

# Как помогает TypeScript

# Общее

2

- TS - ~~язык~~ надмножество JS, система типов и компилятор
- Open-source, разрабатывается Microsoft
- TypeScript, VS Code, Slack, Angular2+, Vue3, Ant.design ...
- Отдельная типизация - @types и .d.ts
- Некоторые редакторы используют TS тайпинги для IntelliSense

# Интерфейс (тип) и простой рефакторинг

3

```
function greet(user) {  
  console.log(user.name)  
}
```

*// Структура user изменилась (name -> firstName)*

```
function greet(user) {  
  console.log(user.name) // упадет в рантайме  
}
```

# Интерфейс (тип) и простой рефакторинг

4

*// Добавим TS*

```
interface User {  
  id: string  
  firstName: string  
  lastName: string  
}
```

```
function greet(user: User) {  
  console.log(user.name) // TS предупреждает
```

```
user.| // автокомплит!  
}  
  
function greet('wrong type param') // ошибка
```

## То же самое в реакт компонентах

```
type Props = {  
  onChange: (value: string) => void  
  value?: string  
}  
  
function Input(props: Props) {  
  return <StyledInput {...props} />  
}
```

```
<Input value="" onChange={val => { console.log(val) }} />  
<Input value={1} /> // ошибки
```

# Class-based компоненты

```
type Props = {  
  name: string  
}  
type State = {  
  value: string
```

```
}  
  
class FormInput extends Component<Props, State> { // Дженирики!  
  onChange = value => { this.setState({ value })}  
  render() {  
    const {name} = this.props // типизирован  
    const {value} = this.state // типизирован  
    return <>  
      <Input value={value} onChange={onChange} />  
      {value && value.split('').reverse().join('')}  
      {value.reverse()} // TS2339: Property 'reverse' does not exist on type 'string'  
    </>  
  }  
}
```

## Union of strings

```
type Props = {  
  onChange: (value) => void  
  value: string  
}
```

```
    size?: 'small' | 'large'  
  }  
  
  function Input(props: Props) {  
    return <StyledInput {...props} />  
  }  
  
<Input size="smoll" /> // Ошибка
```

## Переиспользование типов



```
type Props = {  
    size?: 'small' | 'large' // нарушаем принцип DRY  
}
```

```
class FormInput extends Component<Props, State> {}
```

# Переиспользование типов

```
// Input.tsx
```

```
export type InputSize = 'small' | 'large'
```

```
type Props = {  
  size?: InputSize  
  ...  
}
```

```
// FormInput.tsx
```

```
import { Input, InputSize } from './Input'
```

```
type Props = {  
  size?: InputSize // уже лучше  
}
```

# Type intersection

*// Input.tsx*

```
export type InputProps = {  
  size?: InputSize  
}  
  
type Props = InputProps & {  
  value: string  
  onChange: (value: string) => void  
}
```

*// FormInput.tsx*

```
import { Input, InputProps } from './Input'  
  
class FormInput extends Component<InputProps, State> {}
```

# Наследование интерфейсов

11

*// Input.tsx*

```
export interface InputProps {  
  size?: InputSize  
}  
  
interface Props extends InputProps {  
  value: string  
  onChange: (value: string) => void  
}
```

*// FormInput.tsx*

```
import { Input, InputProps } from './Input'
```

```
class FormInput extends Component<InputProps, State> {}
```

# Условные типы

12

```
// Input.tsx
```

```
export type InputProps = {  
  value: string  
  onChange: (value: string) => void  
  size?: InputSize  
}
```

```
// FormInput.tsx
```

```
import { Input, InputProps } from './Input'
```

```
type Props = Omit<InputProps, 'value' | 'onChange'> // мой выбор
```

# Массивы

```
type User = {  
  id: string  
  name: string  
}
```

```
function getUsers(): Users[] {...}
```

```
const users = getUsers()
```

```
users.forEach(el => el.|) // TS знает, что el имеет тип User
```

# Index signature

```
{  
  '1': {  
    id: '1',  
    name: 'Foo'  
  },  
  type User = {  
    id: string  
    name: string  
  }  
}
```

```

    '2': {
      id: '2',
      name: 'Bar'
    }
  }

  const users: { [key: string]: User } = {
    foo: { // не хватает поля name
      id: 'foo'
    },
    bar: {
      id: 'bar',
      unknown: '' // такого поля нет в User
    },
  }
}

```

## Mapped types и keyof typeof

```

export type Status = 'guest' | 'admin' | 'editor'
export const statuses: { [K in Status]: string } = {
  guest: 'Гость',

```



```
    admin: 'Админ',  
    editor: 'Редактор'  
}  
  
// или так  
export const statuses = {  
  guest: 'Гость',  
  admin: 'Админ',  
  editor: 'Редактор'  
}  
export type Status = keyof typeof statuses
```

## Значения полей объекта в union

```
const roles = {  
  User: 'ROLE_USER',  
  Admin: 'ROLE_ADMIN',  
} as const
```

```
type Roles = typeof roles  
  
// {  
//   'User': 'ROLE_USER',  
//   'Admin': 'ROLE_ADMIN'  
// }
```

```
type RoleValues = Roles[keyof Roles]  
// 'ROLE_USER' | 'ROLE_ADMIN'
```

# Иконки

17

Icon

| --icons

| | --check.svg

| | --close.svg

| | --index.ts

| --Icon.tsx

*// icons/index.ts*

**export** { **default as** check } **from** './check.svg';

**export** { **default as** close } **from** './close.svg';

*// Icon.tsx*

**import** \* **as** icons **from** './icons';

**type** Props = {

    icon: **keyof** **typeof** icons

}

**export function** Icon(props: Props) {

**const** CurrentIcon = icons[props.icon];

**return** <CurrentIcon />

}

*// somewhere*

<Icon icon=**"chek"**/> *// автокомплит и тайпчек по названию иконок*

# Рекомендации

- Редактор с хорошей поддержкой TS (VS Code, WebStorm)
- Strict Mode
- Избегайте `any` и `// @ts-ignore`, стремитесь к 100% покрытию типами
- Typescheck в гит хуках
- Попробуйте, если не пробовали =)

# Ссылки

19

[TypeScript Deep Dive](#)

[Typescript Evolution](#)

[Презентация](#)

[Github](#)

