Async Functions

Roland Guijt

INDEPENDENT SOFTWARE DEVELOPER AND TRAINER | MVP

@rolandguijt rolandguijt.com

Overview

Refresher: callbacks

Refresher: promises

async

await

Callbacks

A function supplied as a parameter to a function

The function will execute it when it's complete

Callbacks

```
fetchRides(rides => {
    //process rides
});
```

```
function fetchRides(callback) {
    //get rides from server e.g. with XmlHttpRequest
    callback(rides);
};
```

Callbacks

```
fetchRides(rides => {
    fetchVisits(rides, visits => {
        fetchVisitors(visits, visitors => {
            //process visitors
        });
    });
});
```

Callbacks

```
try {
    fetchRides(rides => {
        try {
            fetchVisits(rides, visits => {
                try {
                    fetchVisitors(visits, visitors => {
                        //process visitors
                    });
                }
                catch (err) {}
            });
        }
        catch (err) {}
    });
}
catch (err) {}
```

Promises

Placeholder object for eventual result

When resolved it makes the result available

Handles workflow

States: Pending, Fulfilled, Rejected

Chainable

Part of ES2015 aka ES6

Promises

```
fetchRides().then(rides => {
    //process rides
});


function fetchRides() {
    return new Promise((resolve, reject) => {
        //get rides from server e.g. with XmlHttpRequest
        resolve(rides);
        //if error
        reject(errorInfo);
    });
};
```

Callbacks

```
fetchRides(rides => {
    fetchVisits(rides, visits => {
        fetchVisitors(visits, visitors => {
            //process visitors
        });
    });
});
```

Promises

```
fetchRides().then(rides => {
    return fetchVisits(rides);
}).then(visits => {
    return fetchVisitors(visits); {
}).then(visitors => {
    //process visitors
});
```

Callbacks

```
try {
    fetchRides(rides => {
        try {
            fetchVisits(rides, visits => {
                try {
                    fetchVisitors(visits, visitors => {
                        //process visitors
                    });
                }
                catch (err) {}
            });
        }
        catch (err) {}
    });
}
catch (err) {}
```

Promises: Error Handling

```
fetchRides().then(rides => {
    return fetchVisits(rides);
}).then(visits => {
    return fetchVisitors(visits); {
}).then(visitors => {
    //process visitors
}).catch(errorInfo => {
    //handle error
});
```

Portability of Promises

```
let ridesPromise = fetchRides();
//later
ridesPromise.then(rides => {
    //process rides
});
```

Promise.all and Promise.race

Promise.all takes multiple promises and returns a promise which resolves when all supplied promises are done.

Promise.race works the same. It resolves when the first promise is done.

async

Keyword to add to function declaration

Everything you return from that function will be wrapped in a resolved promise

If you throw in the function the promise will return in the rejected state

Returning Promises in an Async Function

```
let fetchRides = async() => {
    return httpGet("https://api.com/rides"); //returns promise
}

let result = fetchRides();

//result is singular promise
```

await

Keyword to add to a call to a function that returns a promise

Only works inside an async function

It makes sure the promise is done before continuing the async function
(all other functions that might run are continuing)

If the function executes successfully the result of await is the return value of the function called

If the function fails await throws the rejection value

Sequential vs Parallel

```
async () => {
    await asyncFunction1();
    await asyncFunction2();
}
```

Sequential vs Parallel

```
async () => {
    await Promise.all([asyncFunction1(), asyncFunction2()]);
}
```

Promisifying

```
function httpRequest (method, url, done) {
    var xhr = new XMLHttpRequest();
    xhr.open(method, url);
    xhr.onload = function () {
        done(null, xhr.response);
    };
    xhr.onerror = function () {
        done(xhr.response);
    };
    xhr.send();
}
```

Promisifying

```
function httpRequest (method, url) {
    return new Promise(function (resolve, reject) {
        var xhr = new XMLHttpRequest();
        xhr.open(method, url);
        xhr.onload = function () {
            if (this.status >= 200 && this.status < 300){
                resolve(xhr.response);
            } else {
                reject({ status: this.status, statusText: xhr.statusText });
            }
        };
        xhr.onerror = function () {
            reject({ status: this.status, statusText: xhr.statusText });
        };
        xhr.send();
    });
}
```

Callbacks

```
try {
    fetchRides(rides => {
        try {
            fetchVisits(rides, visits => {
                try {
                    fetchVisitors(visits, visitors => {
                        //process visitors
                    });
                }
                catch (err) {}
            });
        }
        catch (err) {}
    });
}
catch (err) {}
```

Promises

```
fetchRides().then(rides => {
    return fetchVisits(rides);
}).then(visits => {
    return fetchVisitors(visits); {
}).then(visitors => {
    //process visitors
}).catch(errorInfo => {
    //handle error
});
```

Async and Await

```
try {
    let rides = await fetchRides();
    let visits = await fetchVisits(rides);
    let visitors = await fetchVisitors(visits);
    //process visitors
}
catch (err) {
    //handle error
}
```