LODASH FOR PRESIDENT

Christian Ulbrich, CDO Zalari UG

AGENDA PROPAGANDA

- Recap: LoDash
- Why?
- Installation
- Examples galore
- Links

RECAP - LODASH

„Who has never heard of LoDash?“

RECAP - LODASH

- available on *npm* and *bower*
- generic JavaScript library, thus works in backend *and* frontend
- either load it via script tag or with a module loader of your choice
- custom builds, bundles are available

WHY LODASH

„Look, I invented something…“

WHY LODASH

THE WIEL!

WHY LODASH

```
var triangle = {a:1, b:2, c:3};

function ColoredTriangle() {
  this.color = "red";
}

ColoredTriangle.prototype = triangle;

var obj = new ColoredTriangle();


_.forOwn(obj, function (value, key) {
  console.log('obj.' + key + ' = ' + value);
});
```

WHY LODASH

- Human make errors *and* we don't like to repeat ourselves over and over again

- working with declarative code enables less errors and fosters maintainability

- -> „*human JavaScript*“

INSTALLATION

```
<script src="lodash.js"></script>
```

```
$ npm i --save lodash
```

```
var _ = require('lodash');
```

EXAMPLES COLLECTIONS

- LoDash has *enhanced* Shortcuts for typical array functions, like _.*forEach*, _.*map*, _.*reduce*, _.*filter*
- they used to be performance optimized
- they also work on *Collections* (Arrays + abused Objects)

FILTERING + FINDING

- _.filter returns Elements that the passed predicate is true

- _.find returns the first Element that the passed predicate is true

- brethren: _.reject, _.findLast, _.some, _.without

FILTERING + FINDING

- there are some nice built-in predicate function producing functions:

- _.matches, _.matchesProperty, _.property

- lodash analyzes the signature of the passed argument to *most* of its collection functions and calls them appropriately

FILTERING + FINDING

```
'use strict';

var _ = require('lodash');

var presenters = [{name:'Erik', fame: { guaranted : true, minutes: NaN}, present: true},
                  {name:'Christian', fame: { guaranted : undefined, minutes: Infinity}, present: true},
                  {name:'Veit', fame: {guaranted: true, minutes: 50}, present: true},
                  {name:'Achim', fame: {guaranted: true, minutes: 50}, present: false}
];

//all presenters, that are present
console.log(_.filter(presenters, {present:true}));

//all presenters with fame guaranted
console.log(_.filter(presenters, ['fame.guaranted', true]));

//names only
console.log(_.map(presenters, 'name'));
```

FILTERING + FINDING

- What about combining individual functions and rules?

- How can we figure out only the names of all the presenters that obey the rule *famePredicate*?

LODASH CHAINING

- LoDash allows for *chaining* of its functions via _.*chain* wrapper, then the value of each call is passed along the *chain*

- at the end of the chain, we can access the value with .values()

LODASH CHAINING

```
//famePredicate
//famePredicate
let famePredicate = (obj) => obj.fame.minutes > 15;
}
```

```
//names of presenters with fame
console.log(_.chain(presenters)
  .filter(famePredicate)
  .map('name')
  .value());
```

UTILITY FUNCTIONS

- Object manipulation: _.merge, _.clone, _.cloneDeep
- beware of different LoDash versions
- _.merge *modifies* the **first** parameter!

UTILITY FUNCTIONS

- _.parseInt(numbery)
- _.snakeCases, _.startCase, _.pad
- _.inRange, __.random

UTILITY FUNCTIONS

- _.isUndefined
- _.isNumber
- _.is*Type (Array, Date, Buffer, Integer, Object, …)*

UTILITY FUNCTIONS

```
'use strict';

const _ = require('lodash');

let undef;
let nil = null;


if (!undef) {
  console.log('undef is not undefined');
}

if (!nil) {
  console.log('nil seems to be undefined');
}

if (_.isUndefined(undef)) {
  console.log('undef is really undefined');
}
```

UTILITY FUNCTIONS

```
'use strict';

const _ = require('lodash');

let numberTests = [NaN, Infinity, '3', 'four', 42];

let naiveIsNumber = (numberObj) => !isNaN(numberObj);

let naiveResults = _.map(numberTests, naiveIsNumber);
let realResults = _.map(numberTests, _.isNumber);

console.log('TestNumbers : ', numberTests);
console.log('naiveResults: ', naiveResults);
console.log('realResults : ', realResults);
```

ARRAY GOODNESS

- _.zip

- _.flatten, _.flattenDeep, _.join, _.pull, _.pullAll, …

COLLECTION GOODNESS

- _.sortBy, _.flatMap, _.shuffle, _.groupBy

- _.flatten, _.flattenDeep, _.join, _.pull, _.pullAll, …

HUMAN JAVASCRIPT

- META!
- whenever maintainability is more important than performance, you should write *human* JavaScript
- LoDash' expressive and clever naming of functions allow to do that easily

HUMAN JAVASCRIPT

- simple StateMachine, has valid state transitions
- a declarative approach lets the code *speak*

```
function changeState (fromState, toState) {
  let allowedTransitions =
    [{from:'PENDING', to:'ACCEPTED'}, {from:'PENDING', to:'REJECTED'},
    {from:'PENDING', to:'DENIED'}, {from:'REJECTED', to:'CANCELED'},
    {from:'ACCEPTED', to:'CANCELED'}, {from:'ACCEPTED', to:'CLOSED'}];
  let validStateChangeFound = _.find(allowedTransitions, {from:fromState, to:toState});
```

- find returns *undefined* if **nothing** is found

HUMAN JAVASCRIPT

- crude indexOf syntax with bit operators versus declarative _.*includes*

```
'use strict';

const _ = require('lodash');

let haystack = ['needle', 'Heart of Gold', 'I ♡ JavaScript!', 'PHP is ' +'\u{1F4A9}', 42];

if (~(haystack.indexOf('Heart of Gold'))) {
  console.log('This is super f4st, you 1337 programmar, you!');
}

if (_.includes(haystack, 'Heart of Gold')) {
  console.log('This is for managers delight only.');
}

if (haystack.includes && haystack.includes('Heart of Gold')) {
  console.log(`This JS engine must be from the future, just like ${haystack[3]}`);
}
```

LINKS

- Human JavaScript http://read.humanjavascript.com/

- LoDash et al. http://lodash.com

- 10 Javascript Utility Functions That You Should… http://bit.ly/1IHH5Ba

BONUS

- http://lodash.com and your web console are a poor man's playground for testing LoDash funtions… because LoDash loads itself!