타입스크립트 #4 this

서론 : 이번 강의가 마지막 강의가 될 듯 합니다. 여기까지 따라오느라 고생하신 여러분 진심으로 수고 많으셨습니다.

This ? :

This 키워드는 함수가 호출 될 때 지정되는 변수입니다. 꽤나 어려운 내용 덕분에 바로 따라가기란 어려울 것으로 예상되니, this키워드에 대한 설명과 함께 타입스크립트의 this 설명을 이어 나가도록 하겠습니다.

This를 이것이다 ! 라고 일률 적으로 설명하기는 어렵습니다. 왜냐하면 this는 arrow function 에서의 this와 function키워드를 사용하는 함수에서 가리키는 this가 다르기 때문입니다. 다음과 같은 예제를 함께 보도록 하겠습니다.

const dummy = {

declaration : function(){

console.log(this);

},

arrow : () =>{

console.log(this)

}

}

dummy.declaration();

dummy.arrow();

declation 메서드와 arrow 메서드는 둘다, this키워드를 출력하도록 되어 있습니다. 두개의 함수는 모양만 봐서는 차이가 존재하지 않습니다. 이 두개의 this는 과연 같은 this를 가리키게 될까요? 정답은 X입니다. 함수 선언식으로 작성된 함수는 dummy객체를 가리키고 있고, 함수 표현식으로 작성된 함수는 빈 객체를 가리키고 있습니다. 실은 이 객체는 globalThis 키워드 같은 것이 아니라, module.exports를 가리키고 있습니다. 한번 확인 해볼까요 ?

expression : () =>{

console.log(this);

console.log(this === exports)

}

결과는 놀랍게도 true가 나옵니다. 이번 강의에서는 node.js의 내부를 뜯어 보는 것이 아니기 때문에 이런게 있구나라고 알고 컴공 수업시간에 갑작스레 손을 들어 아는 척을 해보도록 합시다.

그렇다면 이 두 함수 작성법에 따른 this는 어떻게 지정되는 가를 한번 알아보도록 합시다.

먼저 함수 선언식의 this키워드는 “이 함수를 호출 하는 객체를 this 키워드에 할당 하게 됩니다. ”

const dummy = {

declaration : function(){

console.log(this);

}

}

const declaration = dummy.declaration;

declaration();

예제를 통해서 알아봅시다. 다음과 같이 dummy객체를 변경하고 dummy객체의 declaration함수를 전역에 할당 한 후 호출 하면, global 객체가 this에 할당 된 것을 알 수 있습니다. 조금 더 예제를 변경 해보도록 하겠습니다 .

Dummy\_2 객체를 새로 만들고 dummy객체와 동일하게 declaration을 만들고 이에 dummy의 declarartion함수를 할당 해주도록 하겠습니다.

const dummy = {

declaration : function(){

console.log(this);

}

}

const dummy\_2 = {

name : "dummy2",

declaration : dummy.declaration

}

dummy\_2.declaration()

전역 스코프에서 declaration함수를 변수에 할당 한 후 호출 한 것과 같이 “함수를 호출 하는 객체”를 this 키워드에 할당 하는 모습을 볼 수 있습니다.

그렇다면 이번에는 화살표 함수로 작성된 함수에서 this키워드는 무엇을 가리키고 있는 것일까요 ? 바로 “상위 스코프의 this” 이를 영어로 lexical scope라고 합니다.

다음과 같은 예제를 작성 하도록 하겠습니다.

function Factory(name){

function Method(){

return () => console.log(this.name)

}

return {

name,

Method

}

}

const dummy = new Factory("lee");

const dummy\_2 = dummy.Method()

dummy\_2()

function 키워드를 사용한 함수 선언식과 같이 전역스코프에 있는 dummy\_2 변수에 sayMyname을 할당하였지만, dummy 변수의 name을 그대로 사용하는 모습을 볼 수 있습니다. 그래서 정리를 하자면 Function키워드를 통하여 작성된 함수 내에서 this는 “실행 컨텍스트에 따라 this가 동적으로 할당된다.” Arrow function을 사용된 함수 내에서의 this는 “작성된 객체의 상위 스코프의 this를 정적으로 할당한다” 입니다. 그래서 보통 react를 사용 할 때 function 키워드를 사용한 component 메서드를 써 element에 이벤트 콜백으로 사용하기 보다, 정적으로 this가 bind된 arrow function을 주로 사용 합니다.

Typescript this 사용법 :

위의 예제와 비슷한 다음의 상황을 보도록 하겠습니다.

let deck = {

suits: ["hearts", "spades", "clubs", "diamonds"],

cards: Array(52),

createCardPicker: function() {

return function() {

let pickedCard = Math.floor(Math.random() \* 52);

let pickedSuit = Math.floor(pickedCard / 13);

return {suit: this.suits[pickedSuit], card: pickedCard % 13};

}

}

}

let cardPicker = deck.createCardPicker();

let pickedCard = cardPicker();

alert("card: " + pickedCard.card + " of " + pickedCard.suit);

이 예제는 오류를 발생 시킵니다.그 이유는 앞서 봤던 것처럼 createCardPicker가 return 하는 함수가 전역 스코프를 가지고 있는 변수에 할당 되면서, this가 window객체에 설정되었기 때문입니다. 즉 호출 하는 위치가 전역으로 설정되었기 때문에, 발생한 일입니다. 그럼 this를 deck 객체에 바인딩 하려면, 호출하는 위치에서 동적으로 this가 바인딩 되지 않는 화살표 함수를 사용하면 되겠지요 ?

그러나, this변수가 타입이 지정되지 않은 any 상태이기 때문에, this에 shits를 사용 할 수 없다고 에러를 발생합니다. 즉 this의 타입을 지정해주면 됩니다.

interface Card {

suit: string;

card: number;

}

interface Deck {

suits: string[];

cards: number[];

createCardPicker(this: Deck): () => Card;

}

let deck: Deck = {

suits: ["hearts", "spades", "clubs", "diamonds"],

cards: Array(52),

// NOTE: 아래 함수는 이제 callee가 반드시 Deck 타입이어야 함을 명시적으로 지정합니다.

createCardPicker: function(this: Deck) {

return () => {

let pickedCard = Math.floor(Math.random() \* 52);

let pickedSuit = Math.floor(pickedCard / 13);

return {suit: this.suits[pickedSuit], card: pickedCard % 13};

}

}

}

let cardPicker = deck.createCardPicker();

let pickedCard = cardPicker();

alert("card: " + pickedCard.card + " of " + pickedCard.suit);

지금 보면 약간 특이한 상태를 관찰 할 수 있습니다. This가 createCardPicker의 매개변수로 지정되어 있다는 점입니다. 실제로 this는 함수가 호출 될 때 arguments와 this가 같이 전달인자로 들어오게 됩니다. 굳이 javascript에서 this를 매개변수로 지정하면서 작성할 일은 없으니, 참고 하시면 됩니다.

**Callback 사용법 :**

나중에 콜백 함수를 사용하는 라이브러리의 이벤트 핸들러에 전달 할 때 this 때문에 오류가 발생할 수 있습니다. 왜냐하면, 라이브러리는 콜백을 일반 함수 처럼 callback(event) 형식으로 호출 하기 때문에, this는 undefined상태가 됩니다. 그렇기 때문에 라이브러리 작성자는 다음과 같이 이벤트 핸들러의 인터페이스를 작성하여, this가 undefined되는 상태를 피하려고 합니다.

interface UIElement {

addClickListener(onclick: (this: void, e: Event) => void): void;

}

이게 무슨 말인지, 어려운 사람들이 많을 겁니다. 쉽게 풀이하면 라이브러리 제작사는 여러분의 this가 어떤 타입인지 모릅니다. 그렇기 때문에, 라이브러리 입장에서는 this를 사용할 때 혼돈을 피하고자, void로 타입을 지정 합니다.

그럼 this를 사용해야 하는 상황을 한번 볼까요 ?

class Handler {

info: string;

onClickBad(this: Handler, e: Event) {

// 이런, `this`가 여기서 쓰이는군요. 이 콜백을 쓰면 런타임에서 충돌을 일으키겠군요

this.info = e.message;

}

}

let h = new Handler();

uiElement.addClickListener(h.onClickBad); // 오류!

Handler클래스는 uiElement에서 click이벤트가 발생할 때 onClickBad라는 콜백이 호출 되길 원합니다. 또한 event객체의 message프로퍼티를 Handler 인스턴스의 info 프로퍼티에 할당하기를 원합니다. 그러나 이를 UIelement에 Handler에 전달하려고 하니, Interface에 정의된 콜백의 this가 void타입이라, 전달인자 타입 에러를 발생 시킵니다. 에러를 발생시키지 말게 하려면 인터페이스와 동일하게 매개변수로 전달되는 this를 void타입으로 지정해야 합니다.

class Handler {

info: string;

onClickGood(this: void, e: Event) {

// void 타입이기 때문에 this는 이곳에서 쓸 수 없습니다!

console.log('clicked!');

}

}

let h = new Handler();

uiElement.addClickListener(h.onClickGood);

이와 같이 말이죠. 그러나 이번 경우에는 this를 사용하지 못합니다. 이 this를 사용 하고자 한다면, 정적으로 this가 해당 인스턴스에 묶여 있는 화살표 함수를 사용 해야 합니다. 이처럼 말이죠 .

class Handler {

info: string;

onClickGood = (e: Event) => { this.info = e.message }

}