Spickzettel („Cheat Sheet“) TypeScript

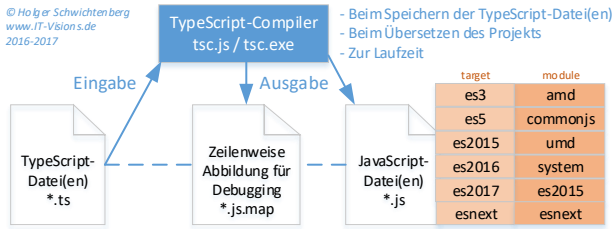
Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

V1.2.1 / 06.09.2017

www.IT-Visions.de
Dr. Holger Schwichtenberg

TypeScript-Compiler

Bezugsquellen: *npm install -g typescript* oder per Nuget *Install-Package Microsoft.TypeScript.Compiler* oder TypeScript Editor-Add-Ins (Visual Studio u.a.): *www.typescriptlang.org*
Installationsverzeichnis (bei Visual Studio):
C:\Program Files (x86)\Microsoft SDKs\TypeScript



Compileroptionen festlegen in

- MSBuild-XML (tlw. im Visual Studio-Projektdialog)
- *tsconfig.js*-Datei (erstellen mit *tsc --init*)
- Kommandozeilenparametern, z.B.: Kompilieren von zwei Dateien immer direkt beim Speichern
  tsc datei1.ts datei2.ts --watch --target es5 --module amd

JavaScript-Syntax

Alle JavaScript-Syntaxkonstrukte (Funktionen, Bedingungen, Schleifen, Objekte, Operatoren etc.) sind gültig in TypeScript.

```
function calc(art, x, y)  {
 print("calc:x=" + x + ",y=" + y);
 if (x === null || y === null) return "Fehler: null";
 if (x === undefined || y === undefined) return
"Fehler: undefined";
 // Objekt für Rückgabe erstellen
 var e = { zeitpunkt: new Date(), wert: 0 };
 switch (art) {
  case 1:
   e.wert = x;
   break;
  case 2:
   e.wert = y;
   break;
  default:
   return 0;
 }
 for (var i = 1; i <= x; i++) {
  e.wert += y + x + 2*i;
 }
 var j = 0;
 while (j < x && e.wert < 10000) {
  e.wert += j++;
}
 do {
  e.wert += j--;
} while (j > -10)
 return e;
}
```

Variablendeklarationen: var vs. let vs. const

Datentypen: number, string, boolean, object (seit v2.2), any, void, Array<T>, Date, Object, eigene Schnittstellen, Klassen und Aufzählungstypen

```
if (true) {
 var x: number = 5;
}
print("x=" + x); // Ausgabe x=5 :-( nur Warnung
if (true) {
 let y: string = "Holger Schwichtenberg";
}
print("y=" + y); // Fehler!
const z: boolean = true;
z = false; // Fehler!
```

Funktionsparameter mit expliziten Typen

```
// s ist optionaler Parameter
// c ist Parameter mit Standardwert
function print(s?: string, count: number = 1): boolean
{
 if (!s) return false;
 for (var i = 0; i < count; i++) {
  // Ausgabe mit jQuery
  $("#content").append(s + "<br>");
  // Ausgabe im Consolenfenster
  console.log(s);
 }
 return true;
}
```

Typprüfung und Typkonvertierung

```
var eingabe: any = 5;
if (typeof eingabe === 'number') {
  var zahl1 = (<number>eingabe) + 1;
  // alternativ:
  var zahl2 = (eingabe as number) + 2;
}
```

Wahr und Falsch

TypeScript hat wie JavaScript eine weite Definition von true und false.

```
// Truthy
if (true) print("false ist true");
if ({}) print("Leeres Objekt ist true");
if ([]) print("Leeres Array ist true");
if (function(){ }) print("Leere Funktion ist true");
if (42) print("42 ist true");
if ("xy") print("Nicht-leere Zeichenkette ist true");
if (-42) print("-42 ist true");
if (Infinity) print("Infinity ist true");
// Falsy
if (!false) print("false ist false");
if (!null) print("null ist false");
if (!undefined) print("undefined ist false");
if (!obj) print("Nicht-initialisiertes Objekt ist
false");
if (!0) print("0 ist false");
if (!NaN) print("NaN ist false");
if (!'') print("Leerstring ist false");
if (!"") print("Leerstring ist false");
```

Union Types (seit v1.4)

```
var mitarbeiterID: number | string;
mitarbeiterID = 123; // OK
mitarbeiterID = "A123"; // auch OK
mitarbeiterID = true; // Fehler!
```

Typaliase (seit v1.4)

```
type ZahlenArray = Array<number>;
var lottozahlen : ZahlenArray;
```

Lambda-Ausdrücke

```
// Funktionstyp deklarieren
var add: (a: number, b: number) => number;
// Funktionstyp implementieren
add = (a, b) => a + b;
// Funktion nutzen
const c = add(1, 2);
// Deklaration und Implementierung zusammen
var print = (s: string) => console.log(s);
// Funktionen als Parameter
function execMathFunc(f: (a: number, b: number) =>
number, a: number, b: number) {    return f(a, b);  }
var e = execMathFunc(add, 2, 3);
```

Template Strings (String Interpolation) (seit v1.4)

```
var e: string = `Kunde #${k.id} heißt ${k.name}.`;
```

Spickzettel („Cheat Sheet“) TypeScript

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

V1.2.1 / 06.09.2017

www.IT-Visions.de
Dr. Holger Schwichtenberg

Namensräume und Schnittstellen

```
namespace CheatSheet.Interfaces {
 export interface IKontakt {
  id: number;
  name: string;
  geprueft: boolean;
  erfassungsdatum: Date;
  toString(details: boolean): string; } }
```

Klassen und Klassenmitglieder

Standard ist *public*! Konstruktoren dürfen nicht überladen werden! *this* ist **zwingend** für Zugriff innerhalb der Klasse zu verwenden! Klassen und Methoden dürfen *abstract* sein (seit v1.6).

```
namespace CheatSheet.GO {
/** Kontakt mit id, name, ort
 * @autor Dr. Holger Schwichtenberg
 */
export class Kontakt implements Interfaces.IKontakt {
 // ---------- Properties ohne Getter/Setter
 public ort: string;
 protected interneID: number;
 private _erfassungsdatum: Date;

 // ---------- Properties mit Getter/Setter
 get erfassungsdatum(): Date {
  return this._erfassungsdatum;
 }
  set erfassungsdatum(erfassungsdatum: Date) {
  this._erfassungsdatum = erfassungsdatum;
 }
 /**
  * @param id Kundennummer
  * @param name Kundenname
  * @param ort Kundenwohnort
  */
 constructor(
  public id: number,
  public name: string) {
  this.erfassungsdatum = new Date();
  this.id = id; // Diese Zeile ist überfluessig !!!
  Kunde.Anzahl++;
 }
 // Öffentliche Methode
 toString(details: boolean = false): string {
  var e: string = `${this.id}: ${this.name}`;
  if (details) e += …`;
  return e;
 }
 // Statisches Mitglied
  static Anzahl: number;
} }
```

Aufzählungstypen (Enums)

```
enum KundenArt { A = 1, B, C }
enum KundenArt24 { // seit ab v2.4
   A = "sehr gut", B = "gut", C = "schlecht" }
```

Vererbung

Ein Konstruktor wird **vererbt**, wenn er nicht überschrieben wird. Ein expliziter Konstruktor in einer abgeleiteten Klasse **muss** den Konstruktor der Basisklasse aufrufen mit *super()*.

```
export class Kunde extends Kontakt {
 public KundenArt: KundenArt;
 constructor(id: number, name: string, umsatz: number)
 {
  super(id, name);
  this.KundenArt = KundenArt.C;
  if (umsatz > 10000) this.KundenArt = KundenArt.A;
  if (umsatz > 5000) this.KundenArt = KundenArt.B;
 }
 toString() { return `Kunde ${super.toString()}`; }
}
```

Klasse instanziieren und nutzen

```
import G = CheatSheet.GO;
…
var k = new G.Kunde(123, "H. Schwichtenberg", 99.98);
var ausgabe1 = k.toString();
// Typprüfung
if (k instanceof CheatSheet.GO.Kunde) { print("…"); }
// Destructuring von Objekten
let { id, name } = k;
var ausgabe2: string = `${id}: ${name}`;
```

Generische Klassen

```
export class Verbindung<T1 extends Kontakt, T2 extends
 Kontakt> {
 constructor(public k1: T1, public k2: T2) {
 }
 public toString(): string {
  return `${this.k1.name} und ${this.k2.name}`;
 }
}
…
var v = new CheatSheet.GO.Verbindung<GO.Kontakt,
   GO.Kunde>(k1, k2);
var ausgabe2 = v.toString();
```

Strukturelle Typäquivalenz (Duck Typing)

```
export class Person {
 public id: number;
 public name: string; }

export class Mitarbeiter {
 public id: number;
 public name: string;
 public einstellungsdatum: Date; }

var m = new CheatSheet.GO.Mitarbeiter();
m.id = 1;
m.name = "Holger Schwichtenberg";
// erlaubt: Person und Mitarbeiter passen zueinander!
var p: CheatSheet.GO.Person = m;
```

Arrays

```
// Deklaration
var lottozahlen: Array<number>; // oder number[]
// Initialisierung
lottozahlen = [11, 19, 28, 34, 41, 48, 5];
// Schleife über alle Elemente
for (var z of lottozahlen) {
  print(z);
};
// Destructuring (seit v1.5)
let [z1,z2,z3,z4,z5,z6,zusatzzahl] = lottozahlen;
print(`Zusatzzahl: ${zusatzzahl}`);
```

Spread-Operator (seit v2.1)

```
let obj1 = { Name:"Holger Schwichtenberg" }
let obj2 = { firma:"www.IT-Visions.de", Ort:"Essen" }
let obj12 = { ...obj1, ...obj2 };
out(`${obj12.Name} arbeitet für ${obj12.firma}`);
```

Tupel (seit v1.3)

```
var PersonTupel: [number, string, boolean];
PersonTupel = [123, "Holger S.", true];
var personName = PersonTupel[1];
```

async/await (seit v1.7 bzw. v2.1 für ältere Browser)

```
// erfordert Polyfill: npmjs.com/package/es6-promise
function randomDelay(milliseconds: number):
Promise<void> {
   return new Promise<void>(resolve => {
    for (let i = 0; i < Math.random() * 7; i++) {
     setTimeprint(resolve, milliseconds);
     console.log(".");
    }
   });
  }
async function aktion(text: string, func: (p: number)
=> Promise<void>, ms: number) {
```

Spickzettel („Cheat Sheet“) TypeScript

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de) V1.2.1 / 06.09.2017


```
   print("Starte: " + text);
   await func(ms);
   print("Beendet: " + text);
  }

aktion("Aufgaben laden", randomDelay, 900);
aktion("Kontakte laden", randomDelay, 700);
aktion("Termine laden", randomDelay, 500);
```

Dekoratoren / Annotationen (seit v1.5)

```
/// <reference path="reflect-metadata/reflect.ts" />
// einfacher Klassendekorator ohne Factory
function klassendekorator(klasse) {
 console.log("# Klassendekorator: " + klasse);
}
// Property-Dekorator mit Factory
function range(min: number, max: number) {
 return function (target: any, name: string,
    descriptor: TypedPropertyDescriptor<number>) {
  let set = descriptor.set;
  descriptor.set = function (value: number) {
   let type = Reflect.getMetadata("design:type",
     target, name);
  if ((value < min || value > max)) {
    console.error(`Fehler bei ${name}: Wert ${value} <
${min} oder >${max}!`);
    }    else     {
    console.log(`OK bei ${name}: Wert ${value} >${min}
 und <${max}!`);
   }
  }
 }
}
```

Dekoratoren anwenden

```
@klassendekorator
class Lottoziehung {
 private _zahl: number;
 @range(1, 49)
 set zahl(value: number) { this._zahl = value; }
 get zahl() { return this._zahl; }
 constructor(z: number) { this.zahl = z; }
}
```

Referenzen

Referenzieren von anderen TypeScript-Dateien:
/// <reference path="modulA.ts" />
Referenzieren von TypeScript-Deklarationsdateien:
/// <reference path="/typings/jquery.d.ts" />

Module (AMD, CommonJS, UMD, System, ES2015)

TypeScript-Dateien bilden Module. Compileroptionen legen Modulformat fest. Zur Laufzeit wird eine passende Bibliothek benötigt. *export* kennzeichnet zu exportierende Klassen oder Funktionen. Nutzung mit:
Import einer Klasse: *import {KlasseA} from './ModulA'*
Import aller Exporte: *import * as modA from './ModulA'*

Links

Offizielle Website: *www.typescriptlang.org*
Quellcode: *github.com/Microsoft/TypeScript*
Beispiele: *github.com/Microsoft/TypeScriptSamples*

Über den Autor

Dr. Holger Schwichtenberg gehört zu den bekanntesten Experten für die Programmierung mit Microsoft-Produkten in Deutschland. Er hat zahlreiche Bücher zu .NET und Webtechniken veröffentlicht und spricht regelmäßig auf Fachkonferenzen wie der BASTA. Sie können ihn und sein Team für Schulungen, Beratungen und Projekte buchen.
E-Mail: **anfragen@IT-Visions.de**
Website: **www.IT-Visions.de**
Weblog: **www.dotnet-doktor.de**