JavaScript Promises

Call me maybe

Prologue

The Synchronous Way

```
function syncFunction(x) {
    var y;
    // Huge-ass implementation which uses x and populates y
    if (!y) { throw new Error('Huge-ass function has not populated y'); }
    return y;
};
var a = 'some generic value';
console.log(syncFunction(a));
console.log('YAY!');
```

The Asynchronous Way?

```
function asyncFunction(x) {
    setTimeout(function(x) {
        var y;
        // Huge-ass implementation which uses x and populates y
        if (!y) { throw new Error('Huge-ass function has not populated y'); }
        return y;
     }, 0);
};
var a = 'some generic value';
console.log(asyncFunction(a)); // <- YOUR CODE DOESN'T WAIT HERE
console.log('YAY!');
```

The Asynchronous Way?

```
function asyncFunction(x) {
    setTimeout(function(x) {
        var y;
        // Huge-ass implementation which uses x and populates y
        if (!y) { throw new Error('Huge-ass function has not populated y'); }
                            // ^ WHO IS GOING TO CATCH THIS ERROR??
         return y; // <- WHO IS GOING TO GET THIS VALUE??
    }, 0);
};
var a = 'some generic value';
console.log(asyncFunction(a)); // <- YOUR CODE DOESN'T WAIT HERE
console.log('YAY!');
```

The Asynchronous Way

- Definition

```
function asyncFunction(x, callback) {
    setTimeout(function(x, callback) {
        var y;
        // Huge-ass implementation which uses x and populates y
        if (!y) { callback(new Error('Huge-ass function has not populated y')); }
        else { callback(null, y); }
    }, 0);
};
```

The Asynchronous Way

- Usage

```
var a = ‘some generic value’;
asyncFunction(a, function(error, result) {
     if (error) { /* Handle Error */ }
     else { console.log(result); }
});
console.log(‘YAY!’); // <- YOU’LL SEE THIS FIRST
```

When Going Async

Cons

- Lose Stack/Returns
- Different Implementations can use different callback formats
- Callbacks can be executed multiple times

Pros

- Non Blocking Calls. YAY!

Is that it?

Callback Hell

```
function functionA(x, callback) { … };
function functionB(y, callback) { … };

var a = ‘some generic value’;
functionA(a, function(error, resultOfA) {
     if (error) { /* Handle Error */ }
     else {
          functionB(resultOfA, function(error, result) {
               if (error) { /* Handle Error */ }
               else {
                    console.log(result);
               }
          }
     }
});
```

Callback Hell.

```
function functionA(a, callback) { … };
function functionB(b, callback) { … };
function functionC(c, callback) { … };

var a = ‘some generic value’;
functionA(a, function(error, result) {
      if (error) { /* Handle Error */ }
      else {
            functionB(resultOfA, function(error, result) {
                  if (error) { /* Handle Error */ }
                  else {
                        functionC(resultOfB, function(error, result) {
                              if (error) { /* Handle Error */ }
                              console.log(result);
                        }
                  }
            }
      }
});
```

Callback Hell..

```
function functionA(a, callback) { … };
function functionB(b, callback) { … };
function functionC(c, callback) { … };
function functionD(d, callback) { … };

var a = ‘some generic value’;
functionA(a, function(error, result) {
        if (error) { /* Handle Error */ }
        else {
                functionB(resultOfA, function(error, result) {
                        if (error) { /* Handle Error */ }
                        else {
                                functionC(resultOfB, function(error, result) {
                                        if (error) { /* Handle Error */ }
                                        else {
                                                functionD(resultOfC, function(error, result) {
                                                        if (error) { /* Handle Error */ }
                                                        else {
                                                                console.log(result);
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
});
```

Callback Hell...

```
function functionA(a, callback) { … };
function functionB(b, callback) { … };
function functionC(c, callback) { … };
function functionD(d, callback) { … };
function functionE(e, callback) { … };

var a = ‘some generic value’;
functionA(a, function(error, result) {
          if (error) { /* Handle Error */ }
          else {
                    functionB(resultOfA, function(error, result) {
                              if (error) { /* Handle Error */ }
                              else {
                                        functionC(resultOfB, function(error, result) {
                                                  if (error) { /* Handle Error */ }
                                                  else {
                                                            functionD(resultOfC, function(error, result) {
                                                                      if (error) { /* Handle Error */ }
                                                                      else {
                                                                                functionE(resultOfD, function(error, result) {
                                                                                          if (error) { /* Handle Error */ }
                                                                                          else {
                                                                                                    console.log(result);
                                                                                          }
                                                                                }
                                                                      }
                                                            }
                                                  }
                                        }
                              }
                    }
          }
});
```

Callback Hell...

```
function functionA(a, callback) { … };
function functionB(b, callback) { … };
function functionC(c, callback) { … };
function functionD(d, callback) { … };
function functionE(e, callback) { … };

var a = ‘some generic value’;
functionA(a, function(error, result) {
          if (error) { /* Handle Error */ }
          else {
                    functionB(resultOfA, function(error, result) {
                              if (error) { /* Handle Error */ }
                              else {
                                        functionC(resultOfB, function(error, result) {
                                                  if (error) { /* Handle Error */ }
                                                  else {
                                                            functionD(resultOfC, function(error, result) {
                                                                      if (error) { /* Handle Error */ }
                                                                      else {
                                                                                functionE(resultOfD, functi
                                                                                          if (error) { /*
                                                                                          else {
                                                                                                    consol
                                                                                          }
                                                                                }
                                                                      }
                                                            }
                                                  }
                                        }
                              }
                    }
          }
});
```

Promises

Promises

- A promise represents the eventual result of an asynchronous operation
  - Contains a ***then*** method which registers callbacks for success or error
  - Must be in one of these three states,
    - Pending
    - Rejected
    - Fulfilled
  - Once **Rejected** or **Fulfilled**, promise must have a value and must not transition to any other state.
- Specifications Promises/A and Promises/A+
- Compliance Test Suite
- Implementations
  - ECMAScript 2015 (ES6)
  - 60+ Recognized Implementations
  - Q.js, bluebird, JQuery (3.0 or above), rsvp.js

Promises - Definition

- Defining a async promise function vs async callback function

```
// Before
function asyncFunction(x, callback) { … };

// After
function asyncFunction(x) {
     return new Promise(function (resolve, reject) {
          var y;
          // Huge-ass implementation which uses x and populates y
          if (!y) { reject(new Error('Huge-ass function has not populated y')); }
          else { resolve(y); }
     });
};
```

Promises - Usage

- Differences between callbacks and promises function usage

```
// Before
asyncFunction(a, function(error, result) {

});


// After
asyncFunction(a).then(function(result) {
     // If resolved it goes here
}, function (error) {
     // If rejected it goes here
});
```

Promises - Chaining

- As long as each async function returns a promise, you can do this

```
function functionA(a) { /* returns Promise object */ };
function functionB(b) { /* returns Promise object */ };
function functionC(c) { /* returns Promise object */ };

functionA(a)
     .then(function(resultOfA) {
          return functionB(resultofA);
     })
     .then(function(resultOfB) {
          return functionC(resultofB);
     })
     .then(undefined, function(error){
          // Handle Error
     });
```

The Asynchronous Way?

```
function functionA(a, callback) { … };
function functionB(b, callback) { … };
function functionC(c, callback) { … };
function functionD(d, callback) { … };
function functionE(e, callback) { … };

var a = ‘some generic value’;
functionA(a, function(error, result) {
          if (error) { /* Handle Error */ }
          else {
                    functionB(resultOfA, function(error, result) {
                              if (error) { /* Handle Error */ }
                              else {
                                        functionC(resultOfB, function(error, result) {
                                                  if (error) { /* Handle Error */ }
                                                  else {
                                                            functionD(resultOfC, function(error, result) {
                                                                      if (error) { /* Handle Error */ }
                                                                      else {
                                                                                functionE(resultOfD, function(error, result) {
                                                                                          if (error) { /* Handle Error */ }
                                                                                          else {
                                                                                                    console.log(result);
                                                                                          }
                                                                                }
                                                                      }
                                                            }
                                                  }
                                        }
                              }
                    }
          }
});
```

The Asynchronous Way!

```
function functionA(a) { /* returns Promise object */ };
function functionB(b) { /* returns Promise object */ };
function functionC(c) { /* returns Promise object */ };
function functionD(d) { /* returns Promise object */ };
function functionE(e) { /* returns Promise object */ };

var a = ‘some generic value’;
functionA(a)
     .then(functionB)
     .then(functionC)
     .then(functionD)
     .then(functionE)
     .then(undefined, handleError);
```

Use Cases & Demos

A word on the libraries used for the demo

SuperAgent

SuperAgent is a small progressive **client-side** HTTP request library, and **Node.js** module with the same API, sporting many high-level HTTP client features. View the docs.

Super Agent Http Client

Bluebird Promise Library

.then() - Chaining is awesome

```
getData('/randomNumber').then(function(response) {
  return getData('/isEven?value=' + response.data);
}).then(function(isEven) {
  console.log(isEven);
});
```

Looking back at callbacks

```
getData('/randomNumber', function(err, randomNumber) {
    Var result = randomNumber.data + 5;
    getData('/isEven?value=' + result, function(err, isEven) {
        console.log(isEven);
    })
});
```

.then() - Returning A Value

```
getData('/randomNumber').then(function(randomNumber) {
  return randomNumber.data + 5;
}).then(function(result) {
  console.log("Value: " + 105);
});
```

Handling Errors

```
function getData(url) {
    return new Promise(function(resolve, reject) {
        superagent.get(url, function(err, res) {
            if (err) {
                reject(err);
            }

            resolve(JSON.parse(res.text));
        });
    });
}
```

Handling Errors

```
getData('/randomNumber').then(function(response) {
    console.log(response);
}, function(error) {
  // handle error here
});
```

Error handlers are tricky

```
getData('/randomNumber')
    .then(functionA, ErrorHandlerA)
    .then(functionB, ErrorHandlerB)
    .then(functionC, ErrorHandlerC)
    .then(undefined, ErrorHandlerD)
```

Handling Errors

```
getData('/randomNumber').then(function(response) {
  if (response.data === 0) {
    throw new Error('Received 0');
  } else {
    return response.data;
  }
}).then(function(data) {
  // Save the number somewhere.
  console.log('Saving the number');
}).then(undefined, function(err) {
  // Will catch the ‘Received 0’ error.
});
```

A handler down the line

```
getData('/randomNumber')
    .then(functionA)
    .then(functionB)
    .then(functionC)
    .then(undefined, ErrorHandlerD)
```

Be Careful though

```
getData('/randomNumber')
    .then(functionA)
    .then(functionB)
    .then(functionC)
    .then(functionD, ErrorHandlerD)
```

The fancy .catch()

```
getData('/randomNumber')
    .then(functionA)
    .then(functionB)
    .then(functionC)
    .catch(ErrorHandler)
```

The fancy .catch()

```
getData('/randomNumber').then(function(response) {
  if (response.data === 0) {
     throw new Error('Received 0');
  }
}).then(function(data) {
  // Save the number somewhere.
  console.log('Save the number');
}).catch(function(err) {
  // Will catch the ‘Received 0’ error.
});
```

Dragons - They are not real

```
function getDragon(name) {
    return getData('/dragon');
}

getDragon('Raegal').then(function(data) {
});
```

Dragons

```
function getDragon(name) {
    return getData('/dragon');
}
```

Raegal

Drogon

Viserion

Dragons

```
function getDragon(name) {
    if (name === "Rhaegal") {
        return {
            name: "Rhaegal",
            Species: "Dragon",
            Status: "Alive"
        };
    } else {
        return getData('/dragon');
    }
}
```

Returning values

```
function getDragon(name) {
    if (name === "Rhaegal") {
        return {
            name: "Rhaegal",
            Species: "Dragon",
            Status: "Alive"
        };
    } else {
        return getData('/dragon'); // This returns a promise!
    }
}
```

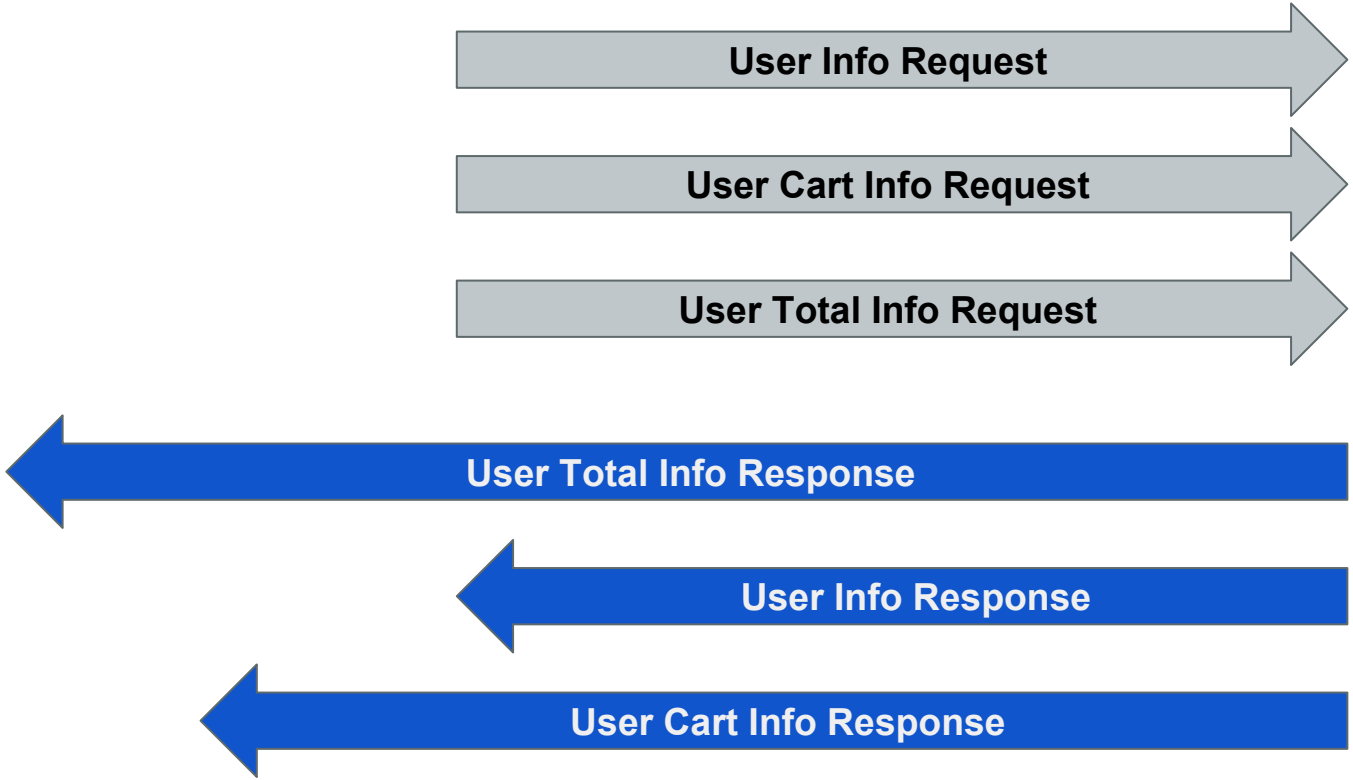
Promise.resolve();

```
function getDragon(name) {
    if (name === "Rhaegal") {
        return Promise.resolve({
            name: "Rhaegal",
            Species: "Dragon",
            Status: "Alive"
        });
    } else {
        return getData('/dragon');
    }
}
```

Promise.reject();

```
function getDragon(name) {
    if (name === "Rhaegal") {
        return Promise.resolve({
            name: "Rhaegal",
            Species: "Dragon",
            Status: "Alive"
        });
    } else if (name === "Drogon") {
        return Promise.reject(new Error("Sorry! You have to talk to Khalesi!"));
    } else {
        return getData('dragon.json');
    }
}
```

Handling Multiple Requests



Promise.all()

```
var userInfoPromise = getData('/userInfo');
var userCartInfo = getData('/cart');
var userTotalInfo = getData('/total');

Promise.all([userInfoPromise, userCartInfo, userTotalInfo]).then(function(results){
    //results is an array with 3 items
    //results[0] => user info
    //results[1] => user cart info
    //results[2] => user total info
}).catch(function(err){
   //opps! Something went wrong.
});
```

Async/Await - The Future

A brave new world of Async/Await

- ES2016 (aka ES7) introduces async/await
- Babel supports

Browser vendors are working hard

Async functions - making promises friendly ☆☆☆☆☆

By Jake Archibald

Human boy working on web standards at Google

Async functions are enabled by default in Chrome 55 and they're quite frankly marvelous. They allow you to write promise-based code as if it were synchronous, but without blocking the main thread. They make your asynchronous code less "clever" and more readable.

Browser support & workarounds

At time of writing, async functions are enabled by default in Chrome 55, but they're being developed in all the main browsers:

- Edge - In build 14342+ behind a flag
- Firefox - active development
- Safari - active development

Async/Await

Inside an async function we have a new keyword await which will wait for a promise.

```
async function getMessage() {
  try {
    var results = await getData('http://localhost:3334/randomNumber');
    console.log(results);
  } catch (e) {
    // Handle error
  }
}
```

Atmo - Mock APIs with ease

https://github.com/Raathigesh/atmo

Or

Google “Atmo mock api”

Thank You!