리액트 인액션 - 챕터 1

컴포넌트

- 리액트에서 구현하고자 하는 기능을 캡슐화하는 기본 단위
- 데이터(속성 및 상태)를 다루고 결과를 UI로 렌더링

라이브러리

React Core

- 리액트 코어 라이브러리는 `react-dom` 및 `react-native` 라이브러리와 함께 동작
- 컴포넌트의 명세 및 정의에 중점을 둠

React DOM

- UI를 렌더링할 수 있는 렌더러 중 하나를 구현한 라이브러리
- 브라우저 환경 및 서버측 렌더링 기능 구현 목적

React Native

- 네이티브 플랫폼에서 UI 렌더링을 위한 라이브러리
- iOS, 안드로이드 등 다른 플랫폼을 위한 리액트 애플리케이션 개발 시 사용

서드파티 라이브러리

- 리액트는 프런트엔드 애플리케이션이 필요로 하는 여러 기능을 지원하지 않음
- 하지만 리액트를 지원하는 다양한 라이브러리가 존재

리액트 애플리케이션 실행

- 어떤 플랫폼에서도 실행 가능
  - 웹, 브라우저 및 서버 기반 애플리케이션
  - React Native, React VR 등 다른 프로젝트를 이용해 그 외의 플랫폼에서 실행되는 앱을 개발할 수도 있음

리액트를 배우기 전

선수 지식

- 리액트는 자바스크립트를 모른다고 해서 배울 수 없다거나 얻는 것이 없는 것은 아님
- 하지만 자바스크립트를 먼저 학습 후 리액트를 학습하면 더욱 도움이 됨
- 자바스크립트에 익숙해져야 리액트 학습에 더 도움이 됨

필요한 자바스크립트 지식

- **ES6+ 문법**: 화살표 함수, 구조 분해 할당, 템플릿 리터럴
- **함수형 프로그래밍**: map, filter, reduce 등 고차 함수
- **비동기 프로그래밍**: Promise, async/await

- **모듈 시스템**: import/export

유용한 도구

패키지 관리자

- **npm**: 자바스크립트를 위한 패키지 관리 도구
- **yarn**: 페이스북에서 개발한 더 빠른 패키지 관리자
- **pnpm**: 디스크 공간을 절약하는 패키지 관리자

개발 도구

- **Create React App**: 리액트 애플리케이션을 쉽게 시작할 수 있는 도구
- **Vite**: 빠른 빌드 도구
- **Webpack**: 모듈 번들러
- **Babel**: 자바스크립트 컴파일러

가상 DOM

개요

- 리액트의 핵심: 복잡한 작업을 단순화, 개발 과정에 불필요한 추상화 제거
- 개발자는 컴포넌트가 어떻게 동작하는지, 여러 상태를 어떻게 보여줄 것인지를 선언
- 리액트 내부 메커니즘은 컴포넌트의 상태 변경과 이를 UI에 반영하는 작업 등에 필요한 복잡한 과정 처리
- 위 과정을 처리하는 주요 기법이 바로 **가상 DOM**

정의

- 브라우저에 존재하는 문서 객체 모델을 흉내 내거나 반영하는 데이터 구조 또는 데이터 구조의 모음

작동 원리

1. **렌더링**: 컴포넌트가 JSX를 반환하면 가상 DOM 트리가 생성됨
2. **재조정(Reconciliation)**: 상태 변경 시 새로운 가상 DOM 트리와 이전 트리를 비교
3. **Diffing**: 변경된 부분만 식별
4. **패치(Patch)**: 실제 DOM에 필요한 변경사항만 적용

DOM

정의

- DOM 또는 문서 객체 모델은 자바스크립트 프로그램이 다양한 종류의 문서를 다루기 위한 프로그래밍 인터페이스
- 실제 구현은 달라도 여러 웹 브라우저에 구현된 DOM은 거의 대동소이함

특징

- DOM을 통해 문서의 여러 부분을 구조적으로 접근, 저장하며 조작할 수 있음
- XML 문서의 계층 구조를 반영한 트리 구조, 노드로 표현되는 자식 구조들로 구성

DOM 조작의 문제점

- **비용이 큼**: DOM 조작은 브라우저의 레이아웃 계산과 리페인트를 유발
- **복잡성**: 직접 DOM을 조작하면 코드가 복잡해지고 버그 발생 가능성 증가
- **성능 이슈**: 불필요한 DOM 업데이트로 인한 성능 저하

가상 DOM의 필요성

문제점

- 대형 웹 애플리케이션을 개발하다 보면 직접 DOM을 조작하기 어려운 경우가 있음
- 통상적으로 이런 어려움은 변경 사항을 탐지하려 할 때 드러남
- 데이터가 변경되면 이 변경사항을 반영하기 위해 UI를 갱신해야 함

해결책

- 위 과정을 효과적이면서 쉽게 처리하기는 매우 어려우므로 리액트가 위 문제를 해결하는데 주안점을 두고 있음
- UI 갱신하는 과정이 복잡하고 어려운 이유는 브라우저가 DOM을 조작하는 방법에 있음

DOM 조작의 복잡성

- DOM 요소에 액세스하고 이를 갱신하거나 새로 생성하기 위해서는 지정된 요소를 구조화된 트리 구조로부터 조회해야 함
- 이 조회 과정은 UI를 갱신하기 위한 첫 번째 절차일 뿐
- 요소를 조회한 후에는 레이아웃, 크기 및 변경을 위한 기타 여러 동작 수행해야 하며 이들 대부분 작업은 복잡한 계산이 필요

가상 DOM의 역할

- 가상 DOM은 위와 같은 문제들을 해결해주는 능력뿐만 아니라 이런 제약을 처리하기 위해 DOM의 갱신을 최적으로 수행
- 애플리케이션을 개발하고 관리할 때 상당한 양의 변화가 DOM에서 발생
- 이러한 변화는 대부분 충돌을 일으키거나 최적화되지 않은 방식으로 처리
- 그 결과 복잡한 시스템이 개발되고 엔지니어들이 유지보수에 어려움을 겪으며 모든 면에서 피해를 보게 됨

성능과 설계 철학

- 성능 문제는 리액트의 디자인 및 구현에 있어 또 다른 핵심
- 가상 DOM을 구현함으로써 이 문제를 어느 정도 해결 가능하나 가상 DOM은 단지 충분히 빠르게 하기 위한 것
- 가상 DOM은 극한의 성능을 내기 위해서라기보다는 **견고한 API**, **간단한 멘탈 모델**, 그리고 **브라우저 간 호환성** 등 더 중요한 요소들을 함께 제공하기 위해 디자인됨

갱신과 변경

동작 방식

- 가상 DOM의 동작 방식은 3차원의 게임 방식과 유사한 점이 있음

- 게임 서버로부터 정보를 받음 → 게임 세계관에 정보를 보냄 → 가상 세계에 어떤 변화가 발생해야 하는지 결정한 후 렌더링 과정 구현
- 리액트는 UI 갱신을 최대한 효과적으로 수행하기 위해 위와 같은 유사한 방식을 도입

처리 과정

1. 리액트는 메모리에 가상 DOM 생성하고 관리
2. 리액트 DOM과 같은 렌더러는 가상 DOM의 변경 사항을 브라우저 DOM에 반영
3. 변경사항 처리
4. 메모리 DOM에서 발생한 변경 사항이 실제 DOM의 변경을 유발한다고 판단되는 경우에 수행
5. 변경이 발생하면 실제 DOM과 메모리 DOM 차이점 탐색
6. 그 후 브라우저 DOM을 효율적으로 갱신
7. 위 과정을 **비교 후 수정 과정**이라 부름

Diffing 알고리즘

- **트리 비교**: 두 가상 DOM 트리를 비교하여 차이점 찾기
- **키(Key) 사용**: 리스트 렌더링 시 각 항목에 고유 키를 부여하여 효율적인 비교
- **타입 비교**: 컴포넌트 타입이 다르면 전체 서브트리를 교체
- **속성 비교**: 같은 타입의 컴포넌트는 속성만 비교하여 업데이트

가상 DOM 속도가 중요?

성능 이상의 가치

- 가상 DOM은 빠른 속도 그 이상의 장점을 제공
- 성능은 리액트의 핵심 장점이기는 하지만 **간결함**이 그보다 더 우선하는 요소
- 가상 DOM은 복잡한 상태 관리 로직에서 벗어나 애플리케이션에서 더 중요한 부분에 집중하도록 도와주는 요소의 일부
- 속도와 간결함의 조합이 개발자에게는 리액트를 쓰는 매우 큰 이유 중 하나
- 가상 DOM이 이런 방식으로 돌아가는지만 이해하면 되고 깊게 파고들 필요는 없음

실제 성능 비교

- **순수 DOM 조작**: 직접 DOM을 조작하는 것보다는 느릴 수 있음
- **복잡한 애플리케이션**: 복잡한 상태 관리가 필요한 경우 가상 DOM이 유리
- **개발자 경험**: 성능보다는 개발 효율성과 코드 가독성이 더 중요

컴포넌트: 리액트의 기본 단위

정의

- 리액트는 컴포넌트를 생성하는 다양한 방법을 제공
- 컴포넌트란 더욱 큰 것의 일부
- 컴포넌트는 우리가 정의하는 어떤 형태로든 구현할 수 있음
    - 컴포넌트로서 적합한 형태가 아니더라도 (예: 인터페이스 전체를 단 하나의 컴포넌트로 해도 구현은 가능하지만 불필요함)
- 각각의 고유한 영역을 컴포넌트로 정의할 수 있음
- 같게 반복되는 아이템들은 하나의 컴포넌트로 정의하고 각기 다른 데이터를 바탕으로 재사용 가능

컴포넌트 종류

- **함수형 컴포넌트**: 함수로 정의된 컴포넌트 (현재 권장 방식)
- **클래스 컴포넌트**: 클래스로 정의된 컴포넌트 (레거시)
- **고차 컴포넌트(HOC)**: 다른 컴포넌트를 감싸는 컴포넌트
- **렌더 프롭**: 함수를 자식으로 받는 컴포넌트

컴포넌트 설계 원칙

- **단일 책임 원칙**: 하나의 컴포넌트는 하나의 책임만 가져야 함
- **재사용성**: 여러 곳에서 사용할 수 있도록 설계
- **테스트 가능성**: 독립적으로 테스트할 수 있어야 함
- **명확한 인터페이스**: props와 state가 명확해야 함

캡슐화와 재사용

특징

- 리액트 컴포넌트는 캡슐화되어 있고 재사용 및 재구성할 수 있음
- 복잡한 코드 대신 간결한 컴포넌트를 조합해서 애플리케이션을 개발할 수 있음
    - 예: 레고를 이용해서 뭔가 만드는 것과 유사

설계 원칙

- 리액트 기반의 컴포넌트를 정의할 때 컴포넌트 디자인에서 그 구조와 일관성에 대해 고려하는 것이 중요
- 위와 같이 진행함으로써 컴포넌트는 더 뛰어난 이식성을 갖추고 논리적으로 그룹화되며 다른 지점으로 이동이 쉽고 애플리케이션 전체에 걸쳐 재사용 가능

장점

- 컴포넌트 간의 경계는 기능의 분할과 잘 정의된 애플리케이션 구조를 확보할 수 있음
- 독립적인 컴포넌트는 재사용성과 이동성을 의미
- 컴포넌트는 상호작용하며 함께 동작
- 여러 컴포넌트를 조합해서 새로운 형태의 합성 컴포넌트를 만들 수도 있음
- 생명주기 메서드들은 예측이 가능하고 컴포넌트가 다른 생명주기 시점으로 이동할 때 활용할 수 있도록 잘 정의되어 있음

컴포넌트 통신

- **Props**: 부모에서 자식으로 데이터 전달
- **State**: 컴포넌트 내부 상태 관리
- **Context**: 전역 상태 관리
- **Refs**: DOM 요소에 직접 접근

팀을 위한 리액트

리액트가 팀에게 훌륭한 도구가 될 수 있는 이유

간결함

- 간결함은 쉽다는 것을 의미하지는 않음
- 대부분 문제를 빠르게 해결하려다 보면 코드가 지저분해지지만 리액트는 간결하고 유연함
- 강력한 추상화 덕분에 리액트는 간결함을 유지하면서도 얼마든지 고급 기술을 적용할 방법을 제공함으로 유연함과 견고함을 동시에 제공

가벼운 라이브러리

- 리액트는 책임과 기능 면에서 가벼운 라이브러리
- 애플리케이션에서 오로지 뷰에 관련된 부분만을 담당
- 따라서 현재 사용 중인 기술에 리액트를 접목하기 쉬우며 부족한 기능들은 원하는 도구들로 보완 가능

팀 개발에서의 장점

- **일관된 코드 스타일**: ESLint, Prettier 등으로 코드 스타일 통일
- **컴포넌트 재사용**: 공통 컴포넌트 라이브러리 구축 가능
- **테스트 용이성**: 각 컴포넌트를 독립적으로 테스트 가능
- **문서화**: Storybook 등을 통한 컴포넌트 문서화

리액트 생태계

상태 관리

- **Redux**: 예측 가능한 상태 컨테이너
- **MobX**: 반응형 상태 관리
- **Zustand**: 간단한 상태 관리
- **Recoil**: 페이스북에서 개발한 실험적 상태 관리

라우팅

- **React Router**: 클라이언트 사이드 라우팅
- **Next.js**: 서버 사이드 렌더링과 라우팅

UI 라이브러리

- **Material-UI**: 구글의 Material Design
- **Ant Design**: 엔터프라이즈급 UI 라이브러리
- **Chakra UI**: 접근성을 중시한 UI 라이브러리
- **Tailwind CSS**: 유틸리티 퍼스트 CSS 프레임워크

개발 도구

- **React DevTools**: 브라우저 확장 프로그램
- **Storybook**: 컴포넌트 개발 환경
- **Jest**: 테스트 프레임워크
- **Testing Library**: 컴포넌트 테스트 라이브러리

요약

- **리액트**는 사용자 인터페이스를 개발하기 위한 라이브러리
- **컴포넌트**에 기초한 간결하면서도 유연한 API를 제공

- **컴포넌트**는 리액트 기본 단위이며 리액트 애플리케이션 내에서 폭넓게 활용됨
- 개발자가 작성한 프로그램과 실제 브라우저 DOM 사이에서 이 둘을 연결해주는 **가상 DOM**을 구현
- **가상 DOM**은 굉장히 빠른 변경 탐지 알고리즘을 이용해 실제 DOM을 효과적으로 갱신
- **가상 DOM**은 뛰어난 성능을 발휘하지만 그보다 더 중요한 것은 **정적인 멘탈 모델**을 제공