FlowType

Tor Arne Kvaløy (FINN.no)

So much has changed!

a few years ago	Now
No external dependencies	package.json, yarn, NPM
No dependencies between files	Browserify, Webpack
No, or crappy listing	ESLint
Manual DOM manipulation	Declerative DOM (React, Vue)
No type checking	FlowType, TypeScript

Why we need 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’};`

Why FlowType

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

Annotating functions

```
// @flow

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

}
```

Annotating objects

```
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 {
     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

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

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

Missing property

```
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});
```

Exact Object Type

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

function save(person: Person): void {}

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

Optional properties

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

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

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

null value

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

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

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);
```

Type casting

```
type Person = {
    name: ‘John’,
    age: 25,
}


const age = (window.person: Person).age;
```

React PropTypes

```
class PersonComp extends Component {
    propTypes = {
        name: PropTypes.string.isRequired,
        age: PropTypes.number.isRequired,
    }

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

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

React class comp

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

class PersonComp extends Component<void, Props, State> {
    render () {
        return <div>{props.name}</div>
    }
}

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

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

```
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);
```

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’;
```

Writing libs

foo.js:

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

foo.js.flow:

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

Libs without types

- flow-typed
  - Moment
  - chai
  - redux-react (…)
- Outdated
  - Jasmine (has defs for 2.4, the newest version is 2.5…)

Demo

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

To conclude

After 3 weeks of using FlowType…

…there is no way back.

We are hiring!

HELP WANTED