Fun with TypeScript

What TypeScript Is

What TypeScript Is How to Use It

What TypeScript Is
How to Use It
Why It's Awesome

What TypeScript Is
How to Use It
Why It's Awesome
~~Adding to Existing Projects~~

What TypeScript Is

JavaScript

JavaScript
with types

JavaScript

with types

(exactly what it says on the tin)

```
function greeter(person) {
    return "Hello, " + person;
}

let user = "Jane User";

document.body.textContent = greeter(user);
```

TypeScript in 5 Minutes

```
function greeter(person: string) {
    return "Hello, " + person;
}

let user = "Jane User";

document.body.textContent = greeter(user);
```

TypeScript in 5 Minutes

```
interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = { firstName: "Jane", lastName: "User" };

document.body.textContent = greeter(user);
```

TypeScript in 5 Minutes

```
class Student {
    fullName: string;
    constructor(
        public firstName: string,
        public middleInitial: string,
        public lastName: string
    ) {
        this.fullName = firstName + " "
            + middleInitial + " "
            + lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
```

TypeScript in 5 Minutes

```
"use strict";
class Student {
    constructor(firstName, middleInitial, lastName) {
        this.firstName = firstName;
        this.middleInitial = middleInitial;
        this.lastName = lastName;
        this.fullName = firstName + " "
            + middleInitial + " "
            + lastName;
    }
}
function greeter(person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}
let user = new Student("Jane", "M.", "User");
document.body.textContent = greeter(user);
```

TypeScript in 5 Minutes

```
"use strict";
var Student = /** @class */ (function () {
    function Student(firstName, middleInitial, lastName) {
        this.firstName = firstName;
        this.middleInitial = middleInitial;
        this.lastName = lastName;
        this.fullName = firstName + " " + middleInitial + " " + l
    }
    return Student;
}());
function greeter(person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}
var user = new Student("Jane", "M.", "User");
document.body.textContent = greeter(user);
```

TypeScript in 5 Minutes

Important Concepts

Important Concepts

- It goes away at runtime.

Important Concepts

- It goes away at runtime.
- It is not sound.

Important Concepts

- It goes away at runtime.
- It is not sound.
- It's 90% modern JavaScript.

Important Concepts

- It goes away at runtime.
- It is not sound.
- It's 90% modern JavaScript.
- The other 10% is just types.

(I'm making up those percentages.)

How to Use It

Types

These are types:

Types

These are types:

- `number`

Types

These are types:

- `number`
- `string`

Types

These are types:

- `number`
- `string`
- `Array<string>` or `string[]`

Types

These are also types:

Types

These are also types:

- `number?` ("optional - may be undefined" - in objects / interfaces and function parameter lists)

Types

These are also types:

- `number?` ("optional - may be undefined" - in objects / interfaces and function parameter lists)
- `string | null` (union types)

Types

This is also a type (from Node's `http` module):

```
export class Server extends net.Server {
    constructor(requestListener?:
        (req: IncomingMessage, res: ServerResponse) => void);

    setTimeout(msecs?: number, callback?: () => void): this;
    setTimeout(callback: () => void): this;
    maxHeadersCount: number;
    timeout: number;
    keepAliveTimeout: number;
}
```

Enum Types

```
enum Direction {
    Up = 1,
    Down,
    Left,
    Right,
}
console.log(Direction.Up);
```

TypeScript Handbook

Enum Types

That gets turned in to something like this:

```
const Direction = {
  '1': 'Up',
  '2': 'Down',
  '3': 'Left',
  '4': 'Right',
  Up: 1,
  Down: 2,
  Left: 3,
  Right: 4
};
```

Types

This is also a type:

```
type NetServerEvent = 'close'
    | 'connection'
    | 'error'
    | 'listening';
```

Types

You can overload based on types:

```
export class Server extends EventEmitter {
    on(event: string, listener: (...args: any[]) => void): this;
    on(event: "close", listener: () => void): this;
    on(event: "connection", listener: (socket: Socket) => void):
    on(event: "error", listener: (err: Error) => void): this;
    on(event: "listening", listener: () => void): this;
}
```

Intersection Types

```
import { RouteComponentProps } from 'react-router';
interface Props {
    pageTitle: string;
    currentUser?: User;
}
export class MyPage
    extends React.PureComponent<Props & RouteComponentProps> {
}
```

Tuples

```
// Declare a tuple type
let x: [string, number];
// Initialize it
x = ["hello", 10]; // OK
// Initialize it incorrectly
x = [10, "hello"]; // Error
```

TypeScript Handbook

Special Types

Special Types

- `any` - “I'll let you do anything you want with this.”

Special Types

- `any` - “I'll let you do anything you want with this.”
- `unknown` - “You'll have to check and cast before you do anything.”

Special Types

- `any` - “I'll let you do anything you want with this.”
- `unknown` - “You'll have to check and cast before you do anything.”
- `never` - “This can never happen.”

```
function error(message: string): never {
    throw new Error(message);
}
```

Trying It Out

Trying It Out

The TypeScript Playground

Trying It Out

The TypeScript Playground

`ts-node`

Compiling TypeScript

Compiling TypeScript

1. tsc, the TypeScript compiler

Compiling TypeScript

1. tsc, the TypeScript compiler
    - ts-loader (what I use)
    - awesome-typescript-loader

Compiling TypeScript

1. tsc, the TypeScript compiler
   - ts-loader (what I use)
   - awesome-typescript-loader
2. Babel 7

Compiling TypeScript

1. tsc, the TypeScript compiler
   - ts-loader (what I use)
   - awesome-typescript-loader
2. Babel 7
   - tsc --noEmit

Compiling TypeScript

Compiling TypeScript

1. Start with a generator (`create-react-app`), boilerplate, etc.

Compiling TypeScript

1. Start with a generator (`create-react-app`), boilerplate, etc.
2. Modify it as needed for your project

Compiling TypeScript

1. Start with a generator (`create-react-app`), boilerplate, etc.
2. Modify it as needed for your project
3. Grumble and complain when things don't work

Compiling TypeScript

1. Start with a generator (`create-react-app`), boilerplate, etc.
2. Modify it as needed for your project
3. Grumble and complain when things don't work
4. Copy and paste new configurations from Stack Overflow or other projects

Compiling TypeScript

1. Start with a generator (`create-react-app`), boilerplate, etc.
2. Modify it as needed for your project
3. Grumble and complain when things don't work
4. Copy and paste new configurations from Stack Overflow or other projects
5. Check it in and hope you never have to touch it again

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",

    "alwaysStrict": true,
    "strictNullChecks": true,
    "noImplicitAny": true,
    "noImplicitReturns": true,
    "noImplicitThis": true,
    "noUnusedLocals": true
  }
}
```

Getting Type Definitions

https://github.com/DefinitelyTyped/DefinitelyTyped

```
# Install Node package for processing command-line arguments
yarn add yargs
# Install the types for it
yarn add --dev @types/yargs
```

Getting Type Definitions

```
{
    "devDependencies": {
        "@types/yargs": "^13.0"
    },
    "dependencies": {
        "yargs": "^13.2.2"
    }
}
```

Getting Type Definitions

```
# Install Node package for HTTP client
yarn add axios
# Install the types for it
yarn add --dev @types/axios
```

Getting Type Definitions

```
# Install Node package for HTTP client
yarn add axios
# Install the types for it
yarn add --dev @types/axios
```

```
warning @types/axios@0.14.0: This is a stub types definition for
axios (https://github.com/mzabriskie/axios). axios provides its
own type definitions, so you don't need @types/axios installed!
```

Getting Type Definitions

```
# Install Node package for HTTP client
yarn add axios
# Install the types for it
yarn add --dev @types/axios
```

```
warning @types/axios@0.14.0: This is a stub types definition for
axios (https://github.com/mzabriskie/axios). axios provides its
own type definitions, so you don't need @types/axios installed!
```

```
# Groovy. Never mind, then.
yarn remove @types/axios
```

Getting Type Definitions

A snippet of axios's package.json

```
{
    "typings": "./index.d.ts"
}
```

If You Don't Have Types

The simple approach: untyped `require`

```
// This is now of type `any`
const untypedModule = require('untyped-module');
```

If You Don't Have Types

The proper approach: Write your own `.d.ts` file

```
declare module 'trie-search' {
    // Add your interfaces, types, etc. here
    namespace TrieSearch {
        // And more types
    }

    export = TrieSearch;
}
```

If You Don't Have Types

The proper approach: Write your own

```
{
  "include": [
    "app/**/*.ts",
    "node_modules/@types/**/*.d.ts",
    // Reference it within tsconfig.json
    "types/**/*.d.ts"
  ],
}
```

Why It's Awesome

strictNullChecks

```
// tsconfig.json
"strictNullChecks": true,
```

```
function showBanner(user: User | null) {
    // Error! User might undefined.
    alert('Hello, ' + user.name);

    if (user) {
        // Okay - within this block, user cannot be null.
        alert('Hello, ' + user.name);
    }

    // Trust me! I know what I'm doing.
    alert('Hello, ' + user!.name);
}
```

I call it my billion-dollar mistake. It was the invention of the null reference in 1965... My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

— Tony Hoare

One Gotcha

```
interface Dictionary<T> {
    [index: string]: T;
}
```

One Gotcha

```
interface Dictionary<T> {
    [index: string]: T;
}
```

```
const lookup: Dictionary<number> = {
    alpha: 1,
    beta: 2,
};

// This evaluates to `undefined` at runtime.
console.log(lookup['gamma']);
```

Demo: Processing Command-Line Arguments

```
const argv = yargs
  .strict()
  .option('fromDate', {
    alias: 'f',
    coerce: asIsoDate,
    description: `Starting date (${ISO_DATE_FORMAT})`,
  })
  .option('toDate', {
    alias: 't',
    coerce: asIsoDate,
    description: `Ending date (${ISO_DATE_FORMAT})`,
  })
  .option('client', {
    alias: 'c',
    required: true,
    description: 'Client name'
```

```
(property) yargs.Argv<{ fromDate: moment.Moment | undefined; } & { toDate: moment
.Moment | undefined; } & { client: string; } & { wikiPath: string; }>.argv: {
    [x: string]: unknown;
    fromDate: moment.Moment | undefined;
    toDate: moment.Moment | undefined;
    client: string;
    wikiPath: string;
    _: string[];
    $0: string;
}
```

Get the arguments as a plain old object.

Arguments without a corresponding flag show up in the `argv._` array.

```
// Hack: As of May 2019, @types/yargs doesn't understand a
// default function, so we reassign arguments to keep it happy.
const options = {
  ...argv,
  fromDate: (argv.fromDate as unknown) as moment.Moment,
  toDate: (argv.toDate as unknown) as moment.Moment,
};
```

Demo: Adding Async

```
import { IncomingMessage, ServerResponse } from 'http';

function configureProxy(
    app: connect.Server,
    proxy: httpProxy,
    getTarget: (req: IncomingMessage) => string,
) {
    app.use((req: IncomingMessage, res: ServerResponse) => {
        handleRequestInternally(req);
        const target = getTarget(req);
        proxy.web(req, res, { target });
    });
}
```

```
import { IncomingMessage, ServerResponse } from 'http';

function configureProxy(
    app: connect.Server,
    proxy: httpProxy,
    getTarget: (req: IncomingMessage) => string | Promise<string>
) {
    app.use((req: IncomingMessage, res: ServerResponse) => {
        handleRequestInternally(req);
        const target = getTarget(req);
        proxy.web(req, res, { target });
    });
}
```

```
Promise.resolve(valueOrPromiseItDoesntMatter)
    .then(function(value) {})
```

https://stackoverflow.com/a/27760489/25507

```
/**
 * Creates a new resolved promise for the provided value.
 * @param value A promise.
 * @returns A promise whose internal state matches the provided p
 */
resolve<T>(value: T | PromiseLike<T>): Promise<T>;
```

lib.es2015.promise.d.ts

```
import { IncomingMessage, ServerResponse } from 'http';

function configureProxy(
    app: connect.Server,
    proxy: httpProxy,
    getTarget: (req: IncomingMessage) => string | Promise<string>
) {
    app.use((req: IncomingMessage, res: ServerResponse) => {
        handleRequestInternally(req);
        const target = getTarget(req);
        proxy.web(req, res, { target });
    });
}
```

```
import { IncomingMessage, ServerResponse } from 'http';

function configureProxy(
    app: connect.Server,
    proxy: httpProxy,
    getTarget: (req: IncomingMessage) => string | Promise<string>
) {
    app.use((req: IncomingMessage, res: ServerResponse) => {
        handleRequestInternally(req);
        Promise.resolve(getTarget(req))
            .then(target => proxy.web(req, res, { target }));
    });
}
```

Demo: A React Options Form

```
interface Options {
    hideStatus: boolean;
    emailOnNewReplies: boolean;
}
```

```
export const setOption = createAction('SET_OPTION',
  (newOptions: Partial<Options>) => ({
    type: 'SET_OPTION',
    payload: newOptions,
  })
);
```

```tsx
const Checkbox = ({ id, children, options, setOption, }: {
  id: string;
  children: any;
  options: Options;
  setOption: (o: Partial<Options>) => void;
}) => (
  <div>
    <input
      id={id} name={id} type="checkbox"
      checked={(options as any)[id]}
      onChange={() => setOption({ [id]: !(options as any)[id] })}
    />
    <label htmlFor={id}>{children}</label>
  </div>
);
```

```tsx
const Checkbox = ({ id, children, options, setOption, }: {
  id: keyof Options;
  children: any;
  options: Options;
  setOption: (o: Partial<Options>) => void;
}) => (
  <div>
    <input
      id={id} name={id} type="checkbox"
      checked={options[id]}
      onChange={() => setOption({ [id]: !options[id] })}
    />
    <label htmlFor={id}>{children}</label>
  </div>
);
```

It's as if we had this:

```
interface Options {
    hideStatus: boolean;
    emailOnNewReplies: boolean;
}
type OptionsKey = 'hideStatus' | 'emailOnNewReplies';
```

Now let's add a new option:

```
interface Options {
    hideStatus: boolean;
    emailOnNewReplies: boolean;
    timeZone: number;
}
```

```tsx
<input
    id={id} name={id} type="checkbox"
    checked={options[id]}
    onChange={() => setOption({ [id]: !options[id] })}
/>
```

```
checked?: boolean;
crossOrigin?: string;
```

```
(JSX attribute) React.InputHTMLAttributes<HTMLInputElement>.checked?: boolean
 | undefined
```

```
Type 'number | boolean | undefined' is not assignable to type 'boolean |
undefined'.
  Type 'number' is not assignable to type 'boolean | undefined'. ts(2322)
```

```
index.d.ts(1886, 9): The expected type comes from property 'checked' which is
declared here on type
'DetailedHTMLProps<InputHTMLAttributes<HTMLInputElement>, HTMLInputElement>'
```

Peek Problem No quick fixes available

```
// https://stackoverflow.com/q/49752151/25507
export type KeysOfType<T, TProp> =
  { [P in keyof T]: T[P] extends TProp ? P : never }[keyof T];

const Checkbox = ({ id, children, options, setOption, }: {
  id: KeyOfType<Options, bool>;
  children: any;
  options: Options;
  setOption: (o: Partial<Options>) => void;
}) => (
  <div>
    <input
      id={id} name={id} type="checkbox"
      checked={options[id]}
      onChange={() => setOption({ [id]: !options[id] })}
    />
```

References

- TypeScript Handbook
- TypeScript With Babel: A Beautiful Marriage
- Choosing between Babel and TypeScript
- “What is soundness, exactly, and how is Typescript unsound?”

Credits

- Windows XP "Bliss"