ECMAScript 6 /ECMA2015

...löst ECMA5 ab und erweitert dies durch eine Reihe (sinnvoller) Syntaxstrukturen

Aktuell ist die neue Version noch nicht gleichmäßig unterstützt. Daher wird häufig noch auf ECMA5 zurückgegriffen.

Okay, es geht also um JavaScript...

...sollte es nicht um TypeScript gehen?

Keine Angst, das wird es! Weil:

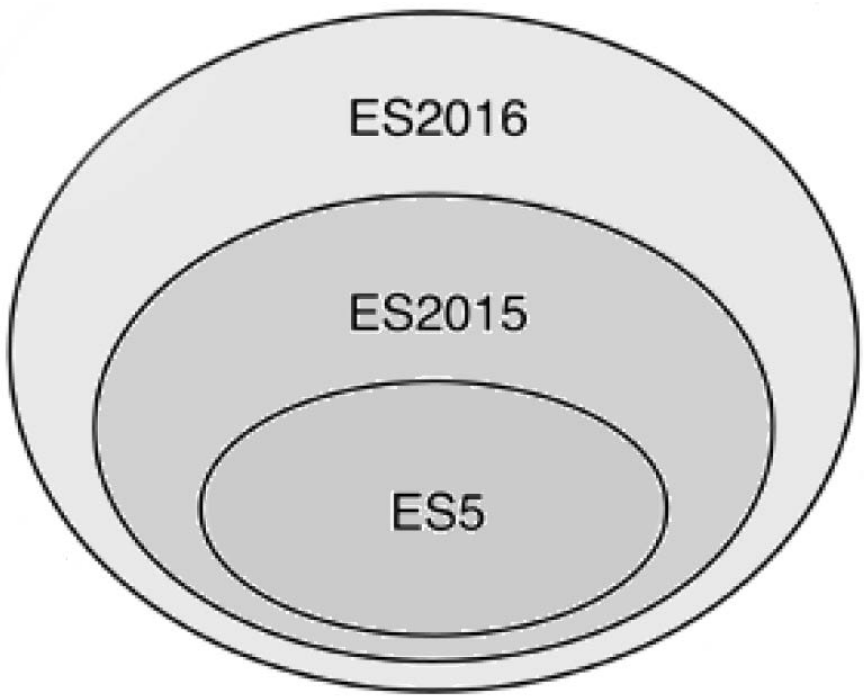
JavaScript === TypeScript

Jedes gültige JavaScript ist gültiges TypeScript.

TypeScript ist jedoch eine *reine Entwicklungssprache*.

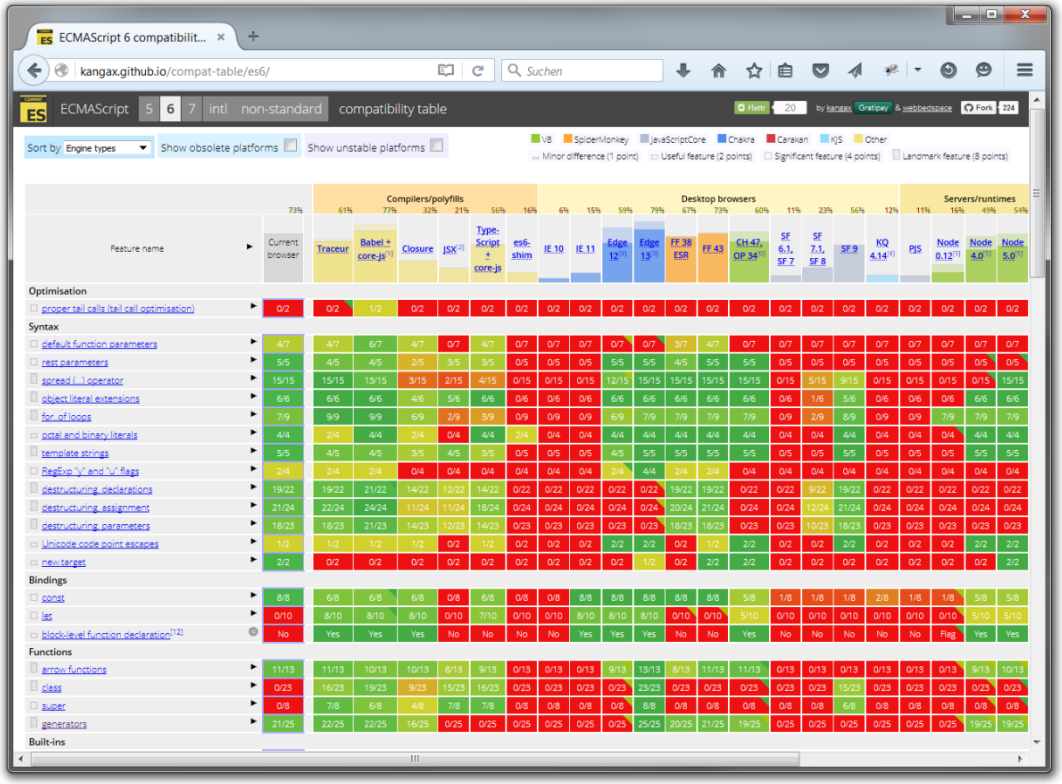
Betrachten wir daher erst in Ruhe JavaScript.

Die ECMA-Zwiebel:



... abwärtskompatibel! Jeder neue Standard baut auf dem vorangehenden auf.

Unterstützung von ECMA2015/2016:



ES6 Supportmatrix: http://kangax.github.io/compat-table/es6

Wie man sieht, weist die Unterstützung Lücken auf

- Aus diesem Grund muss ECMA6-Code (derzeit noch) nach ECMA5 "rückübersetzt" werden.
- **Warum das geht?**
  Alle (beinahe) "neuen" Syntaxstrukturen sind auch in ECMA5 ausdrückbar (wenn auch umständlicher).

Welche Tools kann man als "Transpiler" zum Rückübersetzen verwenden?

- Babel
- Traceur
- **TypeScript-Compiler**

... aha. *Dafür* ist TypeScript also auch gut.

Zurück zu JavaScript (ECMA2015)...

Was ist neu in ECMA2015?

Es gibt Neuerungen in dem Bereichen

- Syntax
- Code Organisation
- Asynchrone Programmierung
- Komplexe Werte
- Allgemeine APIs der Basisobjekte

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren
- Destructuring

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren
- Destructuring
- Template Literale

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren
- Destructuring
- Template Literale
- Defaultparameter für Funktionen

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren
- Destructuring
- Template Literale
- Defaultparameter für Funktionen
- Arrow-Funktionen (“Lambdas”)

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren
- Destructuring
- Template Literale
- Defaultparameter für Funktionen
- Arrow-Funktionen (“Lambdas”)
- Shorthand-Notation für Objektliterale

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren
- Destructuring
- Template Literale
- Defaultparameter für Funktionen
- Arrow-Funktionen (“Lambdas”)
- Shorthand-Notation für Objektliterale
- Neuer Symbol-Typ als "primitive value"

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren
- Destructuring
- Template Literale
- Defaultparameter für Funktionen
- Arrow-Funktionen (“Lambdas”)
- Shorthand-Notation für Objektliterale
- Neuer Symbol-Typ als "primitive value"
- for-of-Schleife

Neue Syntaxstrukturen in ECMA2015:

- Blockscope let und const
- Spread-/ Restoperatoren
- Destructuring
- Template Literale
- Defaultparameter für Funktionen
- Arrow-Funktionen (“Lambdas”)
- Shorthand-Notation für Objektliterale
- Neuer Symbol-Typ als "primitive value"
- for-of-Schleife

Möchte ich im Wesentlichen alles heute (Tag 1) betrachten!

Neu in ECMA2015: Neue komplexe Typen

- Typisierte Arrays
- Maps
- Weak Maps
- Sets
- Weak Sets

Sets haben (quasi) Arraycharakter, **Maps** sind objektähnliche *key-value*-Strukturen. Die "weak"-Versionen sind in bestimmter Hinsicht eingeschränkt (was von Vorteil ist).

Neu in ECMA2015: Neue komplexe Typen

- Typisierte Arrays
- Maps
- Weak Maps
- Sets
- Weak Sets

Sets haben (quasi) Arraycharakter, **Maps** sind objektähnliche *key-value*-Strukturen. Die "weak"-Versionen sind in bestimmter Hinsicht eingeschränkt (was von Vorteil ist).

Etwas speziell.
Lassen wir das beiseite. Okay?

Neu in ECMA2015: Code Organisation

- Iteratoren

Neu in ECMA2015: Code Organisation

- Iteratoren
- Classes

Neu in ECMA2015: Code Organisation

- Iteratoren
- Classes
- Generatoren

Neu in ECMA2015: Code Organisation

- Iteratoren
- Classes
- Generatoren
- Module

Neu in ECMA2015: Code Organisation

- Iteratoren
- Classes
- Generatoren
- Module

Schon spannender. Hier sehen wir genauer hin.

Neu in ECMA2015: Code Organisation

- Iteratoren
- Classes
- Generatoren
- Module

Schon spannender. Hier sehen wir genauer hin.

Zum Teil (vielleicht) heute, Classes an Tag 2, Module an Tag 3!

Neu in ECMA2015: Asynchrone Programmierung

- Promises
- Generatoren (nochmal)

Neu in ECMA2015: Asynchrone Programmierung

- Promises
- Generatoren (nochmal)

An Tag 2! Okay?

Neu in ECMA2015: APIs der Basisobjekte

Erweiterungen der API gibt es für…

- Array
- Object
- Math
- Number
- String

Neu in ECMA2015: APIs der Basisobjekte

Erweiterungen der API gibt es für…

- Array
- Object
- Math
- Number
- String

Betrachten wir nur zum Teil (heute)!

One more thing...

Neu in ECMA2016: Dekoratoren

- Class-Decorator
- Property-Decorator
- Method-Decorator
- Parameter-Decorator
- Decorator-Factory

... womit aspektorientierte Programmierung in JavaScript Einzug hält.

Neu in ECMA2016: Dekoratoren

- Class-Decorator
- Property-Decorator
- Method-Decorator
- Parameter-Decorator
- Decorator Factory

... womit aspektorientierte Programmierung in JavaScript Einzug hält.

Interessant zum Verständnis von Angular. Betrachten wir an Tag 3!