CS472 WAP

Closures

Scope, Closures, and Encapsulation

Maharishi International University Fairfield, Iowa © 2020

Main Point Preview

- Closures are created whenever an inner function with free variables is returned or assigned as a callback. Closures provide encapsulation of methods and data. Encapsulation promotes self-sufficiency, stability, and re-usability.

- *Science of Consciousness: Closures provide a supportive wrapper for actions that will occur in another context. Transcendental consciousness provides a supportive wrapper for our actions that will occur outside of meditation.*

Calling an inner function

```
function init() { //function declaration
  const name = "Mozilla";
  function displayName() {
    console.log(name);
  }
  displayName();
}
init();
```

- like in Java, nested functions have access to variables declared in their outer scope

Returning an inner function

```
function makeFunc() {
  const name = "Mozilla"; //local to makeFunc
  function displayName() {
    console.log(name);
  }
  return displayName;
}
const myFunc = makeFunc();
myFunc();
```

- Q:is the local variable still accessible by myFunc?
- A: yes. Example of saving local state inside a JavaScript closure.

Master the JavaScript Interview: What is a Closure?

- “most competent interviewers will ask you what a closure is, and most of the time, getting the answer wrong will cost you the job.”
  - “Be prepared for a quick follow-up: Can you name two common uses for closures?”
- first and last question in my JavaScript interviews.
  - can’t get very far with JavaScript without learning about closures.
- You can muck around a bit, but will you really understand how to build a serious JavaScript application? Will you really understand what is going on? Not knowing the answer to this question is a **serious red flag**.
- Not only should you know the mechanics of what a closure is,
  - you should know why it matters,
  - should know several possible use-cases for closures.

Closures

- closure: A first-class function that binds to free variables that are defined in its execution environment.

- free variable: A variable referred to by a function that is not one of its parameters or local variables.
  - bound variable: A free variable that is given a fixed value when "closed over" by a function's environment.

- A closure is defined when a(n inner) function is defined that has free variables
  - a closure instance is created when the inner function is returned or assigned to a variable and it attaches itself to the free variables from the surrounding environment to "close" up those stray references.

Closures in JS

```
const x = 1;

function f() {
    let y = 2;
    const sum = function() {
        const z = 3;
        console.log(x + y + z);
    }
    y = 10;
    return sum;
} //end of f

const g = f();
g();
```

- inner function closes over free variables when it is returned
  - Saves references to the names, not values

Common closure bug with fix

```
var funcs = [];
for (var i = 0; i < 5; i++) {
  funcs[i] = function() {
    return i;
  };
}
console.log(funcs[0]());
console.log(funcs[1]());
console.log(funcs[2]());
console.log(funcs[3]());
console.log(funcs[4]());
```

- Closures that bind a loop variable often have this bug.
- Why do all of the functions return 5?

```
/* return a function with no parameters
 that has an ‘embedded parameter’ */
var helper = function(n) {
  return function() {return n;}
}

var funcs = [];
for (var i = 0; i < 5; i++) {
  funcs[i] = helper(i);
};
console.log(funcs[0]());
console.log(funcs[1]());
console.log(funcs[2]());
console.log(funcs[3]());
console.log(funcs[4]());
```

Common closure bug with fix (ES6)

//buggy version with var

```
var funcs = [];
for (var i = 0; i < 5; i++) {
  funcs[i] = function() {
    return i;
  };
}
```

```
//ES6 solution:  let vs var
const funcs = [];
for (let i = 0; i < 5; i++) {
  funcs[i] = function() {
    return i;
  };
}
```

```
console.log(funcs[0]());
console.log(funcs[1]());
console.log(funcs[2]());
console.log(funcs[3]());
console.log(funcs[4]());
```

Practical uses of closures

- A closure lets you associate some data (the environment) with a function
  - parallel to properties and methods in OOP.
- Consequently, use a closure anywhere you might use an object with a single method.
  - objects have properties to capture state info
  - JavaScript closures capture state info by saving references to free variables
- Situations like this are common on the web.
  - an event handler is a single function executed in response to an event.
    - e.g., DOM and timer event handlers
      - .. in 30 seconds print out whatever is in the currentQuestion variable
    - E.g., factory function that sets state information in reusable code (next slide)
  - closures for encapsulation and namespace protection (module pattern)
- Event handlers must be functions without parameters
  - If you need to pass parameter information with an event handler
    - callback with no parameters but include free variables from the lexical environment.
    - JavaScript engine will create closure over bound variables when assign callback to event handler.

Function factory with closures

example of closures being helpful with event handling

```
<a href="#" id="size-12">Size 12</a>
<a href="#" id="size-16">Size 16</a>
<a href="#" id="size-18">Size 18</a>

function makeSizer(size) {
  return function() {
    document.body.style.fontSize = size + "px";
  };
}
document.getElementById("size-12").onclick = makeSizer(12);
document.getElementById("size-16").onclick = makeSizer(16);
document.getElementById("size-18").onclick = makeSizer(18);

//what is the free variable?
//why is the closure necessary?
```

Function factory with closures (cont)

- Have a function that sets the fontsize, and want to have some state info (about the environment)
  - state info is the font size associated with different buttons
  - normally could make this a parameter,
  - but must add parameter without executing the function
  - also, the click event will not pass any parameters to the callback function
  - hence, if want to save some state info along with the function, the common way to do it in JS is to use a closure because it creates a function and can save enclosing state info in the ‘free’ variables

Main Point

- Closures are created whenever an inner function with free variables is returned or assigned as a callback. Closures provide encapsulation of methods and data. Encapsulation promotes self-sufficiency, stability, and re-usability.

- *Science of Consciousness: Closures provide a supportive wrapper for actions that will occur in another context. Transcendental consciousness provides a supportive wrapper for our actions that will occur outside of meditation.*

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Life Is Found in Layers

1. JavaScript is a functional OO language that has a shared global namespace for each page and local scope within functions.
2. Closures provide a lexical scoping mechanism for JavaScript inner functions. Let and const provide this for blocks of ES6 code. These mechanisms promote encapsulation, layering, and abstraction in code.

---

3. **Transcendental consciousness** is the experience of the most fundamental layer of all existence, pure consciousness, the experience of one's own Self.
4. **Impulses within the transcendental field:** The many layers of abstraction required for sophisticated JavaScript implementations will be most successful if they arise from a solid basis of thought that is supported by all the laws of nature.
5. **Wholeness moving within itself:** In unity consciousness, one appreciates that all complex systems are ultimately compositions of pure consciousness, one's own Self.

MAHARISHI INTERNATIONAL UNIVERSITY

HIGHER EDUCATION FOR HIGHER CONSCIOUSNESS