Java Typen in TypeScript nutzen

Dragan Zuvic

Karlsruher Entwicklertag 2017

Dragan Zuvic

Full Stack Entwickler

\+ PL, Architektur, Design, OP ...

Java, Scala, Kotlin, TypeScript

@dzuvic

thecodecampus</>

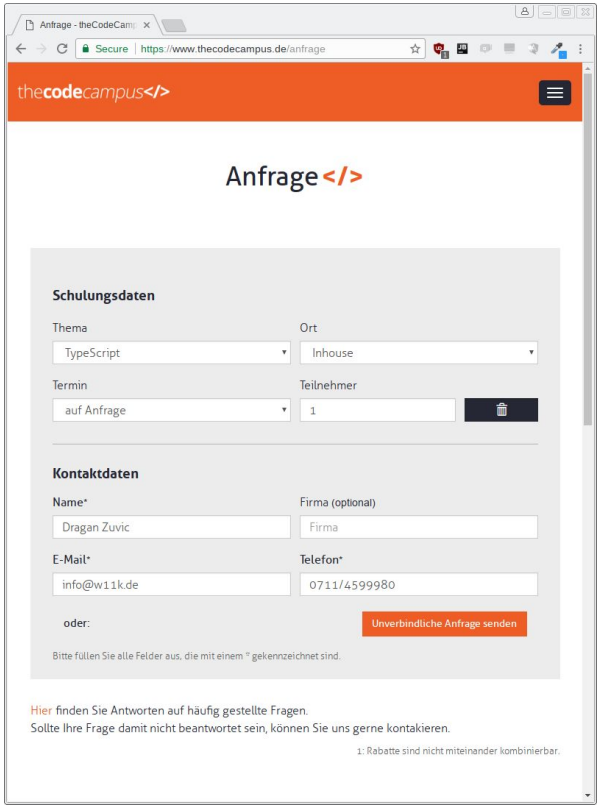
Agenda

Java ∞ TypeScript

Das Problem...

```java
public class Person {
    private String name;
    private LocalDate date;
    private String company;
    private Sex sex;
    private String email;
    private String phone;
}
```

List<Person>



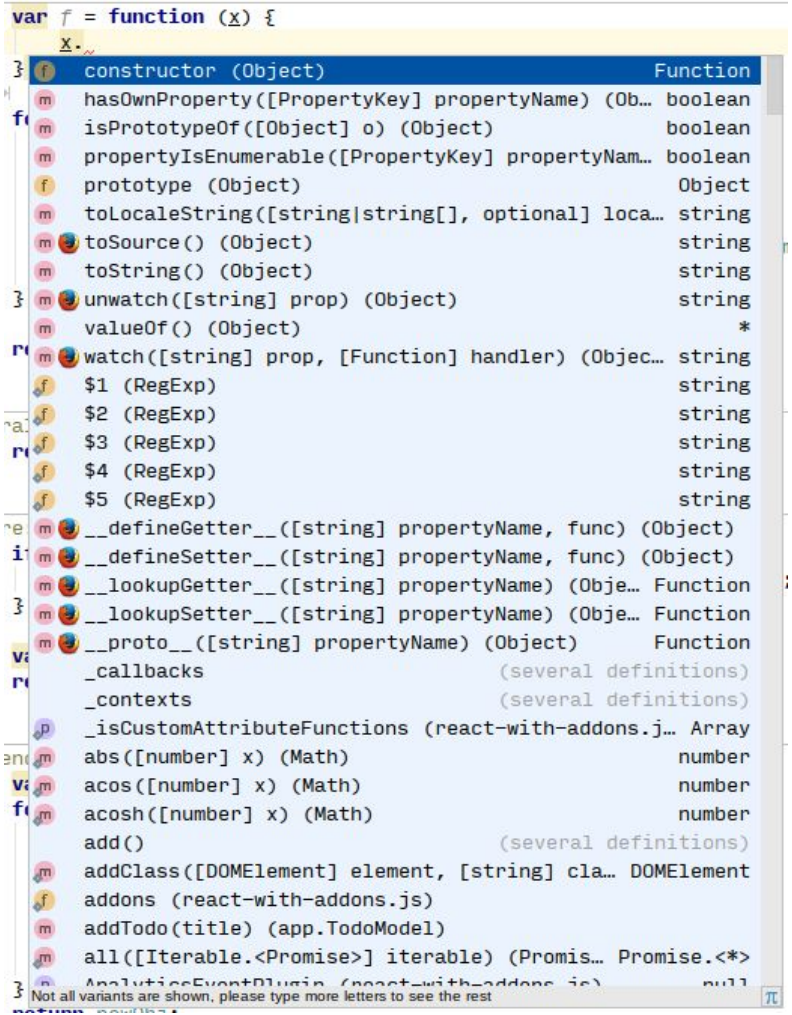
Ein Modell - Zwei Applikationen

Code Completion & Typ-Sicherheit

Grund: Mangelnde Typisierung.
Ist x es ein ?

➢ string, number,
Date,window,[], Promise,
null, symbol, Object, ...

Typsystem erforderlich..



Warum TypeScript?

- Gradueller Typsystem Aufsatz für ES 2015
- Existierender JS Code == gültig
  - Strukturelle Typisierung
  - any =”JS Typ“ ≠Object
  - Union-Types
  - **Ambient Types**
  - `@types`

Beispiele: Angular, NativeScript

TypeScript	Java
boolean number string null undefined symbol	int long double float boolean null
Object String [] function	Object String Array List Function BigDecimal
Array<T>	List<T>
Array<number>	Set<? super Number>
enum	Enum<E extends Enum<E>>
@Decorator (Runtime)	@Annotation (Runtime & Compile)
A || B - A && B - A || null	<T extends A & B> AException | BException

Was brauchen wir denn wirklich?

```
public class Person {

   private String name;

   private LocalDate birthdate;

   private Sex sex;

  ...

}
```

```
{
   "name" : "Immanuel Kant",
   "birthdate" : "1724-04-22",
   "sex" : "M"
}
```

Erzeugen von

Ambienter Typ Deklaration:

```
export interface Person
{
 birthdate: string;
 sex: any;
 name: string;
}
```

```
@Component({
  selector: 'app-list',
  templateUrl: './list.component.html',
  styleUrls: ['./list.component.css']
})
export class ListComponent implements OnInit {

  persons:Person[]=[];

  persons$: Subject<Person[]> = new Subject();

  constructor(private http: Http) {

    this.http.get("/api/person/list").map( response =>
      response.json()
    ).subscribe(
      thePersons => this.persons$.next(thePersons)
    );

    this.persons$.subscribe( (thePersonList : Person[]) =>
      this.persons =
        thePersonList
          .map (x => x.)
    );

  }

  ngOnInit() {

  }

}
```

```
name exampleApi.Person (example-api.d.ts, src/generated/exampleapi)  string
sex exampleApi.Person (example-api.… any
birthdate
const                    const name = expr
let                        let name = expr
not                                !expr
par                              (expr)
typeof                     typeof expr
var                      var name = expr
Did you know that Quick Definition View (Ctrl+Shift+I) works in completion lookups as well? >>
```

Typ zur Laufzeit

- Mapper: Config @ Runtime
- Laufzeit T ➜ TypeScript T
- Type Emitter
  - 💣 Config @ Compile time

IMHO: Config nahe am Code


```
"birthdate": {
 "year": 1724, "month": "APRIL",
 "monthValue": 4, "dayOfMonth": 22,
 "dayOfWeek": "SATURDAY",
 "era": "CE", "dayOfYear": 113,
 "leapYear": true,
 "chronology": {
    "calendarType": "iso8601",
    "id": "ISO" }
}
```

Lösungen....

JSON → TS

Typ Inferenz aus JSON Struktur

Unpraktisch

- $\forall T \in \text{Java} \lesssim_{\rightarrow} \text{JSON}$

`{ x: 0 }` → x? oder x

`{x:{z:0}, y:{z:0}}`

UML

Pflege eines Models und Implementation eines TypeScript Generators

- Es nutzt keiner mehr

Marian Petre “UML in practice” ICSE'13: “... *that practitioners take a broad view of what constitutes ‘modeling’, [...] The majority of those interviewed simply do not use UML [...]*”

DSL

Implementation DSL + Generatoren

Für diesen Use-Case überwiegen die Probleme

- :stimmiges :DSL 🕐
- Pflege
  - Erweiterung?
  - Wachstum?
  - Ghetto (Fowler)?

AST

Parsen von Java Code ➜ Extraktion

Bestehende Bibliotheken

- ANTLR
- JS Lösung: PEG.js
- Annotation Processor

0xCAFEBABE

Java Byte Code ➜ TypeScript

- Kein @javadoc
- Typ-Parameter ?

jsweet & GWT	Umgebung für Entwicklung in Java → JS / TypeScript	*Laufzeitumgebung für Java Typen*
typescript-generator	Maven & Gradle Plugins Type Emitter Java → TypeScript	*Bytecode*
jtsgen	Annotation Processor Type Emitter Java → TypeScript	☆

Annotation Processor

jtsgen

https://github.com/dzuvic/jtsgen

README.md

jtsgen: Convert Java Types to TypeScript

tl;dr Enable code completion of Java types in your TypeScript project.

Annotations: Download 0.1.2 Processor: Download 0.1.2 build passing

Features

This project emits TypeScript ambient types from Java sources. `jtsgen` is implemented as an annotation processor, therefore it should be easily integrated in your current build infrastructure. Usually there are no other plugins required for your build system (maven, gradle).

This project is still in development, so major changes are still on the way and might break using it. Therefore either submit an issue on github or a pull request if you want a specific feature being implemented.

Currently the following features are supported:

- Emitting types for `@TypeScript` annotated Java classes and interfaces
- Ignoring a type using `@TSIgnore` annotation
- creating a module with corresponding package.json. The name is constructed if not configured
- Configuration of the JavaScript / TypeScript module using the `@TSModule` annotation, e.g. the module name or the author of the exported ES module
- Configurable type conversion and exclusion using the `@TSModule` annotation. It also supports type parameters, e.g. a `ImmutableList<T>` can be be mapped to your own type `ImmutableArray<T>` with a corresponding (mapped) type `T`
- Java package as typescript name space
- converting getter/setter to TypeScript types
- `readonly` if no setter is found
- Name Space mapping to minimize the TypeScript name spaces. It can be configured or calculated.

Requirements: The annotation processor only depends on the JDK. Only JDK 8 is currently supported.

Usage

The jtsgen annotation processor registers itself using the ServiceLoader protocol. Therefore when the processor is available on the compile or annotation class path it should be automatically invoked when compiling the annotated Java classes. Any Java class, interface or enum with the annotation `@TypeScript` will be converted, e.g.:

```
@TypeScript
public interface InterFaceSample {
    int getSomeInt();
    String getSomeString();
}
```

Marker Setzen

```
@TypeScript
public class Person {
   private String name;
   private LocalDate birthdate;
   private Sex sex;
   public String getName() {
       return name;
   }
...
```

Processor & Build-Prozess

Steuerung & Ausführung: Java Compiler

- Keine Speziellen Plugins
- Update On Save (im Prinzip)

Autodiscovery via ServiceLoader - Klassenpfad / Processing-Pfad

Ansonsten: `javac -processor className`

Beispiel Gradle

```
buildscript { dependencies {...} }
apply plugin: 'cz.habarta.typescript...
generateTypeScript {
 jsonLibrary = 'jackson2'
 classPatterns = [  'w11k.mod.*' ]
   outputFile = '../client/mod.d.ts'
   outputKind = 'global'
   namespace = 'ProjektModel'
}
```

typescript-generator

```
dependencies {
  compileOnly
     “jtsgen:jtsgen-processor:0.1.2"
}
compileJava {
   options.compilerArgs = [ "-s",
                            tsOutDir ]
}
```

jtsgen

Config: CLI, Datei, Package?

```
@TSModule(moduleName = "exampleApi", author = "Bürger
Perle",
        outputType = OutputType.DECLARED_NAMESPACE,
        customTypeMappings = {
                "java.time.LocalDate -> string",
                "java.lang.Number -> number"
        })
package dz.jtsgen.example;

import dz.jtsgen.annotations.OutputType;
import dz.jtsgen.annotations.TSModule;
```

Annotation Processor

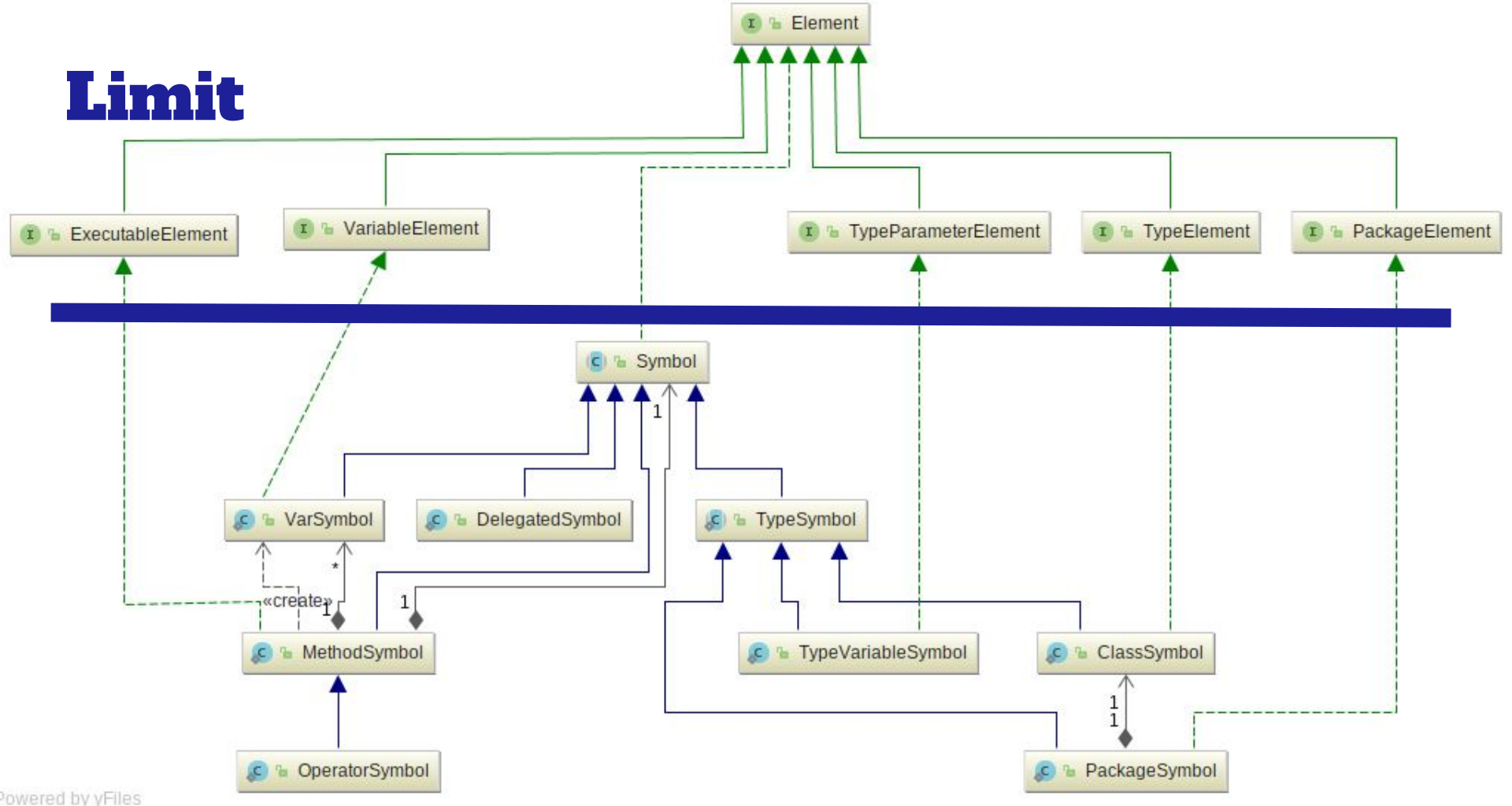
Compile Plugin “Light”

API: **ausschließlich** Deklarationen

```
boolean process(Set<? extends TypeElement>,
                RoundEnvironment)
```

- Java Compiler *Callback*
  - Bausteine: `Element & TypeMirror`
  - Reihenfolge nicht garantiert
- Hinweis: CheckNames Beispiel von J. Darcy

Limit



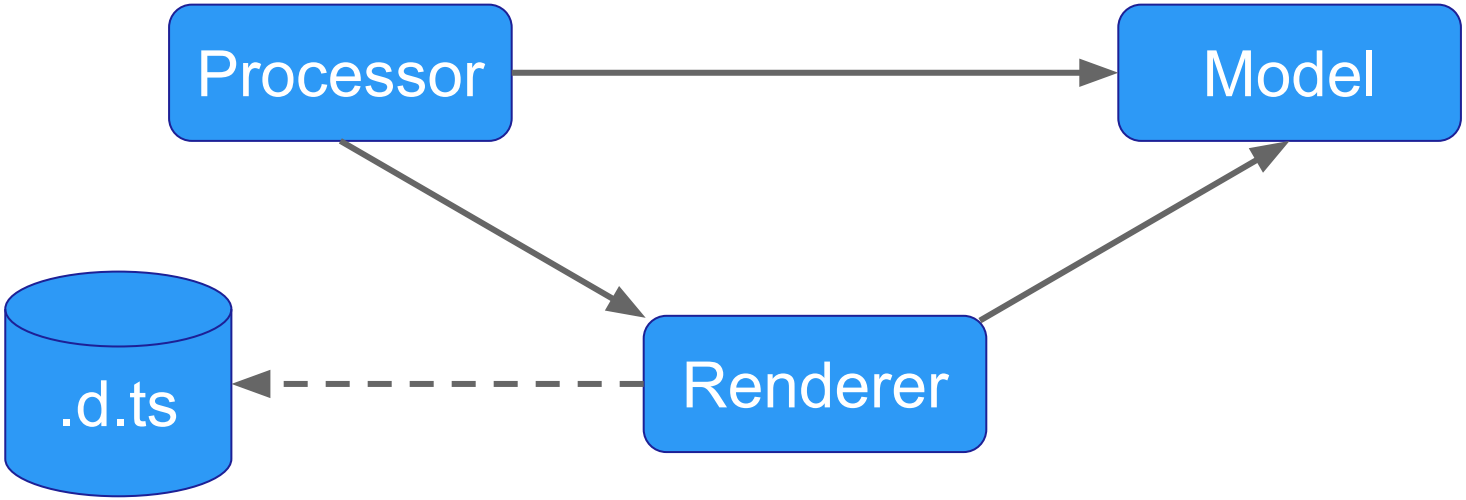
Einschränkung → MVC

Dateien = Package Friendly

Keine Änderung an der Java Klasse

Testen? Von Google: compile-testing

Datei darf nur **einmal** erzeugt werden



Visitor :-(

Absteigen am AST

Abhängig v. Element

- `if ( e.getKind()... )`
- ~~Pattern-Matching~~
- Visitor-Pattern

```
/*
* Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
* ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
*/
public interface ElementVisitor<R, P> {

   R visit(Element e, P p);
   R visit(Element e);
   R visitPackage(PackageElement e, P p);
   R visitType(TypeElement e, P p);
   R visitVariable(VariableElement e, P p);
   R visitExecutable(ExecutableElement e, P p);
   ...
}
```

Multi-Projekt Integration

Export / Import als *Modul*

Projekte entkoppelt

Modul System Auswahl

Update via npm ➔ Versionierung

```
import Person = exampleApi.Person;
```

```
"dependencies": {
  "@types/X": "file:../api/build/X"
}
```

Export / Import als Datei

Projekte Gekoppelt

Einfach Umzusetzen

```
import Person = exampleApi.Person;
```

➔ Pferdefuß: typesRoot != wie Dokumentiert

Fazit

Jedes Problem trägt ein Geschenk in der Hand

Inkrementelle Konvertierung ?

Not Null & readonly → Typescript (Inferenz?)

Datei-Namen = Package-Name: ~~`01-Modülchen`~~

Fremde Annotation (JAX-RS), Klassen, Konstanten & Vererbung

TypeScript Compiler fixiert @types

Kotlin Spezifisch: KAPT + Source Target = ⛔

quod erat demonstrandum

- ✓ Java Typen ➜ TypeScript via Annotation
- ✓ Konfigurierbare Abbildung der Typen
- ✓ Konfiguration nahe am Code
- ✓ Direktes Einbinden der `d.ts` Datei möglich

THE WEB ENGINEERS

Einbau - Ausblick: Ausbau

@dzuvic

https://w11k.de

Diese Folien: goo.gl/lvNCMn

github.com / dzuvic / jtsgen

github.com / dzuvic / jtsgen-example

Referenzen

Petre, Marian (2013): “UML in practice.”; 35th International Conference on Software Engineering (ICSE 2013), 18-26 May 2013, San Francisco, CA, USA, pp. 722–731.

Fowler, Martin (2010): “Domain Specific Languages”; Addison-Wesley Professional

typescript-generator: https://github.com/vojtechhabarta

compile-testing: https://github.com/google/compile-testing/releases

jtsgen Default Mapping

- Primitive: double, long, int, float -> number ; boolean -> boolean
- Referenz-Typen wie folgt:

```
java.lang.Object -> any
java.lang.Void -> Void
java.lang.String -> string
java.lang.Integer -> number
java.lang.Double -> number
java.lang.Number -> number
java.lang.Long -> number
java.lang.Short -> number
java.lang.Boolean -> boolean
java.util.Collection<T> -> Array<T>
java.util.Map<U,V> -> Map<U,V>
```