

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:

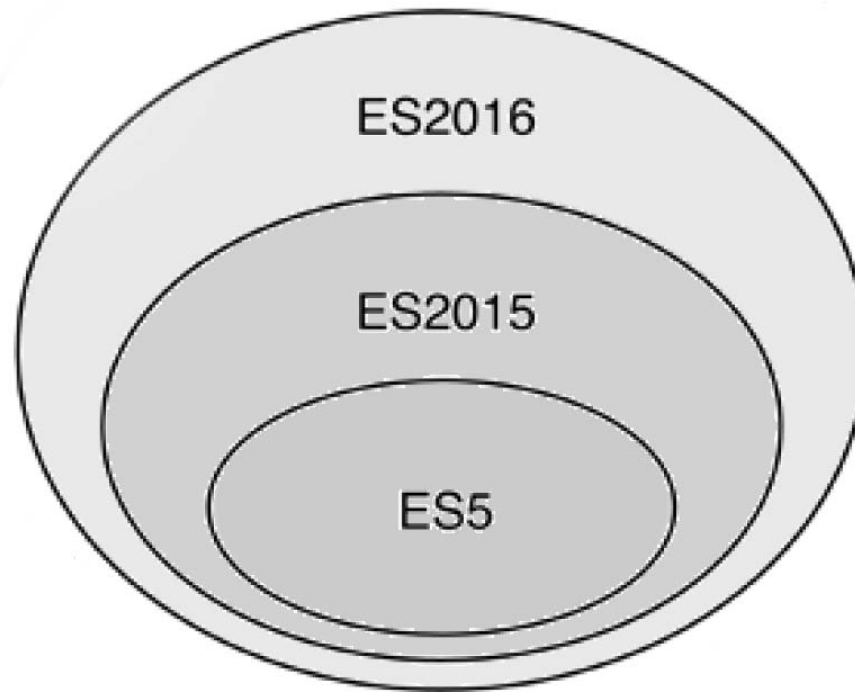
JavaScript === TypeScript

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.

00 ECMA6: Überblick

Unterstützung von ECMA2015/2016:

The screenshot shows a web browser displaying the ES6 compatibility table at kangax.github.io/compat-table/es6/. The table lists various ES6 features and their support status across different browsers and environments. The features are categorized into sections like Optimisation, Syntax, Bindings, Functions, and Built-ins. Each feature has a 'Current browser' column and a grid of colored cells representing support in various browsers. The colors indicate the level of support: green for full support, yellow for partial support, and red for no support. The table also includes a legend for feature types and a search bar.

Feature name	Current browser	Tracur	Babel + core-js	Closure	JSX	Type-Script + core-js	es6-shim	IE 10	IE 11	Edge 12	Edge 13	FF 38	FF 43	CH 42	SE 6.1	SE 7	SE 8	SE 9	KQ 4.14	JS	Node 0.12	Node 4.0	Node 5.0
Optimisation		0/2	0/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
Syntax																							
default function parameters		4/7	4/7	6/7	4/7	0/7	0/7	0/7	0/7	0/7	0/7	3/7	4/7	0/7	0/7	0/7	0/7	0/7	0/7	0/7	0/7	0/7	0/7
rest parameters		5/5	4/5	4/5	2/5	3/5	3/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
spread / ...operator		15/15	13/15	13/15	3/15	2/15	4/15	0/15	0/15	0/15	12/15	15/15	15/15	15/15	0/15	5/15	9/15	0/15	0/15	0/15	0/15	0/15	15/15
object literal extensions		6/6	6/6	6/6	4/6	5/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	0/6	1/6	5/6	0/6	0/6	0/6	0/6	6/6	6/6
for...of loops		7/9	9/9	9/9	6/9	2/9	5/9	0/9	0/9	6/9	7/9	7/9	7/9	7/9	0/9	2/9	8/9	0/9	0/9	0/9	7/9	7/9	7/9
comma and binary literals		4/4	2/4	4/4	2/4	0/4	4/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	0/4	0/4	0/4	4/4	4/4	4/4
template strings		5/5	4/5	4/5	3/5	4/5	3/5	0/5	0/5	4/5	5/5	5/5	5/5	5/5	0/5	0/5	5/5	0/5	0/5	0/5	5/5	5/5	5/5
RegExp "y" and "u" flags		2/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	0/4	2/4	4/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
destructuring declarations		19/22	19/22	21/22	14/22	12/22	14/22	0/22	0/22	0/22	0/22	19/22	19/22	0/22	0/22	9/22	19/22	0/22	0/22	0/22	0/22	0/22	0/22
destructuring assignments		21/24	22/24	24/24	11/24	11/24	18/24	0/24	0/24	0/24	0/24	0/24	20/24	21/24	0/24	0/24	12/24	21/24	0/24	0/24	0/24	0/24	0/24
destructuring parameters		18/23	18/23	21/23	14/23	12/23	14/23	0/23	0/23	0/23	0/23	0/23	18/23	18/23	0/23	0/23	10/23	18/23	0/23	0/23	0/23	0/23	0/23
Unicode code point escapes		1/2	1/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	0/2	2/2	0/2	0/2	2/2	0/2	0/2	0/2	0/2	2/2	2/2
new.target		2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	0/2	2/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2
Bindings																							
const		8/8	6/8	6/8	6/8	0/8	6/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	5/8	1/8	1/8	1/8	2/8	1/8	1/8	5/8	5/8
let		0/10	8/10	8/10	8/10	0/10	7/10	0/10	8/10	8/10	8/10	9/10	9/10	5/10	0/10	0/10	0/10	0/10	0/10	0/10	0/10	5/10	5/10
block-level function declaration		No	Yes	Yes	Yes	No	No	No	No	Yes	Yes	Yes	No	No	Yes	No	No	No	No	No	Flag	Yes	Yes
Functions																							
arrow functions		11/13	11/13	10/13	10/13	8/13	9/13	0/13	0/13	0/13	9/13	13/13	8/13	11/13	0/13	0/13	0/13	0/13	0/13	0/13	0/13	10/13	10/13
class		0/23	15/23	15/23	9/23	15/23	16/23	0/23	0/23	0/23	0/23	23/23	0/23	0/23	0/23	0/23	15/23	0/23	0/23	0/23	0/23	0/23	0/23
super		0/8	7/8	6/8	4/8	7/8	7/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	6/8	0/8	0/8	0/8	0/8	0/8	0/8
generators		21/25	22/25	22/25	16/25	0/25	0/25	0/25	0/25	0/25	0/25	25/25	20/25	21/25	19/25	0/25	0/25	0/25	0/25	0/25	0/25	19/25	19/25
Built-ins																							

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.

00 ECMA6: Überblick

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)
- Async/Await

Neu in ECMA2015: Asynchrone Programmierung

- Promises
- Generatoren (nochmal)
- Async/Await

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)!**

00 ECMA6: Überblick

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!**