Reactive Programming

Konzepte der Programmiersprachen am 14.11.2017

Overview

- Javascript recap
  - Basics
  - Event handling
  - Callback
  - Promise
- Reactive Principles
- Reactive in JavaScript
  - RxJS
  - Socket.io
- Exercise

Javascript recap

Javascript basics

- Variables

```
var newString = “String”;
let a, b = 10;
```

||

```
let newString = “String”;
let c = 10, d = “String”;
```

- Constants

```
const newConstant = 10;
```

Javascript basics

- Objects

```
var myObject = {
   Name : “Max”,
   Nachname : “Mustermann”,
   Address : {
              Straße : “Musterstraße”,
              Hausnummer : 5,
              PLZ : “11111” ,
              Stadt : “Musterstadt”}}
```

- Arrays

```
var myElements = [ 2 , 3, 4, 5 ]; || var myElements = new Array(2 , 3, 4, 5);

var alsoMyElements = [ 2, “Hello”, new Date(), 5 ];
```

Javascript basics

- Functions

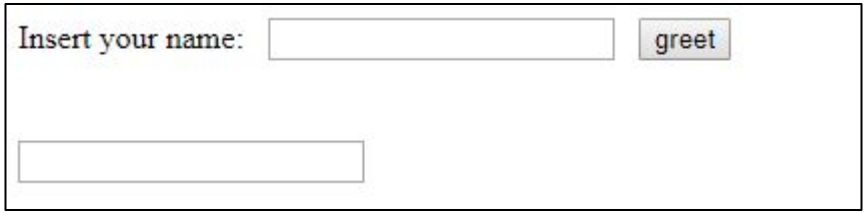
```
function myObjectFunction  (vorname, nachname){
        return { Vorname :  vorname,    Nachname : nachname};
}

var myObjectFunction  = function (vorname, nachname){
        return { Vorname  :  vorname,    Nachname  : nachname};
};


function myFunction  (vorname, nachname){
        var name  = myObjectFunction(vorname, nachname);
        return name;
}
```

Javascript basics

- Global functions

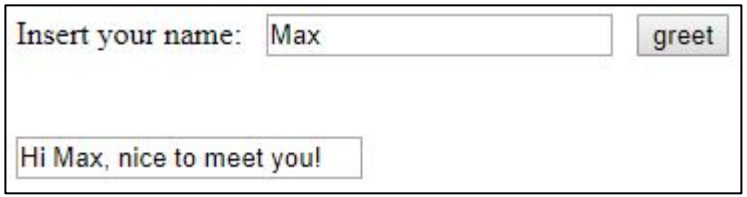
eval	eval(“3 + 2 * 4”) // 11
isNaN(Value)	isNaN(“TRUE”) // true || isNaN(8) // false
Number(Object)	Number(“354646.55”) // 354646.55
String(Object)	String(354646.55) // “354646.55”
parseFloat(String)	parseFloat(“33.33333333”) // 33.33333333
parseInt(String)	parseInt(“456.235665”) // 456

Javascript basics - event handling

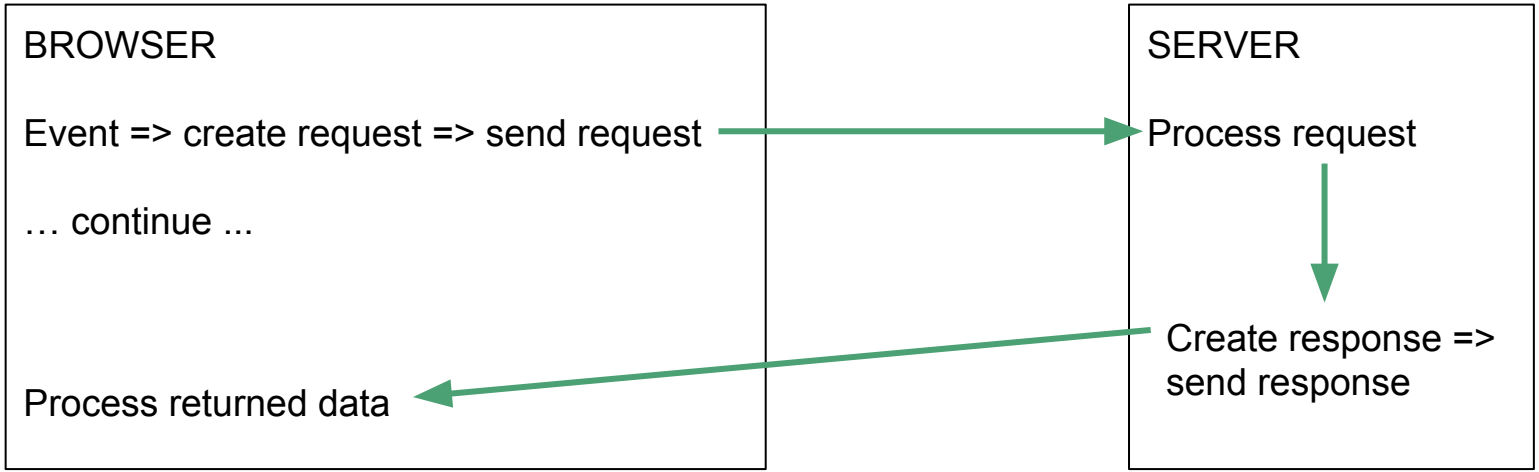



```
<div>
    <div>
        <label for="text">Insert your name:</label>
        <input id="text" type="text">
        <input type="button" value="greet" onClick="greet()">
    </div>
    <div>
        <input id="result" type="text" readonly></input>
    </div>
</div>
```

```
function greet() {
    var value = document.getElementById("text").value;
    document.getElementById("result").value = "Hi " + value
            + ", nice to meet you!";
}
```



Javascript basics - callback



Javascript basics - callback

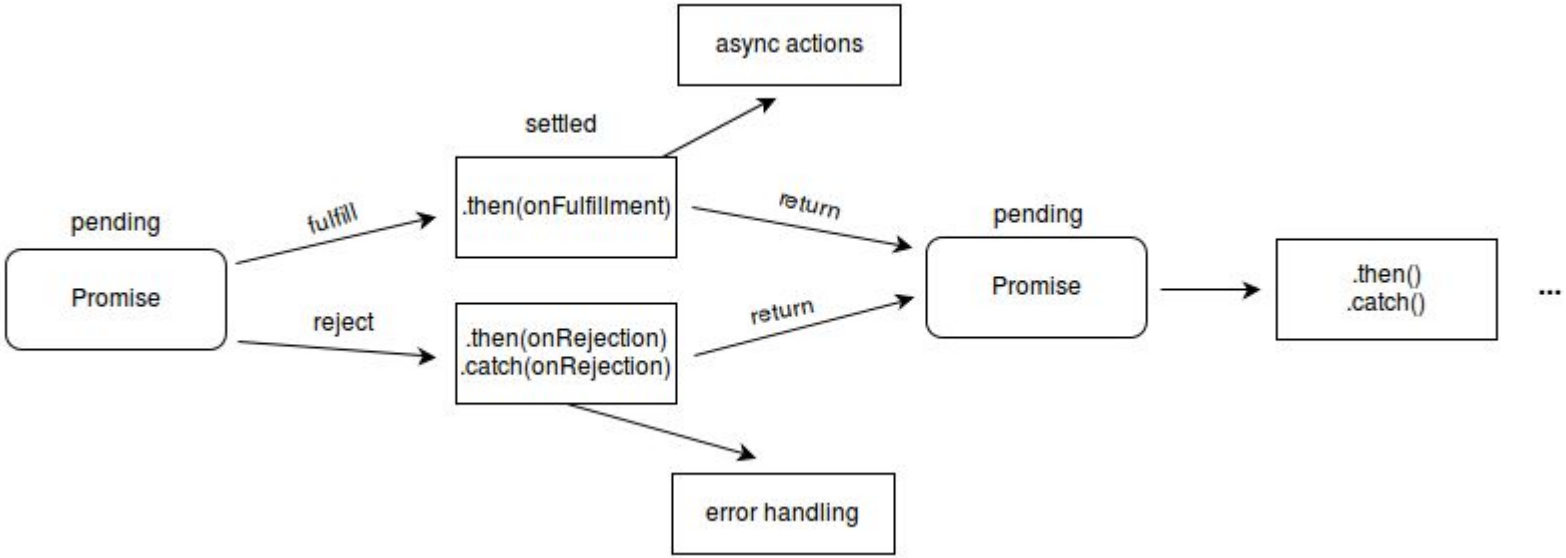
```
<div>
    <input type="button" value="Load Data" onClick="loadData()">
    <div id="demo"></div>
</div>
```

```
function loadData() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function(){
        if (this.readyState == 4 && this.status == 200) {
            processData(this);
        }
    };
    xhttp.open("GET","URL",true);
    xhttp.send();
}
function processData(that) {
        var response = JSON.parse(that.responseText);
        var txt = "<ul>"
        for (i in response.ProductCollection) {
            txt += "<li>" + response.ProductCollection[i].ProductId + " - "
                    + response.ProductCollection[i].Category + "</li>";
        }
        txt += "</ul>"
        document.getElementById("demo").innerHTML = txt;

}
```

Javascript basics - promise

```
var myPromise = new Promise(function(resolve, reject) {
    if (/* condition */) {
        resolve(/* value */);  // fulfilled successfully
    }
    else {
        reject(/* reason */);  // error, rejected
    }
});

myPromise.then((val) => console.log("fulfilled:", val),
               (err) => console.log("rejected: ", err));

           or:

myPromise.then((val) => console.log("fulfilled:", val))
 .catch((err) => console.log("rejected:", err));
```

Javascript basics - promise



Javascript basics - promise

```
function promiseTest(){

      var myPromise = promise();
      myPromise.then((that) => {var response = JSON.parse(that.responseText);//render content})
           .catch((err) => console.log("rejected:", err));

}
function promise(){
    return new Promise((resolve, reject) => {
        var xhttp = new XMLHttpRequest();
        xhttp.open("GET","URL",true);
        xhttp.onreadystatechange = function(){
            if (this.readyState == 4 && this.status == 200) {
                resolve (this);
            }else if (this.readyState == 4 && this.status !== 400){
                reject( false);
            }
        };
        xhttp.send();
    })
}
```

Javascript basics - functional programming

```
var light = {
        on  : false
};
function switchLight(){
    light.on = !light.on;
}
switchLight();
```

```
var switchedOffLight = {
        on : false
};
function switchLight(light){
    return { on: !light.on };
}
var switchedOnLight = switchLight(switchedOffLight);
```

- pure functions
- immutability

Javascript basics - functional programming

```
var values = [1, 2, 3, 4];

var multiplied = values.map(function multiplyBy2(value){
  return value * 2;
});

//multiplied [2, 4, 6, 8]

var reduced = values.reduce(function(accumulator, currentValue) {
    return accumulator + currentValue;
});

//reduced 10


var filtered = values.filter(value => value % 2 === 0);

//filtered [2, 4]
```

Reactive programming

“ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences.”

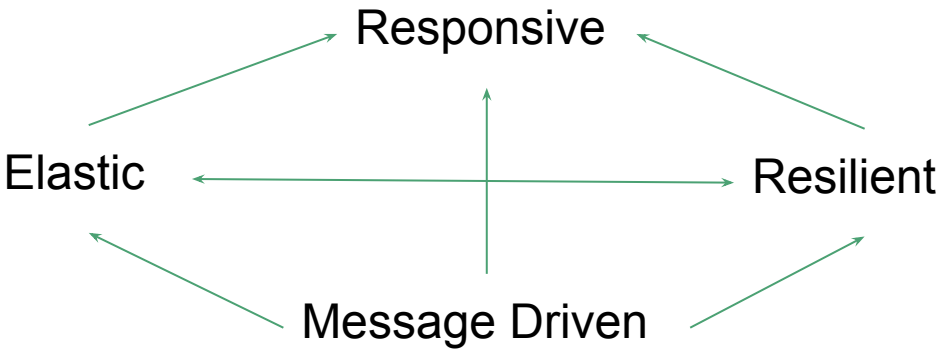
Observables fill the gap by being the ideal way to access asynchronous sequences of multiple items

	single items	multiple items
synchronous	`T getData()`	`Iterable<T> getData()`
asynchronous	`Future<T> getData()`	`Observable<T> getData()`

ReactiveX

- API definition (no implementation!)
  - Implementation Details (like multithreading) are NOT defined
- *“combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming”*
- *It’s “Reactive Programming” (= stream of discrete values)*
- *Not “Functional Reactive Programming” (= stream of continuous values - e.g. time)*
- *21 official implementations (RxJava, RxJS, RxScala, ...)*

Reactive manifesto



Reactive in JavaScript

RxJS & Socket.io

Example: Double Clicks (RxJS)

```
const button = document.getElementById('myButton');
const clickStream = Rx.Observable.fromEvent(button, 'click');

const multiClicks = clickStream
  .buffer(clickStream.debounce(250))
  .map(x => x.length)
  .filter(x => x>=2);

multiClicks.subscribe(x => console.log(x + ' Click!'));
```

Example: Wikipedia suggestions (RxJS)

```
const inputs = Rx.Observable.fromEvent(myInputField, 'input')
                            .map(e => e.target.value);

const newSearchTerm = inputs.filter(text => text.length > 2)
                            .throttle(500);

newSearchTerm.flatMapLatest(searchWikipedia)
             .subscribe(renderData, renderError);

function searchWikipedia(searchTerm) { /* ... */} // returns a promise
function renderData(data) { /* ... */}
function renderError(error) { /* ... */}
```

Tweet Displayer

The old way

```
class TweetDisplayer {

  tweets = [];
  tweetService;
  previousRawData;

displayNewTweets() {
  this.tweetService.fetchNewData();
  this.pollForNewTweets();
}
```

```
pollForNewTweets() {
  const rawData = this.tweetService.getData();

  if(rawData === null || rawData === previousRawData) {
    setTimeout(() => this.update(), 200);
  } else {
    this.previousRawData = rawData;
    this.tweets = transformData(rawData);
  }
}

}
```

The new way

```
class PromiseTweetDisplayer {
  tweets = [];
  tweetService;

  displayNewTweets() {
    this.tweetService.getData()
      .then((rawData) => {
        this.tweets = transformData(rawData);
      });
  }
}
```

```
class TweetService {
  getData() {
    return $.get(twitterUrl);
  }
}
```

The reactive way (RxJS)

```
class ReactiveTweetDisplayer {
  tweets = [];
  tweetService;

  constructor() {
    tweetService.getData().subscribe((rawData) => {
      this.tweets = transformData(rawData);
    });
  }

  displayNewTweets() {
    this.tweetService.fetchNewData();
  }
}
```

```
class TweetService {
  constructor() {
    this.updateStream = new Rx.Subject();
    this.tweetStream = this.updateStream
      .flatMapLatest(() => $.get(twitterUrl));
  }

  getData() {
    return this.tweetStream;
  }

  fetchNewData() {
    this.updateStream.onNext();
  }

}
```

Facts about RxJS

- Dependency free
- Highly compatible (jQuery, IE6 (!), ...)
- Open source
- Well documented
- Popular
- Default part of Angular

Socket.io

```
const app = require('express')();
const http = require('http').Server(app);
const io = require('socket.io')(http);

io.on('connection', function (socket) {

    socket.on('chat message', function (msg) {
        io.emit('chat message', message);
    });

});

http.listen(3000);
```

Exercises

Exercises

https://goo.gl/zgYYWs

Check the links at the top of the Plunker for docs.

Recommended order:

1. Input
2. Arrays / Objects
3. Chat (connect to Eduroam wlan!)
4. Typing of the Dead