CMSC335

Web Application Development with JavaScript

Promises

Department of Computer Science

University of MD, College Park

Slides material developed by Ilchul Yoon, Nelson Padua-Perez

Asynchronous Programming

- Asynchronous functions cannot use **return** to return the result of the asynchronous computation or inform about errors using the approach we use in synchronous code
- To return values and errors, asynchronous functions use a callback function
  - The callback function is called by the asynchronous function to return the result or, in the case of error, an error object
- Structure
  - asyncFunc(callback)
  - callback – function(error, result)
    - » error - error object or **null** if no error occurred
    - » result - the result of the asynchronous computation
- **Example:** AsyncFunc.html
- **Reference:** JavaScript, The Comprehensive Guide, Philip Ackermann, ISBN-13 : 978-1493222865

Pyramid of Doom

- Pyramid of Doom/Callback Hell
  - When another asynchronous function (B) is called in the callback of an asynchronous function (A), and the callback of asynchronous function (B) calls another asynchronous function (C), etc.
  - The more nested, the more the code moves to the right, like a pyramid
- Pyramid of Doom

```
asyncFuncA((error1, result1) => {
    // callbackA code
    asyncFuncB((error2, result2) => {
        // callbackB code
        asyncFuncC((error3, result3) => {
            ....
```

Promises

- Promises - prevent the Pyramid of Doom and provide support for asynchronous programming
- Promise - A promise is an object of the **Promise** type that is a placeholder for the result of an asynchronous function
- The Promise object encapsulates two functions:
  - resolve() - the one used to return the result
  - reject() - the one to notify about errors
- How does the asynchronous function definition change from a function that receives a callback?
  - The asynchronous function that performs a task will not receive a callback; instead, it will return a Promise that has the code for the task
  - The callback(s) is now provided to the promise object using **then()**
- **then()** method - promise method that can receive two callbacks as arguments. The first argument is the callback for handling the result, and the second for handling errors. Notice the order of the callbacks we pass to then() is the result callback, followed by error callback (the reverse of what we saw earlier)
- Usually you will see then() used with the result callback and then a catch() with the error callback
- Notice: **then()** does not block the current method execution!
- **Example:** AsyncFuncUsingPromise.html

Promises

- A promise must return a value using resolve()) or an error (using reject()) otherwise, it will NOT work
- **then()** method returns a promise which allows successive calls to be concatenated (prevents the pyramid of doom)
- Using promises (no pyramid of doom)

```
asyncFuncA()
  .then(result => {
      // code for asyncFuncB
  })
  .then(result => {
      // code for asyncFuncC
  })
  …
```

States of Promises

- **States**
  - **Pending** - waiting for the asynchronous task to finish
  - **Fullfilled/Resolved** - the asynchronous task was completed successfully
  - **Rejected** - the asynchronous task failed
- **settle** - a settled promise is one that is not pending (it has been fulfilled or rejected)
- **Example:** PromiseStates.html
- **Example:** PromiseSqrt.html