Typescript This

Taken from an older version of the Typescript handbook, as the latest does not include this, but is still relevant.

This.

Learning how to use this in JavaScript is something of a rite of passage. Since TypeScript is a superset of JavaScript, TypeScript developers also need to learn how to use this and how to spot when it's not being used correctly. Fortunately, TypeScript lets you catch incorrect uses of this with a couple of techniques. If you need to learn how this works in JavaScript, though, first read Yehuda Katz's Understanding JavaScript Function Invocation and "this". Yehuda's article explains the inner workings of this very well, so we'll just cover the basics here.

this and arrow functions

In JavaScript, this is a variable that's set when a function is called. This makes it a very powerful and flexible feature, but it comes at the cost of always having to know about the context that a function is executing in. This is notoriously confusing, especially when returning a function or passing a function as an argument.

Let's look at an example:

```
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);
```

Notice that createCardPicker is a function that itself returns a function. If we tried to run the example, we would get an error instead of the expected alert box. This is because the this being used in the function created by createCardPicker will be set to window instead of our deck object. That’s because we call cardPicker() on its own. A top-level non-method syntax call like this will use window for this. (Note: under strict mode, this will be undefined rather than window).

We can fix this by making sure the function is bound to the correct this before we return the function to be used later. This way, regardless of how it’s later used, it will still be able to see the original deck object. To do this, we change the function expression to use the ECMAScript 6 arrow syntax. Arrow functions capture the this where the function is created rather than where it is invoked:

```
let deck = {
  suits: ["hearts", "spades", "clubs", "diamonds"],
  cards: Array(52),
  createCardPicker: function () {
    // NOTE: the line below is now an arrow function, allowing us to capture 'this' right here
    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);
```

Notice. In the above example, the function defined inside `createCardPicker` was changed to a lambda, giving it this syntax: `return () => {}`

This subtle change, tells the compiler to inject the ‘this’ context into the lambda.

Takeaway

So. Whenever you come across a spot in your code, where you have a callback method that uses ‘this’, but the this is not the same as the object where the callback resides, then the this context has inadvertently been swapped out. You can fix the problem with the same technique as above.

In addition, TypeScript will warn you when you make this mistake if you pass the noImplicitThis flag to the compiler. It will point out that this in this.suits[pickedSuit] is of type any.

Revision #2
Created 22 March 2025 23:23:41 by glwhite
Updated 22 March 2025 23:24:49 by glwhite