Florian Rappl

TypeScript

Profil

- Freiberuflicher IT-Berater
- Doktorand Theoretische Physik
- Microsoft MVP für Visual C#
- CodeProject MVP

Kurzer Abriss

- Aufschwung von JS
- Vielzahl von Bibliotheken / Code
- Größere Projekte
- Einsatz erweitert
- Sprache stagniert

~~Probleme~~ Vorteile von JS

- ~~Wenig elementare Typen~~ Alles vorhanden
- ~~Speicherverwaltung eingeschränkt~~ Ein Problem weniger!
- ~~Keine Kompilierung~~ Schneller (jit), flexibler
- ~~Viele Implementierungen~~ Hohe Verbreitung
- ~~Langsamer Standard~~ Solide Entwicklung

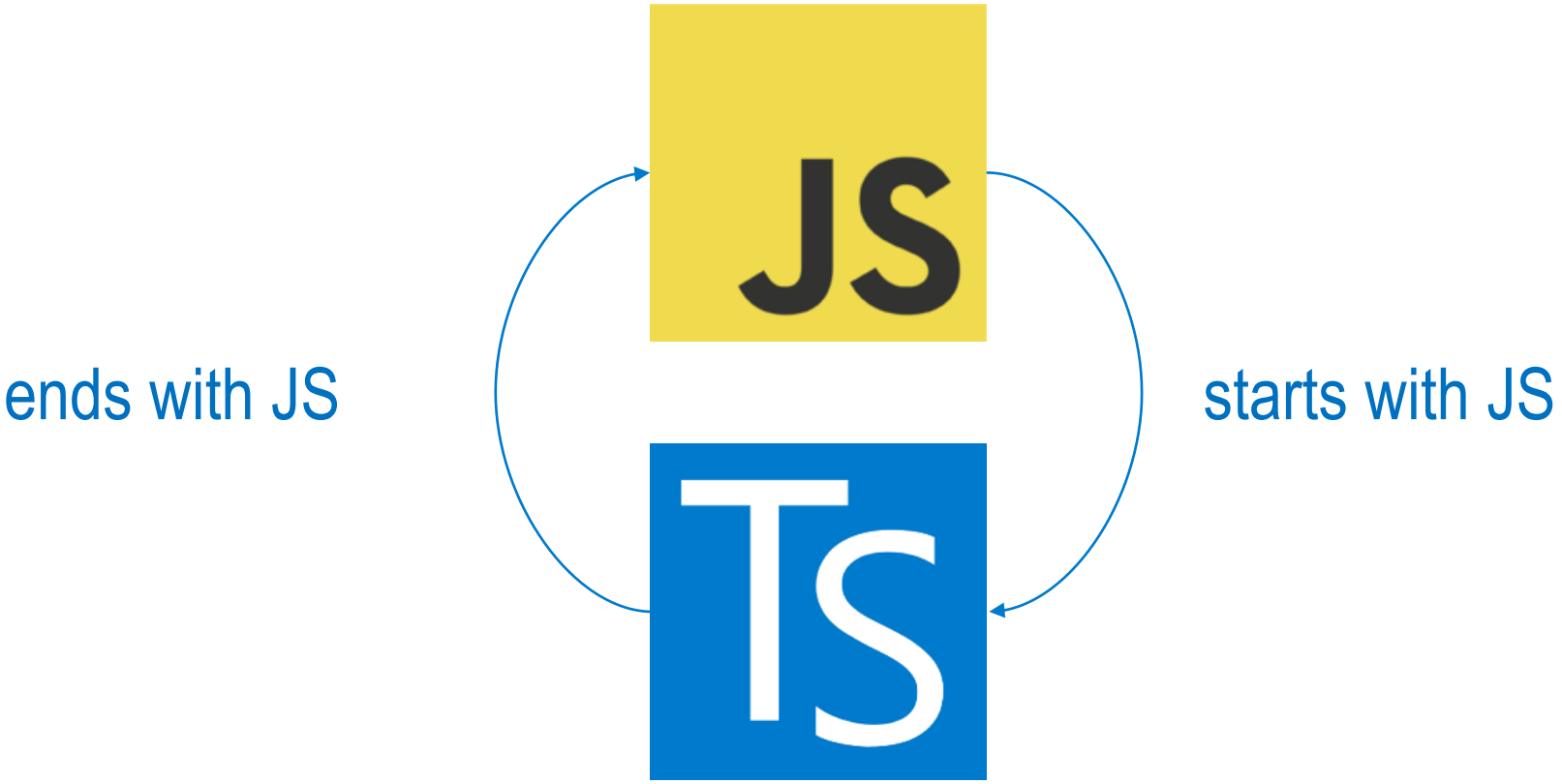
Keep those benefits!

aber…

Optional

- Typenüberprüfung
- Syntaxchecking
- Modularisierung
- IntelliSense
- Sinnvolle Wrapper

TypeScript



TypeScript

- JavaScript ist bereits TypeScript
- Zusätzlich:
  - Optionale Typisierung
  - Modularisierung
  - ES6 Syntax
  - Definitionen
- TypeScript wird zu JavaScript

Wait a minute …

- *X* zu JS wurde bereits umgesetzt
  - CoffeeScript
  - Dart
  - PyJS
  - GWT
  - …

USP ♥

- JavaScript wird nicht versteckt
- Obermenge von ES(6)
- Inkrementelles einpflegen möglich
- Cross-Implementation sicher
- Besseres Tooling

Features

- Typenannotationen
- Klassen (inkl. Vererbung)
- Interfaces + Generics
- Enumerationen
- Definitionen + Module
- Scope Funktionen

```
var fib = function(n: number): number {
  n = ~~n;
  if (n < 1) return 0;
  else if (n === 1) return 1;
  else return fib(n - 1) + fib(n - 2);
};

var result = fib(12);
console.log(result);
```

TypeScript

learn play download interact

TypeScript | Walkthrough: Generics | Share

```typescript
class Greeter<T> {
    greeting: T;
    constructor(message: T) {
        this.greeting = message;
    }
    greet() {
        return this.greeting;
    }
}

var greeter = new Greeter<string>("Hello, 

var button = document.createElement('butto
button.textContent = "Say Hello";
button.onclick = function () {
    alert(greeter.greet());
}

document.body.appendChild(button);

```

Run | JavaScript

```javascript
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return this.greeting;
    };
    return Greeter;
})();

var greeter = new Greeter("Hello, world");

var button = document.createElement('butto
button.textContent = "Say Hello";
button.onclick = function () {
    alert(greeter.greet());
};

document.body.appendChild(button);

```

TypeScript Compiler

- Geschrieben in TypeScript
- Syntaxanalyse + Typüberprüfung
- Verschiedene ES Targets (3, 5)
- Optionen für z.B. Module
  - CommonJS
  - AMD

```
$ tsc hello.ts

$ tsc --declaration hello.ts

$ tsc --sourcemap hello.ts

$ tsc --module amd hello.ts
```

Definitionen

- Deklarieren von Funktionalität
- Einschränken der Verwendbarkeit
- Hilfestellungen
- Dokumentation

*.d.ts

- TypeScript Definition File
- Nur Kommentare und Definitionen
- Automatische Erstellung möglich
- Einbindung als Referenz
- Wichtige Compiler-Hilfe

Annotationen

- Möglichkeit für Typüberprüfung
- Doppelpunkt trennt Name und Typ
- Automatische Erkennung möglich
- Größter Vorteil in Funktionen

TypeSystem

- Elementare Typen:
  - `any`
  - `boolean`
  - `number`
  - `string`
  - `void`
- Arrays, z.B. `number[]`

Interfaces

- Benutzerdefiniertes Schema
- Definiert Schnittstelle
- Wiederverwendbar
- Erweiterbar (“merging”)
- Hybrid (z.B. Funktion + Eigenschaften)

```
interface Counter {
  (start: number): string;
  interval: number;
  reset(): void;
}
```

Klassen

- Definition wie in ES6
- Modifizierer vorhanden
- Baut Konstruktorfunktion
- Vererbung möglich

```
class Animal {
  constructor(public name: string) {
  }
  move(meters: number) {
    console.log(this.name + " moved " +
meters + "m.");
  }
}
```

```
class Snake extends Animal {
  private length: number;
  constructor(name: string, len: number) {
    this.length = len;
    super(name);
  }
  move() {
    super.move(this.length);
  }
}
```

Weiteres

- Enumerationen (nur `number`)
- Scope Funktionen
- Generics

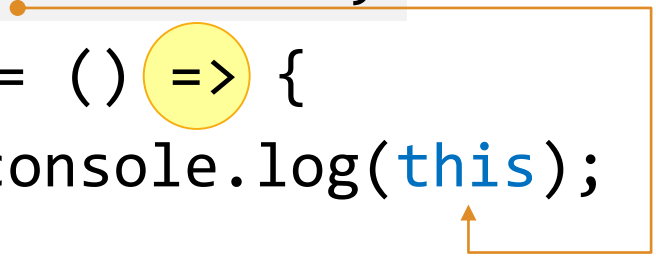
Enumerationen

```
enum Color {
  Red = 1,
  Green,
  Blue
};
var c: Color = Color.Green;
var f = function(color: Color) {
  // ...
};
```

Fat-Arrow Operator

```
var obj: any = { };

obj.f = function() {
      console.log(this);
};
var _this = this;
obj.g = () => {
      console.log(this);
};
```



Generics

```
function identity<T>(x: T): T {
  return x;
}

var num = identity(5);
var str = identity('Hello');
var obj = identity({ a : 3, b : 9 });
```

Module

- Allgemeine Syntax
- `import` (verwenden)
- `export` (bereitstellen)
- Volle Compiler Unterstützung
- Auch erweiterbar

Beispiel

```
import http = require(“http”);

module Validation {
  export interface StringValidator {
    isAcceptable(s: string): boolean;
  }
}
```

IDE Support

- Online Playground
- Direkt in Microsofts DIE
- Eclipse Erweitungen
- Plugins für viele Editoren

Demos

Unterstützung

- Visual Studio (2013)
  - Integriert
  - IntelliSense
  - Direkter Build
  - Code Analysis
  - Definitions
- Brackets
  - Plugin
  - IntelliSense
  - Optionaler Build
  - Definitions

Build Automatisierung

- Integration vorhanden
- VS automatisiert
- Grunt Plugins
- Node Modules

Demo: gulp.js

Fazit

- Ideal für größere Projekte
- Versteckt JS nicht
- Einfacher Einstieg
- Gute Verbreitung
- Mächtiges Tooling

Vielen Dank!

@FlorianRappl

http://florian-rappl.de

https://github.com/FlorianRappl/Mario5TS