크립트 프레임워크**: 주로 React, Angular, Vue.js와 같은 프레임워크를 사용하여 구현됩니다.

▼ 🌟🌟**CBD 방법론**

Component-Based Development (CBD)는 소프트웨어 개발 방법론 중 하나로, **소프트웨어 시스템을 독립적인 재사용 가능한 컴포넌트(모듈)로 나누어 개발하는 접근 방식을 말합니다.** 이 방법론은 대규모 소프트웨어 시스템의 개발 및 유지보수를 효율적으로 하기 위해 도입되었습니다. CBD의 주요 특징과 장점을 설명하면 다음과 같습니다:

1. **재사용성 (Reusability)**: 컴포넌트는 독립적이며 재사용 가능하게 설계되어, 다른 프로젝트나 시스템에서도 동일한 컴포넌트를 재사용할 수 있습니다. 이는 개발 시간을 단축하고 비용을 절감하는 데 도움이 됩니다.
2. **모듈성 (Modularity)**: 시스템을 작은 모듈로 나누어 개발함으로써 각 모듈은 독립적으로 개발, 테스트 및 유지보수될 수 있습니다. 이는 시스템의 복잡성을 줄이고, 특정 기능의 변경이 전체 시스템에 미치는 영향을 최소화합니다.
3. **확장성 (Scalability)**: 새로운 기능을 추가하거나 기존 기능을 확장할 때 전체 시스템을 수정할 필요 없이 필요한 컴포넌트만 수정하거나 추가하면 됩니다.
4. **유지보수성 (Maintainability)**: 독립적인 컴포넌트로 구성된 시스템은 버그 수정, 업데이트 및 개선 작업이 용이합니다. 문제가 발생한 컴포넌트를 찾아서 수정하는 것이 더 쉽기 때문입니다.
5. **표준화 (Standardization)**: 컴포넌트는 명확한 인터페이스와 표준을 따르기 때문에 서로 다른 개발 팀이나 조직 간의 협업이 수월해집니다.

CBD는 주로 대규모 엔터프라이즈 시스템, 분산 시스템 및 웹 애플리케이션 개발에서 많이 사용됩니다. 또한, 서비스 지향 아키텍처(SOA)나 마이크로서비스 아키텍처와 같은 현대적인 소프트웨어 아키텍처 패턴에서도 중요한 역할을 합니다.

대표적인 예로는, 웹 개발에서 널리 사용되는 React.js와 같은 프론트엔드 프레임워크가 있습니다. React는 사용자 인터페이스를 컴포넌트 단위로 나누어 개발할 수 있게 하여, 각 컴포넌트가 독립적으로 개발 및 재사용될 수 있게 합니다.

▼ 어플리케이션 아키텍처

어플리케이션 아키텍처(Application Architecture)는 소프트웨어 어플리케이션을 구성하는 요소들 간의 구조와 상호작용을 정의하는 방법론입니다.

이는 소프트웨어 시스템이 어떻게 설계되고 구현되며 배포되는지를 규정하는 일련의 원칙, 패턴 및 지침을 포함합니다. 어플리케이션 아키텍처는 소프트웨어 개발의 전반적인 품질, 유지보수성, 확장성 및 성능에 큰 영향을 미칩니다.

주요 개념

1. **레이어(Layers)**: 어플리케이션 아키텍처는 종종 여러 레이어로 나뉩니다. 각 레이어는 특정 기능을 수행하며, 레이어 간의 상호작용을 통해 전체 시스템이 동작합니다. 일반적인 레이어로는 프레젠테이션 레이어(사용자 인터페이스), 비즈니스 로직 레이어, 데이터 액세스 레이어가 있습니다.

2. **모듈(Modularity)**: 어플리케이션을 독립적인 모듈로 나누어 개발, 테스트, 유지보수를 용이하게 합니다. 모듈화는 재사용성과 관리 용이성을 높입니다.
3. **컴포넌트(Components)**: 어플리케이션의 기능적 단위로, 각 컴포넌트는 독립적으로 배포되고 재사용될 수 있습니다. 컴포넌트는 명확한 인터페이스를 통해 다른 컴포넌트와 상호작용합니다.
4. **서비스(Services)**: 서비스 지향 아키텍처(SOA)와 마이크로서비스 아키텍처와 같은 패턴에서는 어플리케이션 기능을 독립적인 서비스로 구성합니다. 각 서비스는 특정 비즈니스 기능을 수행하며, 다른 서비스와 통신하여 전체 어플리케이션을 구성합니다.
5. **패턴(Patterns)**: 어플리케이션 아키텍처는 설계 패턴을 활용하여 일반적인 문제에 대한 검증된 해결책을 제공합니다. 예를 들어, MVC(Model-View-Controller) 패턴, MVP(Model-View-Presenter) 패턴 등이 있습니다.

주요 요소

1. **프레젠테이션 레이어**: 사용자와 상호작용하는 인터페이스를 담당합니다. UI 컴포넌트, 웹 페이지, 모바일 앱 화면 등이 포함됩니다.
2. **비즈니스 로직 레이어**: 어플리케이션의 핵심 비즈니스 규칙과 로직을 처리합니다. 데이터 처리, 계산, 상태 관리 등이 포함됩니다.
3. **데이터 액세스 레이어**: 데이터베이스나 외부 데이터 소스로부터 데이터를 가져오고 저장하는 역할을 합니다. SQL 쿼리, ORM(Object-Relational Mapping) 등이 사용됩니다.
4. **통신 레이어**: 분산 시스템에서 컴포넌트나 서비스 간의 통신을 관리합니다. HTTP, REST API, 메시지 큐 등이 포함됩니다.
5. **보안 레이어**: 어플리케이션의 보안 요구사항을 처리합니다. 인증, 권한 부여, 데이터 암호화 등이 포함됩니다.

아키텍처 스타일

1. **단일 아키텍처 (Monolithic Architecture)**: 모든 기능이 하나의 코드베이스에 포함된 구조로, 단일 배포 단위를 갖습니다. 관리가 단순하지만, 확장성과 유지보수에 어려움이 있을 수 있습니다.
2. **마이크로서비스 아키텍처 (Microservices Architecture)**: 어플리케이션을 작고 독립적인 서비스로 나누어 각 서비스가 독립적으로 배포되고 확장될 수 있

도록 합니다. 확장성과 유연성이 뛰어나지만, 서비스 간의 통신과 관리가 복잡해질 수 있습니다.

3. **서버리스 아키텍처 (Serverless Architecture)**: 서버 관리를 신경쓰지 않고 기능 단위로 어플리케이션을 개발하여 클라우드에서 실행합니다. 비용 효율성과 빠른 배포가 가능하지만, 벤더 종속성과 한계가 있을 수 있습니다.

▼ 서비스 지향 아키텍처(SOA, Service-Oriented Architecture)

서비스 지향 아키텍처(SOA)와 마이크로서비스 아키텍처는 현대적인 소프트웨어 아키텍처 패턴으로, 대규모 시스템의 유연성과 확장성을 높이는 데 중점을 둡니다.

SOA는 소프트웨어 시스템을 서비스 단위로 구성하는 아키텍처 패턴입니다. 각 서비스는 독립적으로 작동하며, 특정 비즈니스 기능을 수행합니다.

SOA의 주요 특징은 다음과 같습니다:

1. **서비스**: SOA에서 서비스는 독립적인 비즈니스 기능을 수행하는 단위입니다. 서비스는 네트워크를 통해 호출할 수 있는 명확한 인터페이스를 가지고 있습니다.
2. **재사용성**: 서비스는 독립적으로 설계되어 다른 응용 프로그램이나 서비스에서 재사용될 수 있습니다.
3. **상호 운용성**: SOA는 서로 다른 플랫폼과 기술을 사용하는 시스템 간의 상호 운용성을 지원합니다. 이는 주로 표준 프로토콜(예: SOAP, REST)을 통해 이루어집니다.
4. **유연성**: 새로운 기능을 추가하거나 기존 기능을 변경할 때 전체 시스템을 수정할 필요 없이 특정 서비스를 수정하거나 추가할 수 있습니다.

▼ 마이크로서비스 아키텍처(Microservices Architecture)

마이크로서비스 아키텍처는 SOA의 개념을 더 작은 단위로 세분화하여, 각각의 마이크로서비스가 독립적으로 배포되고 운영될 수 있도록 하는 패턴입니다. 마이크로서비스 아키텍처의 주요 특징은 다음과 같습니다:

1. **독립적 배포**: 각 마이크로서비스는 독립적으로 배포될 수 있으며, 하나의 서비스가 업데이트되거나 장애가 발생해도 전체 시스템에 영향을 주지 않습니다.
2. **작은 서비스**: 마이크로서비스는 매우 작은 단위의 서비스로, 특정 비즈니스 기능을 수행합니다. 이는 서비스가 더 작고 관리하기 쉽도록 합니다.

3. **다양한 기술 스택**: 각 마이크로서비스는 독립적으로 개발되기 때문에, 서로 다른 프로그래밍 언어와 기술 스택을 사용할 수 있습니다.
4. **자율 팀**: 마이크로서비스 아키텍처는 팀이 독립적으로 작업할 수 있도록 하여, 빠른 개발과 배포 주기를 가능하게 합니다.

▼ ECMAScript

ECMAScript는 자바스크립트의 표준 사양인 ECMA-262를 말하며, 프로그래밍 언어의 값, 타입, 객체와 프로퍼티, 함수, 표준 빌트인 객체 등 핵심 문법을 규정한다.

각 브라우저 제조사는 ECMAScript 사양을 준수해서 브라우저에 내장되는 자바스크립트 엔진을 구현한다.

▼ 🌟클라이언트 사이드 Web API

자바스크립트는 프로그래밍 언어로서 기본 뼈대를 이루는 ECMAScript와 브라우저가 별도로 지원하는 클라이언트 사이드 Web API를 아우르는 개념이다.

클라이언트 사이드 Web API는 Node.js환경에서는 실행할 수 없다.

클라이언트 사이드 Web API는 ECMAScript와는 별도로 월드 와이드 웹 콘소시엄 World Wide Web Consortium: W3C에서 별도의 사양으로 관리하고 있다.

Web API(DOM,BOM,Canvas, XMLHttpRequest, fetch, requestAnimationFrame, SVG, Web Storage, Web Component, Web Worker 등)

▼ 인터프리터 언어 vs 컴파일러 언어

p14 전통적인 컴파일러 언어와 인터프리터 언어를 비교하면 다음과 같다.

컴파일러 언어	인터프리터 언어
코드가 실행되기 전 단계인 컴파일 타임에 소스코드 전체를 한번에 머신코드로 변환한 후 실행한다.	코드가 실행되는 단계인 런타임에 문 단위로 한 줄씩 중간코드인 바이트코드로 변환한 후 실행한다.
실행파일을 생성한다	실행 파일을 생성하지 않는다.

컴파일러 언어	인터프리터 언어
컴파일 단계와 실행 단계가 분리되어 있다. 명시적인 컴파일 단계를 거치고, 명시적으로 실행 파일을 실행한다.	인터프리트 단계와 실행 단계가 분리되어 있지 않다. 인터프리터는 한 줄 씩 바이트 코드로 변환하고 즉시 실행한다.
실행에 앞서 컴파일은 단 한번 수행된다.	코드가 실행될 때마다 인터프리트 과정이 반복 수행된다.
컴파일과 실행 단계가 분리되어 있으므로 코드 실행 속도가 빠르다.	인터프리트 단계와 실행 단계가 분리되어 있지 않고 반복 수행되므로 코드 실행 속도가 비교적 느리다.

하지만 대부분의 모던 브라우저에서 사용되는 인터프리터는 전통적인 컴파일러 언어 처럼 명시적인 컴파일 단계를 거치지는 않지만 복잡한 과정을 거치며 일부 소스코드를 컴파일하고 실행한다. 이를 통해 **인터프리터 언어의 장점인 동적 기능 지원을 살리면서 실행 속도가 느리다는 단점을 극복한다.** 따라서 현재는 컴파일러와 인터프리터의 기술적 구분이 모호해져 가는 추세다.

하지만 **자바스크립트는 런타임에 컴파일되며 실행 파일이 생성되지 않고 인터프리터의 도움 없이 실행할 수 없기 때문에** 컴파일러 언어라고는 할수 없다.

▼ 🌟자바스크립트의 특징

- 자바스크립트는 HTML, CSS와 함께 웹을 구성하는 요소 중 하나로 **웹 브라우저에서 동작하는 유일한 프로그래밍 언어다.**
- 자바스크립트는 개발자가 별도의 컴파일 작업을 수행하지 않는 **인터프리터 언어**다.
- 자바스크립트는 **프로토타입 기반의 객체지향 언어**다.
- 자바스크립트는 명령형, 함수형, 프로토타입 기반 객체지향 프로그래밍을 지원하는 **멀티패러다임 프로그래밍 언어**다.

자바스크립트가 "명령형, 함수형, 프로토타입 기반 객체지향 프로그래밍을 지원하는 멀티 패러다임 프로그래밍 언어"라는 것은 자바스크립트가 **여러 가지 프로그래밍 스타일(패러다임)을 지원**하는 다목적 언어라는 뜻입니다. 각각의 패러다임에 대해 자세히 설명하면 다음과 같습니다:

1. **명령형 프로그래밍 (Imperative Programming)**:
    - **설명**: 프로그램이 수행해야 할 명령들을 순서대로 기술하는 방식입니다. 컴퓨터에게 무엇을 해야 하는지를 절차적으로 설명합니다.

- **예시**: 변수에 값을 대입하고, 조건문(if)이나 반복문(for)을 사용하여 코드의 흐름을 제어합니다.
- **자바스크립트 예**:

```
let sum = 0;
for (let i = 0; i < 10; i++) {
    sum += i;
}
console.log(sum);
```

2. **함수형 프로그래밍 (Functional Programming)**:
   - **설명**: 함수의 호출과 조합을 통해 프로그램을 작성하는 방식입니다. 상태와 부작용을 피하고 순수 함수를 사용하는 것을 지향합니다.
   - **예시**: 고차 함수(map, reduce) 등을 사용하여 데이터를 변환하고 처리합니다.
   - **자바스크립트 예**:

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map(n => n * 2);
console.log(doubled);
```

3. **프로토타입 기반 객체지향 프로그래밍 (Prototype-based Object-oriented Programming)**:
   - **설명**: 클래스 대신 객체를 직접 복사하여 새로운 객체를 만드는 방식입니다. 객체는 다른 객체를 프로토타입으로 삼아 상속받을 수 있습니다.
   - **예시**: 객체를 생성하고, 그 객체를 프로토타입으로 사용하여 다른 객체를 생성합니다.
   - **자바스크립트 예**:

```
const person = {
    name: 'Alice',
    greet() {
        console.log(`Hello, my name is ${this.nam
```

```
e}`);
    }
};

const anotherPerson = Object.create(person);
anotherPerson.name = 'Bob';
anotherPerson.greet();  // Hello, my name is Bob
```

멀티 패러다임 프로그래밍 언어 (Multi-paradigm Programming Language):

- **설명**: 자바스크립트는 위에서 설명한 명령형, 함수형, 객체지향 프로그래밍을 모두 지원합니다. 즉, 개발자는 문제에 맞는 다양한 패러다임을 조합하여 사용할 수 있습니다.
- **장점**: 다양한 프로그래밍 스타일을 활용할 수 있으므로, 문제 해결에 적합한 접근 방식을 선택할 수 있습니다. 이는 유연성과 표현력을 높여 줍니다.

요약하자면, 자바스크립트는 여러 프로그래밍 스타일을 모두 지원하는 다재다능한 언어이며, 개발자는 이를 통해 다양한 문제를 효과적으로 해결할 수 있습니다.