**1. http통신과 소켓통신은 어떻게 틀린가.**

**- HTTP통신이란?**

일반적으로 http 통신은Client의 요청(Request)이 있을 때만 서버가(Response)하여 해당 정보를 전송하고 바로 연결을 종료하는 방식이다.

따라서 server가 client로 요청을 보낼 수 없다. 또한, http 통신은 주로 요청을 보내 데이터를 가져오기 위한 안드로이드나 윈도우 응용 프로그램의 rest api를 만들때 사용된다.

실시간 연결이 아니고 필요한 경우에만 요청을 보내는 상황에 유용하게 사용할 수 있다.

즉, http통신은 기본적으로 단방향 통신이다.

**- Socket통신이란?**

소켓은 무엇을 연결할 수 있는 기구나 장치를 말한다.

Http 통신과는 달리 Server와 Client가 특정 포트를 통해 연결을 성립하고 있어 실시간으로 양방향 통신을 하는 방식이다.

보통 Client와 Client 사이의 정보를 주고받는 게임 같은 경우에는 server hub를 이용해 데이터를 hub에 거친 후 상대방에게 다시 데이터를 보내주어 Client <-> Client 데이터 교환과 같이 만든다.

그렇다면 웹 멀티 게임은 어떤식으로 통신을 하는지 궁금해집니다.

Http로 게임 클라이언트 페이지를 로드한 뒤 소켓 서버를 통해 양방향 통신을 하여 멀티 플레이 게임을 즐깁니다.

**-차이점**

Http는 URL을 이용하여 요청을 보낸다. <-> Socket는 아이피와 포트의 주소에 바로 요청을 보낸다.

Http는 단방향 통신이다. <-> Socket 는 양방향 통신이다.

Http는 데이터를 불러오는데 자주 쓰인다 <-> Socket은 데이터를 교환하는데 자주 쓰인다.

물론, HTTP 통신으로 양방향 통신을 구축 할 수 있다.

A클라이언트의 캐릭터 좌표 요청(플레이어 좌표정보) -> 서버 -> 플레이어좌표 정보를 상대방 클라이언트에 던져주고 A클라이언트는 요청의 응답으로 상대방 플레이어의 좌표 정보를 불러온다.

출처

<https://blog-ju0.tistory.com/168>

**2. 컴포넌트의 각 생명주기는 어떻게 되고 각 시점은 언제 호출되는가?**

아래와 같이 3가지의 시점으로 나뉩니다. ( 마운트, 업데이트, 언마운트 )



**1. 마운트**

리액트 컴포넌트가 Instance로 생성되어 DOM tree에 삽입되어 브라우저 상에 나타나는것을 마운트라고 합니다.

**마운트시 호출하는 메서드**

**constructor**

컴포넌트를 새로 만들 때 마다 호출되는 클래스 생성자 메서드입니다.

constructor는 this.state의 초기값 적용, 인스턴스에 이벤트 처리 메서드를 바인딩 하기 위해 사용합니다.



메서드를 바인딩하거나 state를 초기화하는 작업이 없다면, 해당 React 컴포넌트에는 생성자를 구현하지 않아도 됩니다.

React 컴포넌트의 생성자는 해당 컴포넌트가 마운트되기 전에 호출됩니다. React.Component를 상속한 컴포넌트의 생성자를 구현할 때에는 다른 구문에 앞서 super(props)를 호출해야 합니다. 그렇지 않으면 this.props가 생성자 내에서 정의되지 않아 버그로 이어질 수 있습니다.

React에서 생성자는 보통 아래의 두 가지 목적을 위하여 사용됩니다:

this.state에 객체를 할당하여 지역 state를 초기화

인스턴스에 이벤트 처리 메서드를 바인딩

constructor() 내부에서 setState()를 호출하면 안 됩니다. 컴포넌트에 지역 state가 필요하다면 생성자 내에서 this.state에 초기 state 값을 할당하면 됩니다.



생성자는 this.state를 직접 할당할 수 있는 유일한 곳입니다. 그 외의 메서드에서는 this.setState()를 사용해야 합니다.

생성자 내에서는 부수 효과를 발생시키거나 구독 작업(subscription)을 수행하면 안 됩니다. 해당 경우에는 componentDidMount()를 대신 사용하세요



이것은 불필요한 작업이며(this.props.color를 직접 사용하면 됩니다), 버그를 발생시킵니다(color props의 값이 변하더라도 state에 반영되지 않습니다).

**static getDerivedStateFromProps() (잘 사용하지 않는다. )**



getDerivedStateFromProps()는 최초 마운트 시와 갱신 시 모두에서 render() 메서드를 호출하기 직전에 호출됩니다. state를 갱신하기 위한 객체를 반환하거나, null을 반환하여 아무 것도 갱신하지 않을 수 있습니다.

이 메서드는 시간이 흐름에 따라 변하는 props에 state가 의존하는 아주 드문 사용례를 위하여 존재합니다. 예를 들어, 무엇을 움직이도록 만들지 결정하기 위하여 이전과 현재의 자식 엘리먼트를 비교하는 <Transition>와 같은 컴포넌트를 구현할 때에 편리하게 사용할 수 있습니다.

state를 끌어오면 코드가 장황해지고, 이로 인하여 컴포넌트를 이해하기 어려워집니다. 보다 간단한 다른 대안들에 익숙해지는 것을 권장합니다.

**render()**

render() 메서드는 클래스 컴포넌트에서 반드시 구현돼야하는 유일한 메서드입니다.

이 메서드가 호출되면 this.props와 this.state의 값을 활용하여 아래의 것 중 하나를 반환해야 합니다.



render() 함수는 순수해야 합니다. 즉, 컴포넌트의 state를 변경하지 않고, 호출될 때마다 동일한 결과를 반환해야 하며, 브라우저와 직접적으로 상호작용을 하지 않습니다.

브라우저와 상호작용하는 작업이 필요하다면, 해당 작업을 componentDidMount()이나 다른 생명주기 메서드 내에서 수행하세요. render()를 순수하게 유지하여야 컴포넌트의 동작을 이해하기 쉽습니다.



**componentDidMount()**

componentDidMount()는 컴포넌트가 마운트된 직후, 즉 트리에 삽입된 직후에 호출됩니다. DOM 노드가 있어야 하는 초기화 작업은 이 메서드에서 이루어지면 됩니다. 외부에서 데이터를 불러와야 한다면, 네트워크 요청을 보내기 적절한 위치입니다.

이 메서드는 데이터 구독을 설정하기 좋은 위치입니다. 데이터 구독이 이루어졌다면, componentWillUnmount()에서 구독 해제 작업을 반드시 수행하기 바랍니다.

componentDidMount()에서 즉시 setState()를 호출하는 경우도 있습니다. 이로 인하여 추가적인 렌더링이 발생하지만, 브라우저가 화면을 갱신하기 전에 이루어질 것입니다. 이 경우 render()가 두 번 호출되지만, 사용자는 그 중간 과정을 볼 수 없을 것입니다. 이런 사용 방식은 성능 문제로 이어지기 쉬우므로 주의가 필요합니다. 대부분의 경우, 앞의 방식을 대신하여 constructor() 메서드에서 초기 state를 할당할 수 있습니다. 하지만 모달(Modal) 또는 툴팁과 같이 렌더링에 앞서 DOM 노드의 크기나 위치를 먼저 측정해야 하는 경우 이러한 방식이 필요할 수 있습니다.

**2. 업데이트**

리액트 컴포넌트가 새로 갱신되는것 ( 업데이트 ) 시점은 다음의 4가지 입니다.

1. props가 바뀔때

2. state가 바뀔때

3. 부모 컴포넌트가 리렌더링 될 때

4. 강제로 트리거를 발생시킬 때 ( this.forceUpdate )

업데이트시 호출하는 메서드

**static getDerivedStateFromProps**

getDerivedStateFromProps()는 최초 마운트 시와 갱신 시 모두에서 render() 메서드를 호출하기 직전에 호출됩니다. state를 갱신하기 위한 객체를 반환하거나, null을 반환하여 아무 것도 갱신하지 않을 수 있습니다.

이 메서드는 시간이 흐름에 따라 변하는 props에 state가 의존하는 아주 드문 사용례를 위하여 존재합니다

**shouldComponentUpdate**

컴포넌트가 다시렌더링을 해야 할지 말아야 할지 결정하는 메서드입니다. 초기렌더링 또는 forceUpdate()호출시에 이메서드는 호출되지 않습니다. 이메서드는 렌더링을 방지하여 성능을 최적화하는 목적으로 사용됩니다.

**render**

render() 메서드는 클래스 컴포넌트에서 반드시 구현돼야하는 유일한 메서드입니다.

**getSnapshotBeforeUpdate**

render메서드 호출후 DOM 변화를 반영하기 직전에 호출되는 메서드입니다. 이메서드에서 return하는 값을 componentDidUpdate에서 3번째 파라미터로 받아올 수 있습니다.

**componentDidUpdate**

리렌더링을 완료한 후 실행되는 메서드입니다. 최초렌더링에서는 호출되지 않습니다. 컴포넌트가 업데이트 되었을시에 DOM을 조작하기 위해 사용합니다.

컴포넌트가 갱신되었을 때 DOM을 조작하기 위하여 이 메서드를 활용하면 좋습니다. 또한, 이전과 현재의 props를 비교하여 네트워크 요청을 보내는 작업도 이 메서드에서 이루어지면 됩니다 (가령, props가 변하지 않았다면 네트워크 요청을 보낼 필요가 없습니다).



**3. 언마운트**

리액트 컴포넌트가 DOM상에서 제거되는것을 언마운트라고 합니다.

**언마운트시 호출하는 메서드**

**componentWillUnmount**

componentWillUnmount()는 컴포넌트가 마운트 해제되어 제거되기 직전에 호출됩니다. 이 메서드 내에서 타이머 제거, 네트워크 요청 취소, componentDidMount() 내에서 생성된 구독 해제 등 필요한 모든 정리 작업을 수행하세요.

이제 컴포넌트는 다시 렌더링되지 않으므로, componentWillUnmount() 내에서 setState()를 호출하면 안 됩니다. 컴포넌트 인스턴스가 마운트 해제되고 나면, 절대로 다시 마운트되지 않습니다.

**출처**

<https://ko.reactjs.org/docs/react-component.html>

<https://medium.com/humanscape-tech/react-%EC%BB%B4%ED%8F%AC%EB%84%8C%ED%8A%B8-%EC%83%9D%EB%AA%85%EC%A3%BC%EA%B8%B0-c7f45ef2d0be>

**3. 자바스크립트에서 사용하는 자료구조에는 어떤것들이 있는가?**

**Stack**

LIFO(Last In, First Out) 구조입니다.

연결 리스트인데 뒤로 넣고 뒤로만 뺄 수 있습니다. 앞으로는 넣지도, 빼지도 못합니다. 쉽게 생각하면 자바스크립트 배열인데 shift, unshift 없이 push와 pop만 있다고 생각하시면 됩니다

사실 push와 pop이라는 메소드 이름이 스택에서 나온 겁니다. 또 한 가지의 메소드는 stackTop으로 스택의 마지막 요소를 알려주는 겁니다. 스택의 마지막 요소를 top이라고 부르기도 합니다.



**Queue**

FIFO(First In, First Out) 구조입니다.

enqueue, dequeue

새로 온 사람은 줄 맨 뒤에 서고, 제일 앞 사람은 필요한 행동을 한 후 빠집니다. 이렇게 뒤에서 들어가고(enqueue) 앞에서 빠지는(dequeue) 구조입니다. 자바스크립트의 배열로 따지면 push(enqueue)와 shift(dequeue) 메소드만 있는 거라고 생각하시면 됩니다. 거기에 추가로 제일 앞의 데이터를 알 수 있는 front가 있습니다.

**Linked List**

연결 리스트는 여러 개의 노드로 이루어져 있습니다. 각각의 노드는 데이터와 다음 노드가 뭔지 알려주는 주소를 가지고 있습니다. 또한 연결 리스트는 새로운 데이터를 추가하거나, 데이터의 위치를 찾거나, 제거하는 기능이 있어야 합니다.

1->2->3->4->5 라는 연결 리스트가 있다면 1,2,3,4,5는 데이터고 ->는 주소입니다. 안타깝게도, 이미 자바스크립트에는 이것이 구현되어 있습니다. 바로 배열입니다. [1,2,3,4,5]가 있으면 1,2,3,4,5는 데이터이고, array[0], array[1] 등은 데이터가 담긴 위치를 말해주고 있죠. 또한, array.push();를 통해 데이터를 추가할 수 있고, array.splice();를 통해 데이터를 제거할 수 있습니다.

하지만 배열과는 다릅니다.

**(배열에 비해)데이터의 추가/삽입 및 삭제가 용이합니다.**

배열은 데이터를 중간에 삽입하면, 다른 데이터들의 위치를 모두 바꿔줘야하기때문에 O(n)의 시간이 소요됩니다.

반면 링크드리스트는 데이터를 중간에 삽입하면 링크의 연결만 조정해주면 되기때문에 O(1)의 시간이 소요됩니다.

**(배열에 비해)데이터의 검색 속도가 떨어집니다.**

배열에서 인덱스만 알면 값을 읽는데 O(1)이 소요되는 반면, 링크드리스트는 각 노드가 다음 노드의 주소만을 가지고 있기 때문에 최악의 경우 모든 요소를 순회하는 O(n)이 소요됩니다.

**출처**

<https://www.zerocho.com/category/Algorithm/post/58008a628475ed00152d6c4d>