

Paradigmen in JavaScript

Alex Thalhammer

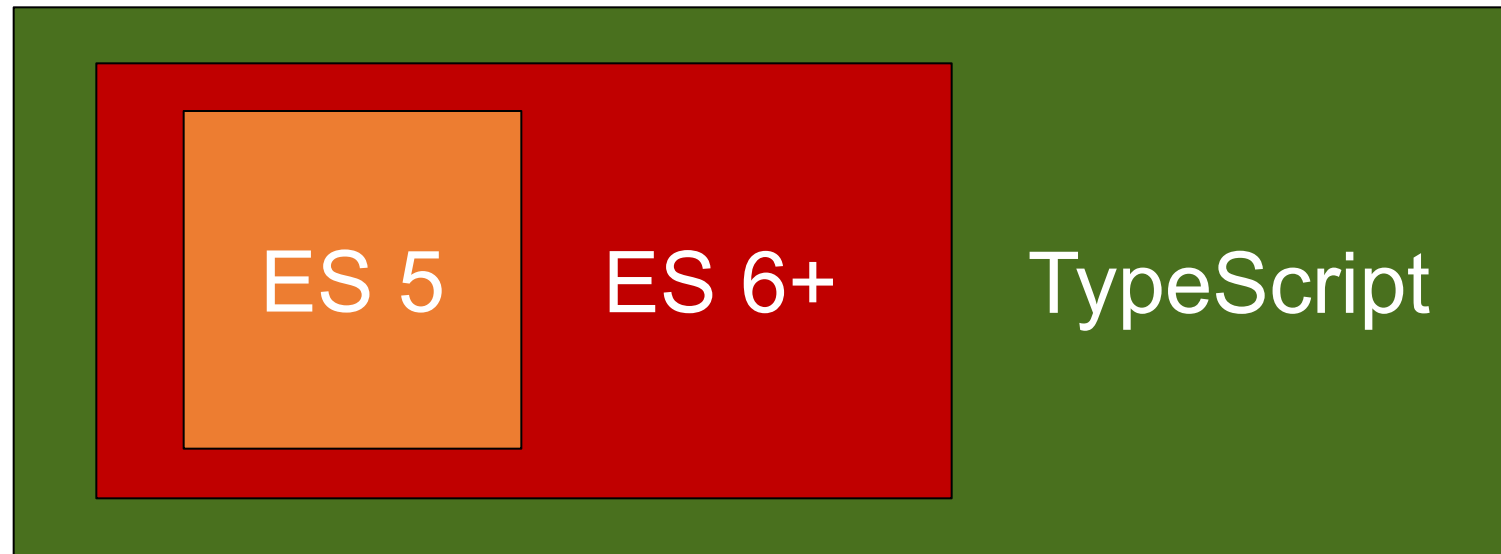
Inhalt

- Überblick
 - Prozedurales Paradigma
 - Funktionales Paradigma
 - Objektorientiertes Paradigma
 - DEMO
- Mehr Details
 - Funktionen und this
 - Datentypen
 - Exceptions
 - Prototypen
 - Modulares Paradigma
 - DEMO

Überblick

ES 5 < ES 6 < TypeScript

ES 6+: Offiziell
ES 2015, 2016, 2017



←
Kompilierung

Prozedurales Paradigma

Das prozedurale Paradigma

```
function calcZins(k, p, t) {  
    var result = k * p * t / 36000;  
    return result;  
}
```

```
var result = calcZins(200,2,360);  
alert("Ergebnis: " + result);
```

Ausgewählte vordefinierte Prozeduren

```
var two = parseInt("2");
```

```
var twoPointTwo = parseFloat("2.2");
```

```
var isSevenNaN = isNaN("seven");
```

Funktionales Paradigma

Das funktionale Paradigma

```
function forEach(ary, action) {  
    for (var i = 0; i < ary.length; i++) {  
        action(ary[i]);  
    }  
}
```

Das funktionale Paradigma

```
function forEach(ary, action) {  
    for (var i = 0; i < ary.length; i++) {  
        action(ary[i]);  
    }  
}
```

```
function showItem(item) { alert(item); }  
var myInts = [1, 2, 3, 4];  
forEach(myInts, showItem);
```

Das funktionale Paradigma

```
function forEach(ary, action) {  
    for (var i = 0; i < ary.length; i++) {  
        action(ary[i]);  
    }  
}
```

```
function showItem(item) { alert(item); }  
var myInts = [1, 2, 3, 4];  
forEach(myInts, showItem);
```

```
forEach(myInts, function (item) {  
    alert(item);  
});
```

Lambda-Ausdrücke ab EcmaScript 6

```
forEach(myInts, (item) => {  
    alert(item);  
});
```

```
forEach(myInts, item => { // Nur ein Parameter  
    alert(item);  
});
```

```
forEach(myInts, item => alert(item) ); // Nur eine Zeile == Rückgabewert
```

DEMO

functions

Objektorientiertes Paradigma

Das objektorientierte Paradigma

```
var flugBuchung = {  
  von: "Graz",  
  nach: "Mallorca",  
  passagiere: [  
    {  
      vorname: "Max", nachname: "Muster"  
    },  
    {  
      vorname: "Susi", nachname: "Sorglos"  
    }  
  ],  
  bezahlung: {  
    art: "Kreditkarte", betrag: 250, bezahlt: true  
  }  
};
```

Objektliterale

Konstruktor-Funktionen

```
function Person(id, vorname, nachname) {  
    this.vorname = vorname;  
    this.nachname = nachname;  
  
    this.vollerName = function () {  
        return id + ": " + this.vorname + " " + this.nachname;  
    }  
}
```

```
var rudi = new Person(47, "Rudolf", "Rentier");  
alert(rudi.vorname);  
alert(rudi.nachname);  
alert(rudi.vollerName());
```


Klassen ab ES6

```
class Person {  
  
    constructor(id, vorname, nachname) {  
        this.id = id;  
        this.vorname = vorname;  
        this.nachname = nachname;  
    }  
  
    vollerName() {  
        return this.id + ": " + this.vorname + " " + this.nachname;  
    }  
}
```

DEMO

class

Übung 1.1

Mehr Details

Funktionen und this

This

- *this* in Funktion verweist auf aktuellen „Kontext“
- Aufrufer legt Kontext fest

Kontext

- `obj.methode()`
 - `this == obj`
- `func.call(x, y, z)`
 - `this == x`, Parameter: `y, z`
- `new Func()`
 - `this ==` Neues „leeres“ Objekt
- `func()`
 - `this ==` globales Objekt (*window* in Browser)

Gedankenexperiment

- Worauf verweist this in doStuff?
- dbj.doStuff();
- var m = obj.doStuff;
 m();

Function

- Jede Funktion wird durch ein Function-Objekt repräsentiert
- Methoden:
 - `func.call(thisArg, arg1, arg2, ...)`
 - `func.apply(thisArg, aryArray)`
 - `func2 = func.bind(thisArg)`

Lambda-Ausdrücke binden this

```
forEach(myInts, function (item) {  
    console.debug(this); // Aufrufer (= forEach kann this festlegen)  
});
```

```
var that = this;  
forEach(myInts, function (item) {  
    console.debug(that);  
});
```

```
forEach(myInts, (item) => {  
    console.debug(this);  
});
```

DEMO

this

Datentypen

Überblick

- number
 - `var num = 3.14;`
 - `var i = 0;`
- boolean
 - `var ok = true;`
- string
 - `var name = "Max";`
 - `var multiline = `Hallo ${name}!`;` // ES6
- array
 - `var ary = [1, 2, 3];`
- object
 - `var obj = { x: 1, y: 2 };`
- function
 - `var f = function () { ... }`
- null
 - `var maybe = null;`
 - "Eigenschaft hat keinen Wert"
- undefined
 - `var maybe = undefined;`
 - "Eigenschaft existiert nicht"

typeof

- Liefert Datentyp als String zurück
- `if (typeof value == "undefined") { ... }`

Vergleiche

- == und != führen Typumwandlungen durch
 - "1" == 1 // true
- === und !== verlangen auch Gleichheit bei Typen
 - "1" === 1 // false

Booleans

- Falsy
 - false, null, undefined, 0, "", NaN
- Truthy
 - !falsy

Objekte sind Dictionaries

- `rud.name == rudi['name']`
- Ersteres kann besser optimiert werden

Keys eines Objektes iterieren

- for (let key **of Object.keys(rudi)**) { // of: ES6
 console.debug(key, rudi[key]);
}
- for (let key **in** rudi) {
 console.debug(key, rudi[key]);
}

Deklarationen

- `var x;`
 - Scope: Gesamte Funktion, gilt ab Beginn der Funktion (hoisting)
- `let y;`
 - Scope: Aktueller Block, gilt ab Deklaration
- `const z = 3.14;`
 - Konstante; Scope, wie bei.

Globale Objekte (Auszug)

Number

Boolean

String

Date

Array

RegExp

Exception

Exceptions

// Behandlung

```
try {  
    ...  
}  
catch (e) {  
    ...  
}  
finally {  
    ...  
}
```

// Auslösen

```
throw 17;  
throw "error!";  
throw new Error("Fehler");
```

Error dient auch als Basis-Klasse für eigene Exception-Typen

Exceptions sind nicht Teil von Methoden-Signaturen!

SOFTWAREarchitekt.at

DEMO

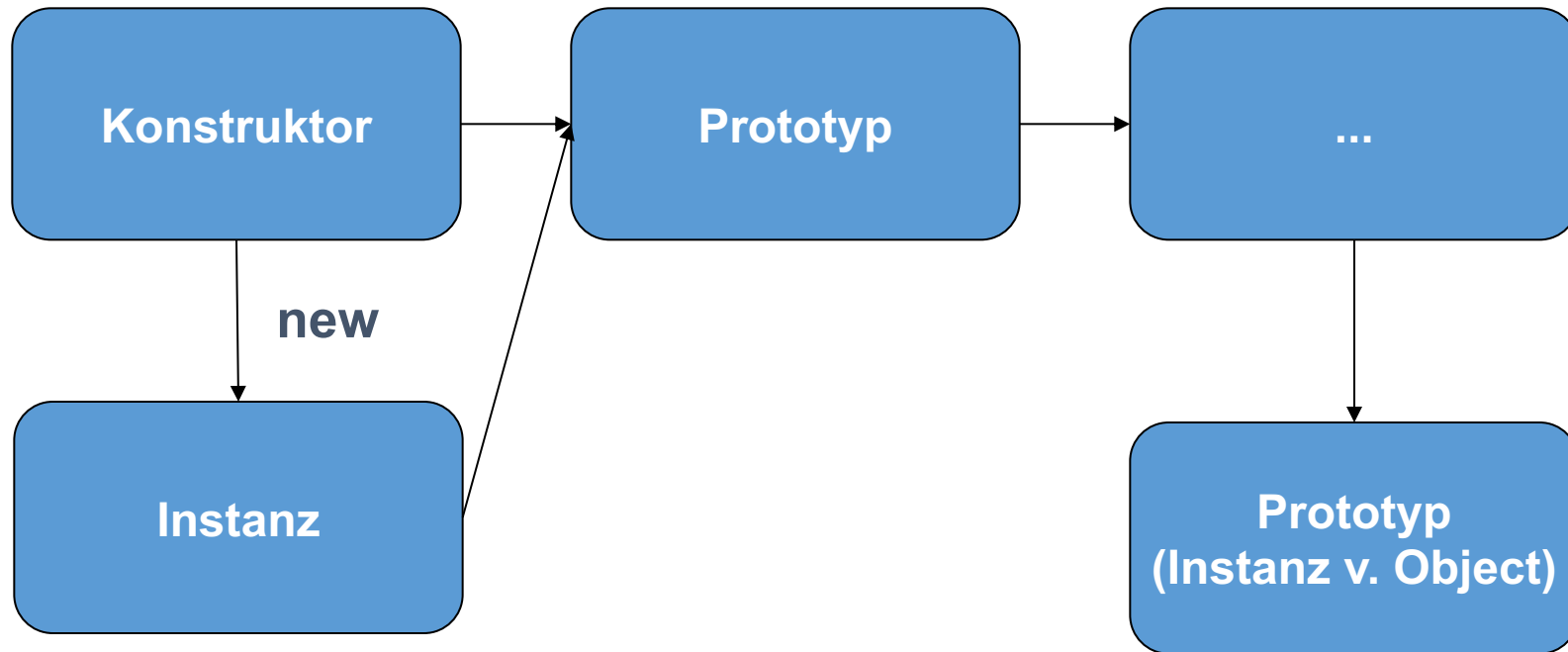
Exception bei ungültigen Parametern

Prototypen

Prototypen

- Jedes Objekt hat einen Prototyp
- Properties (Methoden), die im Objekt nicht gefunden werden, sucht JavaScript im Prototyp

Prototypen



Beispiel ohne Prototyp

```
function Person(id, vorname, nachname) {  
    this.id = id;  
    this.vorname = vorname;  
    this.nachname = nachname;  
  
    this.vollerName = function() {  
        return this.vorname + " " + this.nachname;  
    }  
}
```

Beispiel für Prototypen

```
function Person(id, vorname, nachname) {  
    this.id = id;  
    this.vorname = vorname;  
    this.nachname = nachname;  
}
```

```
Person.prototype.vollerName = function () {  
    return this.vorname + " " + this.nachname;  
}
```

Beispiel für Prototypen

```
function Dienstnehmer(id, vorname, nachname, abteilung) {  
    this.abteilung = abteilung;  
}
```

Beispiel für Prototypen

```
function Dienstnehmer(id, vorname, nachname, abteilung) {  
    this.abteilung = abteilung;  
}
```

```
Dienstnehmer.prototype = new Person();
```

Beispiel für Prototypen

```
function Dienstnehmer(id, vorname, nachname, abteilung) {  
    Person.call(this, id, vorname, nachname);  
    this.abteilung = abteilung;  
}
```

```
Dienstnehmer.prototype = new Person();
```

Beispiel für Prototypen

```
function Dienstnehmer(id, vorname, nachname, abteilung) {  
    Person.call(this, id, vorname, nachname);  
    this.abteilung = abteilung;  
}
```

```
Dienstnehmer.prototype = new Person();
```

```
Dienstnehmer.prototype.wechsle = function(neueAbteilung) {  
    console.debug(this.vollerName() + " wechselt zu " + neueAbteilung);  
  
    this.abteilung = neueAbteilung;  
}
```


Beispiel für Prototypen

```
var dn = new Dienstnehmer(1, "Max", "Muster", "Management");  
console.debug('Dienstnehmer', dn);  
dn.wechsle("Dev");  
console.debug('Nach Wechsel', dn);
```

DEMO

Subclass

Spread-Operator

Beispiel für Spreading

```
const dn2 = { ...dn, vorname: 'Maria' };
```

```
const myIntegersExtended = [ ...myIntegers, 4 ];
```

DEMO

spreading

Module

Das modulare Paradigma

```
(function () {  
    var info = "Hallo Welt";  
    function sum(a, b) { return a + b; }  
    function alertInfo() { alert(info); }  
})();
```

IIFE: Immediately-invoked
function expression

Das modulare Paradigma

```
var tools = tools || {}; // <-- "Leeres" Objekt
```

```
(function (root) {  
    var info = "Hallo Welt";  
    root.sum = function(a, b) { return a + b; }  
    root.sayHello = function() { alert(info); }  
})(tools);
```

```
var sum = tools.sum(1,2);  
alert(sum);  
tools.sayHello();
```

```
var sumFunc = tools.sum; // import tools.sum;  
sum = sumFunc(1,3);
```


EcmaScript-Modulsystem

- Ab EcmaScript 6
- Jede Datei ist ein Modul
- Dateien können Inhalte für andere Dateien exportieren
- Andere Dateien können diese Inhalte importieren

export und import

```
// a.js  
function calcPriceInternal(flightId, discount) { ... }  
export function calcPrice(flightId) { ... }
```

```
// b.js  
import { calcPrice } from './a';  
calcPrice(17);
```

Übung 1.2