

TypeScript – Teil 2

Narrowing, fortgeschrittene Typen & Workshop

Informatik – 3. Semester

1 Type Inference & Narrowing

3 Mini-Workshop

2 Fortgeschrittene Typen

4 Wrap-up & Ausblick

Type Inference und Type Narrowing

- TypeScript leitet Typen aus Literalen, Ausdrücken und Rückgabewerten ab
- Hilft, Boilerplate zu reduzieren, solange der Compiler genug Kontext hat

```
const count = 0;  
const names = ['Ada'];  
const pair = [1, 'x'];
```

Explizit typisieren, wenn Intent wichtig ist (z. B. `const status: 'open'`)

- Bei `any`-Durchreichung geht Info verloren
- `let result;` → `any` bis erste Zuweisung
- Funktion ohne Rückgabetyt kann `any` ausschwemmen

```
function parse(data) {  
    return JSON.parse(data);  
}  
  
const payload = parse('{}');
```

- Ziel: Aus einer Union einen spezifischeren Typ ableiten
- Strategien: `typeof`, `instanceof`, Equality Checks

```
function formatValue(value: string | number) {  
  if (typeof value === 'string') {  
    return value.toUpperCase();  
  }  
  return value.toFixed(2);  
}
```

- Prüft, ob eine Eigenschaft auf einem Objekt existiert

```
type Success = { data: string };
type Failure = { error: string };
type Result = Success | Failure;

function handle(result: Result) {
  if ('data' in result) {
    console.log(result.data);
  } else {
    console.error(result.error);
  }
}
```

- Funktioniert mit Klassen und Konstruktorfunktionen

```
class FileTracker {  
  constructor(public path: string) {}  
}  
  
function log(item: FileTracker | string) {  
  if (item instanceof FileTracker) {  
    console.log('tracking', item.path);  
  } else {  
    console.log('raw path', item);  
  }  
}
```

- Eigene Funktionen, die eine Type-Predicate-Signatur haben

```
type TeamMember = { name: string; skills?: string[] };

function hasSkills(member: TeamMember): member is TeamMember & { skills: string[] } {
    return Array.isArray(member.skills) && member.skills.length > 0;
}

function describe(member: TeamMember) {
    if (hasSkills(member)) {
        return `${member.name}: ${member.skills.join(', ')}`;
    }
    return `${member.name}: Skills TBD`;
}
```

- `as const` konserviert Literaltypen
- Generics mit Default (`<T = string>`) erleichtern APIs
- Enums/const Assertions halten Werte stabil

```
const config = {  
  env: 'prod',  
  retries: 3,  
} as const;  
  
type Config = typeof config;
```

```
type User = { id: number; name: string };  
type Team = User | User[];  
  
function getFirst(entity: Team) {  
    // TODO: Type Guards ergänzen  
}
```

- Ergänzt Narrowing, sodass getFirst immer einen User zurückgibt
- Schritt 1: Prüfen auf Array `Array.isArray`
- Schritt 2: Optional benutzerdefinierten Guard schreiben

5 min

```
function getFirst(entity: Team): User {  
  if (Array.isArray(entity)) {  
    return entity[0];  
  }  
  return entity;  
}
```

1 Type Inference & Narrowing

3 Mini-Workshop

2 **Fortgeschrittene Typen**

4 Wrap-up & Ausblick

die wichtigsten TS Konstrukte

- Literaltypen beschränken Variablen auf konkrete Werte
- Oft genutzt für Status, Aktionen, Feature-Flags

```
type Theme = 'light' | 'dark';  
const currentTheme: Theme = 'light';  
  
function setTheme(theme: Theme) {  
  document.body.dataset.theme = theme;  
}
```

```
type Shape =  
  | { kind: 'circle'; radius: number }  
  | { kind: 'rectangle'; width: number; height: number };  
  
function area(shape: Shape) {  
  switch (shape.kind) {  
    case 'circle':  
      return Math.PI * shape.radius ** 2;  
    case 'rectangle':  
      return shape.width * shape.height;  
  }  
}
```

- Typen über vorhandene Keys generieren

```
type Optional<T> = {  
  [K in keyof T]?: T[K];  
};  
  
type User = { id: number; name: string; email: string };  
type UserDraft = Optional<User>;
```

- `Partial<T>` – macht alle Felder optional
- `Pick<T, K>` – wählt Felder aus
- `Record<K, T>` – Map von Keys zu Werten
- `Readonly<T>` – verhindert Mutation

```
type User = { id: number; name: string; email: string };  
type UpdateUser = Partial<User>;  
type UserPreview = Pick<User, 'id' | 'name'>;  
type Permissions = Record<'read' | 'write', boolean>;
```

- Typentscheidung basierend auf einer Bedingung `extends ? A : B`

```
type ResponseType<T> = T extends Promise<infer U> ? U : T;  
  
type A = ResponseType<Promise<string>>;  
type B = ResponseType<number>;
```

- `any` : Deaktiviert Typprüfung – schnelle, aber unsichere Lösung
- `unknown` : Typ muss vor Nutzung eingegrenzt werden

```
function handleInput(value: unknown) {  
  if (typeof value === 'string') {  
    return value.trim();  
  }  
  throw new Error('Unsupported type');  
}
```

- TypeScript hat globale Utility Types, die auf Typen angewandt werden können
 - Discriminated Unions sind perfekt, um einen Typen zu “narrowen”
1. Welcher Utility Type eignet sich für eine Map von Rollen zu Rechten?
 2. Wie signalisiert ihr bei Discriminated Unions den Typ?
 3. Welcher Conditional Type entpackt `Promise<T>` ?

1 Type Inference & Narrowing

2 Fortgeschrittene Typen

3 Mini-Workshop

4 Wrap-up & Ausblick

Modul strikt typisieren

- Kleines Modul (Task-Board) gemeinsam von any zu sauberen Typen führen
- Fokus: Datenmodell, Service-Funktionen, Fehlerobjekte
- Output: Pull-Request-ready Typdefinition + kurze Tests

```
// board.js
let tasks = [];

function addTask(task) {
  tasks.push(task);
}

function findTask(id) {
  return tasks.find(t => t.id === id);
}
```

```
type Task = { id: string; title: string; status: 'open' | 'done'; assignee?: string };
type Board = { tasks: Task[] };

function addTask(board: Board, task: Task): Board {
  return { ...board, tasks: [...board.tasks, task] };
}

function findTask(board: Board, id: string): Task | undefined {
  return board.tasks.find(task => task.id === id);
}
```

- 1 Type Inference & Narrowing
- 3 Mini-Workshop

- 2 Fortgeschrittene Typen
- 4 Wrap-up & Ausblick**

Wie geht's weiter?

- Typsystem = Team-Superpower: Fehlertypen rutschen vom Prod-Log ins IDE-Panel
- Kombination aus Inferenz + gezielten Annotationen liefert Tempo und Sicherheit
- Generics & Utility Types helfen, Code zu teilen statt zu kopieren

- Welches Konzept wollt ihr sofort im Projekt einsetzen?
- Wo braucht ihr noch Support?

2 min

Q&A

+ Feedbackrunde