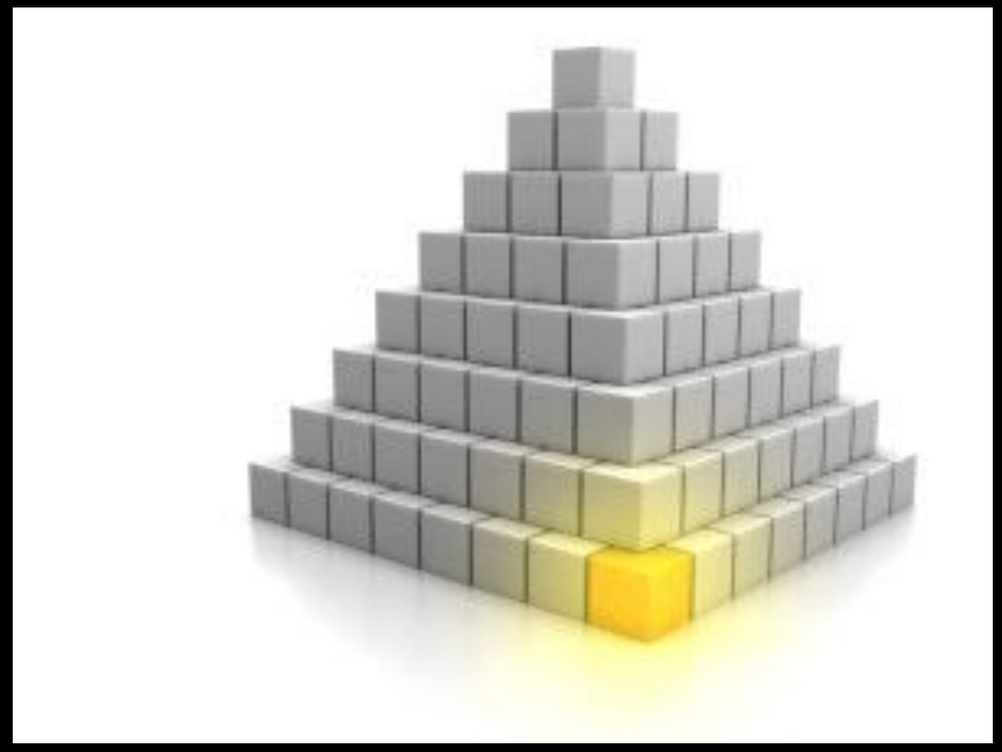
Introducing FlowType compared to TypeScript

Tor Arne Kvaløy (FINN.no)

JavaScript

Java



Types

- Improve productivity
  - `function save(person){…}`
  - Self documented code
- Find bugs at ‘compile time’
  - Missing/wrong object properties
  - Missing/wrong function parameters
  - Missing/wrong function return value
  - Variable nullable/undefined or not
- Simplify refactoring
  - `const person = {eMail: ’John@gmail.com’};`

FlowType vs TypeScript

- FlowType:
  - Annotation of standard javascript
  - Babel plugin: transform-flow-strip-types
    - `const name:string = ‘John’;`
    - `const name = ’John’;`
  - Weak mode
  - Good native support for React

- TypeScript:
  - Better documentation
  - Better editor integration
  - Microsoft compiler…
  - Annoying .ts-extension

Annotating functions

```
// @flow

function save(name: string, age: number): void {
}
```

TS: .ts file extension

Annotating objects

```
type Person = {
    name: string,
    age: number,
}

function save(person: Person): void {
    person.name = ‘John’;
    person.age = 25;
}
```

TS: **interface**

Type checking

```
type Person = {
    name: string,
    age: number,
}

function save(person: Person): void {
    person.name = ‘John’;
    person.age = ’25’;
}
```

Type checking

```
type Person = {
    name: string,
    age: number,
}

function save(person: Person): void {}

save(25);
save({age: 25});
save({name: ‘John’, age: 25});
```

Function params

```
type Person = {
    name: string,
    age: number,
}

function save(person: Person, tx: boolean): void {}

save({name: ‘John’, age: 25});
save({name: ‘John’, age: 25}, true);
```

Annotated objects are sealed

```ts
type Person = {
    name: string,
    age: number,
}

function save(person: Person): void {
    const name = person.name;
    const phone = person.phone;
}
```

Missing property

```ts
type Person = {
    name: string,
    age: number,
}

function save(person: Person): void {}

save({name: ‘John’, age: 25});
save({name: ‘John’});
```

Extra property is ok

```
type Person = {
    name: string,
    age: number,
}
function save(person: Person): void {}
save({name: ‘John’, age: 25, phone: 123});
```

PS: Interfaces are exact

Exact Object Type

```
type Person = {|
    name: string,
    age: number,
|}

function save(person: Person): void {}

save({name: ‘John’, age: 25, phone: 123});
```

PS:TypeScript: Object are exact

Optional properties

```ts
type Person = {
    name: string,
    age?: number,
}

function save(person: Person): void {
    const age = person.age;

    if (person.age) {
      const age = person.age;
    }
}
```

null/undefined value

```
type Person = {
    name: string,
    age: ?number,
}
```

```
function save(person: Person): void {
    person.name = null;
    person.age = null;
}
```

TS: age: number | null | undefined

Arrays

```
type Person = {
    name: string,
    age: number,
}

const persons: Person[] = [];

persons.push({name: ‘John’});

persons.push({name: ‘John’, age: 25});

const name = persons[0].name
```

Callbacks

```
type Person = {
    name: string,
    age: number,
}

function save(getPerson: () => Person): void {
    const person = getPerson();
    const name = person.name;
}
```

Derived types

```
const person = {
    name: ‘John’,
    age: 25,
}

person.name = ‘Phillip’;

person.name = 30;

const phone = person.phone;
```

Derived parameter type

```
const person = {
    name: ‘John’,
    age: 25,
}
```

```
function save(person: Person): void {
   foo(person);
}

function foo(person) {
  const name = person.name;
  const phone = person.phone;
}
```

Derived return type

```
const person = {
    name: ‘John’,
    age: 25,
}


function getName(person: Person) {
   return person.name;
}

cont name: number = getName(person);
```

React class comp

```
type Props = {
    name: string,
    age: number,
}

class PersonComp extends Component {
    props: Props;
    render () {
        return <div>{props.name}</div>
    }
}

<PersonComp name=‘John’ />
<PersonComp name=‘John’ age={25}/>
```

React functional comp

```tsx
type Props = {
    name: string,
    age: number,
}

function PersonComp({ name, age }: Props) {
    return <div>{name}</div>
}


<PersonComp name='John' />
<PersonComp name='John' age={25}/>
```

React Redux comp

```
type Props = {
    name: string,
    age: number,
}

class MyComp extends Component {
    props: Props & { dispatch: Dispatch };

    onClick = (event: SyntheticEvent) => {
       this.props.dispatch(myAction(event.foo.bar));
    };

    render () {
         return <div onClick={this.onClick}>{props.name}</div>
     }
}

function mapStateToProps(rootState: RootState) {
   return { …rootState.foo };
}

return connect(mapStateToProps)(MyComp);
```

TypeScript class

```
class Person {
    private name: string;
    public age: number;
    readonly email: string;
    constructor(public foo:string){
        // transpiles to: this.foo = foo;
    }
}
```

Model boundaries

```
export function save (person) {}

export function save (person: Person): void {}
```

import / export

foo.js:

```
export type Person = {
    name: string,
    age: number,
}
```

bar.js.:

```
import { type Person } from ‘./foo’;
```

```
TS: import { Person } from ‘./foo’
```

Typify libs

foo.js:

```
export function save (person) {
    return true;
}
```

foo.js.flow:

```
//@flow
declare export function save (person: Person): boolean;
```

How FlowType has improved my productivity

- Clear definition of objects and functions(less guessing)
- Safer to send objects around
- More conscious of how objects are used over time
- More conscious of initial state of objects
- More auto-completion
- Much faster refactoring

Demo

- Type chcking
  - dateTimePickupReducer.js
- Refactoring
  - submitActions.js

To conclude

After 3 weeks of using FlowType…

…there is no way back.