香港中文大學
The Chinese University of Hong Kong

# CSCI2720 - Building Web Applications

## Lecture 7: Fetch API and Asynchronous JS

Dr Colin Tsang

# Outline

- Asynchrony in JS
- Callback functions
- Promise

- The Fetch API
- AJAX

# Asynchrony in JS

- JS is single-threaded
  - One execution at a time: the *Call Stack* determines what to run next, sequentially.
    - Last-In-First-Out
  - If there is an action which takes a while, every else is blocked.

- Callback queue
  - a.k.a. event queue or message queue.
  - Hold the functions that are scheduled to be executed once certain conditions are met.

- *Event loop* in the JS engine keeps checking if the *call stack* is empty and brings events from the call back queue to the call stack.

https://thecodest.co/blog/asynchronous-and-single-threaded-javascript-meet-the-event-loop/

# Asynchrony in JS

- Asynchronous programming in JS
  - A task is started, but without waiting for it to finish
  - When the task is done, something would happen
    - Events, callbacks, promises,…
  - This is important for I/O which requires waiting
    - Web data submission or retrieval
    - Database execution
  - A precise control of the execution order of step is necessary

# Scheduling calls in JS

- Two methods for executing a function later:
  - **setTimeout()** – run the function after certain time
  - **setInterval()** – run the function repeatedly at interval
  - **clearTimeout()** and **clearInterval()** are the stopping mechanisms

- Possible to nest them for special time settings
- https://javascript.info/settimeout-setinterval

```
setTimeout(() => console.log("hello"), 2000);
// hello appears after 2 seconds
```

# Callback functions

- When a function is passed as an *argument* of another function to be called later, that is a *callback function* (or just a *callback*)
  - It will be called when the calling function is done

*callback*

```
setTimeout(() => console.log("hello"), 2000);
// hello appears after 2 seconds
```

# Callback functions

For old browsers only

- The callback hell: multiple waits are possible by chaining up callbacks, but the code looks bad. (Try arrow function!)

```
> function waitnprint(str, cb){
      setTimeout( function(){
          console.log(str);
          cb();
      }, 2000);
  }

<· undefined
```

```
> waitnprint("hello", function() {
      waitnprint("world", function() {
          waitnprint("!", function(){
              waitnprint("END", function(){});
          })
      })
  })

<· undefined

  hello

  world

  !

  END
```

# Promise

- The **Promise** object represents the results of an asynchronous execution, with 3 state:
  - **pending**: initial state
  - **fulfilled**: task was done -> a result value can be found
  - **rejected**: task failed -> an error object can be found

- Involves a *success* callback and a *failure* callback
  - Both are optional
- The **Promise** object is now widely used by async operations
  - Used via the **then()** method of the promise, which takes two callbacks
- Many old browsers don’t support promise.

# Promise

- The syntax of a Promise:

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time) Slow task
  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

- See: https://www.w3schools.com/js/js_promise.asp

# Promise chain

```
// supporting Promise
function waitnprint(str) {
  return new Promise((resolve, reject) => {
    setTimeout( function() { // ...wait for a while...
      console.log(str);
      resolve();
    }, 1000);
  })
}
```

```
> waitnprint("hello")
  .then(()=>waitnprint("world"))
  .then(()=>waitnprint("!"))
  .then(()=>waitnprint("END"))
  .catch((err)=>console.log(err));

<· ▶ Promise {<pending>}
  hello
  world
  !
  END
```

# Fetch and Finally

- Similar to a chain of try-catch-finally, now the syntax is applicable to promises as well
  - We will talk about *fetch()* in next slides

```
fetch('https://www.google.com')
  .then((response) => {
    console.log(response.status);
  })
  .catch((error) => {
    console.log(error);
  })
  .finally(() => {
    document.querySelector('#spinner').style.display='none';
  });
```

fetch() create a Promise object automatically

if success

if reject

execute it anyway

# The FETCH API

- Without reloading a web page, how can new data be retrieved from the server?
  - Asynchronous data retrieval: **fetch()**
  - **fetch()** returns a Promise object for easy handling

- For security reasons, such async JS data loading by default requires “same origin”, i.e., on the same server/port

- **fetch()** can also be used for submitting data to server

- See: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

# The Fetch API - example

- *async*: declares the function as an asynchronous function, which allow us to use *await*.
- **fetch()**: send a GET request to the URL, which is a JSON file in this case.
- *await*: this keyword is used to pause the execution of the function until the *Promise* returned by **fetch()** is resolved.
- **json()**: this method is called on the response object to extract the JSON data from the HTTP response.

```
async function logMovies() {
  const response = await fetch("http://example.com/movies.json");
  const movies = await response.json();
  console.log(movies);
}
```

- Other type of data:
  - **text()**, **formData()**, **arrayBuffer()**,…

# AJAX

- Some years ago, most of async data retrieval was don't with *AJAX* (asynchronous JavaScript and XML), using an *XMLHttpRequest* object, with the help of *jQuery*.
- Nowadays **fetch()** becomes a more prominent way thanks to the simplicity with syntax.

- There are subtle differences between the two options.

# Further readings

- MDN:
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

- A guide writing asynchronous JS programs:
- http://callbackhell.com/

- Fetch on javascript.info:
- https://javascript.info/fetch