Brief introduction to Asynchronous JavaScript

WASA: Web and Software Architecture

Enrico Bassetti

Promises

A **Promise** is an object that refer to a future value (a.k.a. "proxy").

If a function returns a Promise, it means that the return value will be available in the future (the execution is asynchronous).

To use a Promise, you need to specify a function that is executed when the result is available, and a function that is executed if something goes wrong.

Promises

A Promise is in one of these states:

- pending: before starting, or async code is running
- fulfilled: the operation was completed successfully
- rejected: the operation failed

Using promises

Let's assume that doSomething() is returning a Promise.

```
doSomething().
  then((result) => {
    // Do something with the result
  })
  .catch((error) => {
    // Do something with the error
  })
```

Using promises: async

Execution is asynchronous!

```
var txt;
doSomething().
  then((result) => {
    txt = result;
  })
  .catch((error) => {
    // Do something with the error
  });
// The following alert may be executed BEFORE txt is set
alert(txt);
```

Async-await pattern

The **async-await** pattern is a way of writing asynchronous code using the “synchronous” style.

Functions that needs to use this style should be declared as async.

Note: this is syntactic sugar. The code is still asynchronous, and you should be aware of that.

Async-await example

E.g., this code:

```
function buttonClicked() {
  doSomething().then((result) => {
      alert(result);
    }).catch((error) => {
      // Do something with the error
    });
}
```

Code in async-await pattern:

```
async function buttonClicked() {
  let txt = await doSomething();
  alert(txt);
}
```