Class 2022/09/06

Node and Json

前面还有一部分没记

- Functions are just an objects. A way to create a function is to use the `function` key word and assign it to a variable.

```
const f = function(x) {
    return x + 1
}

console.log(typeof f)
// will print 'function'
```

```
const arr = [1, 2, 3]

arr.forEach(function(x) { console.log(x + 1) })
// will print 2 3 4
```

The part `function(x) { console.log(x + 1) }` is an example of an anonymous function.

Hoisting

- Hoisting （变量提升）：

  When you declare a function using the keyword function and give it a name immediately, the function declaration and its body was actually brought to the top of the file. That means you can do this:

```
printInc(3)

function printInc(x) {
    console.log(x + 1)
}
// will print 4
```

- 变量提升也适用于其他数据类型和变量。变量可以在声明之前进行初始化和使用。但是如果没有初始化，就不能使用它们。
- 函数和变量相比，会被优先提升。这意味着函数会被提升到更靠前的位置。
- JavaScript 只会提升声明，不会提升其初始化。如果一个变量先被使用再被声明和赋值的话，使用时的值是 undefined。

Ways of declaring functions

- Hoisting won't work if you use `const` to define a function, because `const` and `let` are not hoisted.

```
inc(3)

const inc = function(x) {
    console.log(x + 1)
}
// error
```

- We can also use arrow to declare functions, in the format `parameters => body`. There's an implicit return, it'll automatically return the result of the body, so we don't need to add a return statement.

```
//this is an anonymous function
x => console.log(x + 1)

//giving it a name
const inc = x => console.log(x + 1)

arr.forEach(inc) // will print 2 3 4
```

  We can also accept multiple parameters

```
const add = (a, b) => a + b
console.log(add(5, 5)) // prints 10
```

  If we want multiple lines of body, we need to put them in curly braces. Also we need to add a return.

```
const add = (a, b) => {
    return a + b
}
console.log(add(5, 5)) // prints 10
```

  - Advantages:
    - No hoisting
    - More terse (concise), shorter to write
  - Side effects:
    - don't have `this`
  - Don't use arrow functions as event handlers.
  - Don't use when creating methods.
- Summary: Ways of declaring functions:

```javascript
// 1. Function declaration
function f() {

}

// 2. Constant declaring and set it equal to a function expression
const f = ......(one of the following two)
    //arrow function
const f = x => x+1
    //anonymous function
const f = function(x) {return x+1}

// 3. Anonymous functions
```

Event Handler

- What is an event handler?
    - Node.js is a "server" side JavaScript -- runtime + framework(networking, file io, process management)
    - it's different from a browser JavaScript
        - Document (html)
        - worker management
        - local storage in browser
    - Typically, in node your app is a "single process" (blocking)
        - All io is non-blocking / Asynchronous
        -
          ```javascript
          const request = require('request');
          console.log("Start");
          request('http://www.google.com', function (error, response, body) {
              //just print out the first 30 characters of the response body
              console.log(body.slice(0,30))
          })
          console.log("Done!");
          // Will print "Start" and "Done" first, then the html
          ```
        - the function is a callback function.

          `function (error,response,body) {`
          `    console.log(body.slice(0,30))`
          `}`

          This means that it's not immediately invoked.

          it's called when the request function finishes.
    - io:
        - reading a file
        - writing a file
        - read/write from a database
        - requesting from network
    - These are all asynchronous, because we don't know how long it's going to take, so we just continue to the next line.
- Therefore, we need an event handler to let us know when the io (reading or writing is finished)

- In this case,

  ```
  function (error,response,body) {
      console.log(body.slice(0,30))
  }
  ```

- BTW, another way of dealing with async in io is the function `async` and `await`.
  - `await` do the block for you, and the rest of you app can still run.
  - One of the older ways

Functions as parameters

- Functions as parameters
  - Example:

    ```
    const arr = [1,2,3]

    arr.forEach(console.log) //prints 一串很怪的东西
    // Note that console.log() is also a call back function here

    arr.map(x => x * 2) //returns [2,4,6]
    ```

  - Note that we can pass functions as arguments, either using already defined functions or anonymous functions.

Default value of parameter

- Set default value in function declaration
  - `foo(a, b, c='default'){}`
  - If you don't pass enough parameters in JavaScript, it'll just treat the rest as undefined.

Spread operator

- Array methods
  - The `...` operator
  - rest operator
    - It'll consider the arguments as an array

    ```
    function foo(...args) {
        console.log(args)
    }
    foo(1,2,3)      //prints [1,2,3]
    foo(1,2,3,4,5)  //prints [1,2,3,4,5]
    ```

  - spread operator
    - Takes elements of an array and puts it as positional arguments
    - `parseInt('number', radix)`

```
parseInt('100')             // 100
parseInt('100', 2)          // 4

const arr = ['100', 2]

parseInt(arr)               // 100
parseInt(arr[0],arr[1])     // 4
parseInt(...arr)            // 4
// ... is the spread operator
```

- Also works with arrays.
- Instead of `concat` we can use it to combine two arrays.

```
const arr = ['100', 2]
const arr2 = [300,400]

console.log([...arr, ...arr2])
//['100', 2, 300, 400]

console.log(['wat', ...arr, ...arr2])
//['wat','100', 2, 300, 400]
```

Packages and Modules

NPM

- npm:
  - The actual command line tool to install packages / modules
  - the repository where these modules are listed
- alternatives to npm
  - yarn
  - pnpm
- `package.json` = meta info (project name, version, dependencies, dev deps, etc.)
  - generated on npm install
  - generated on npm init
- `package-lock.json` = exact tree of deps for your project
  - generated on npm install
- `dependencies` and `devDependencies` difference
  - `devDependencies` are things that your app doesn't depend on, but your development process does.
- npx
- npm allows installation of
  - modules
  - command line tools
  - these are typically server side
- NPM does allow installation of frontend library, but they don't work on server (node context) necessarily.
- 后面没听懂