
FEDAY 北京 2017
第三届
FEDAY(前端开发日)
UCLOUD
100offer
upyun
Broadview
图灵教育
GenePlan
StuQ
CSDN
W3C
掘金
前端早读课
中生代技术
segmentfault
码云
Gitee.com

如何构建大型 Web 应用

孟红伦（云际）
2017.08

About Me

负责钉钉桌面端

分享大纲

- 使用 TypeScript 解决代码复杂，数据模型复杂的问题
- TypeScript 社区概况
- 使用 RxJS 更好的解决异步带来的问题

超过300万家企业组织正在使用钉钉

钉钉正在共享各行各业优秀的工作方式，帮助中国企业进入智能移动办公时代。
让工作更简单、高效、安全

50万+
IT互联网企业

查看优秀案例 >

35万+
制造型企业

查看优秀案例 >

20万+
贸易/物流企业

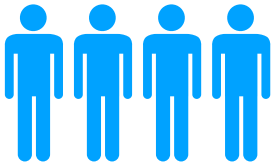
查看优秀案例 >

20万+
建筑/房地产企业

查看优秀案例 >

2014 年

湖畔花园



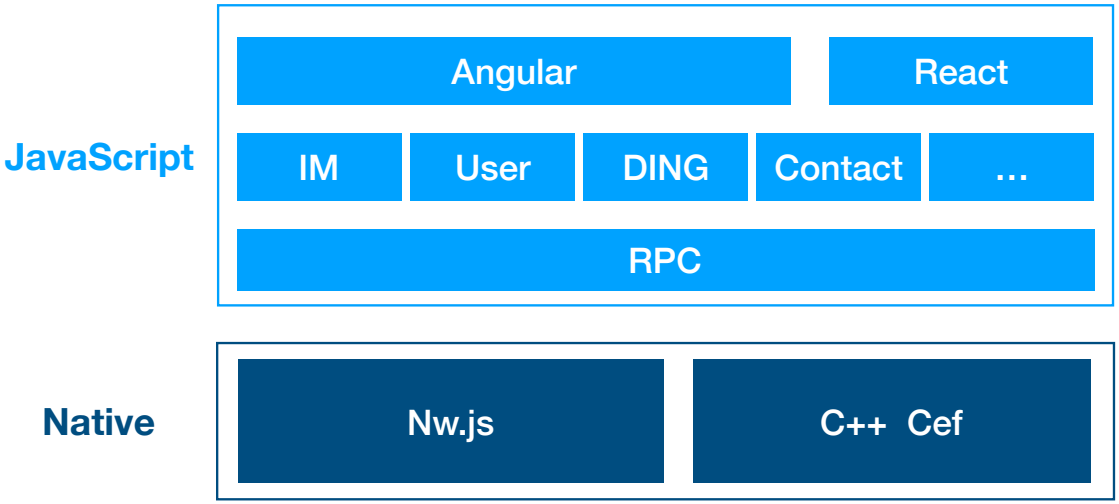
前端 X 4

0 个 Mac/Windows Native 工程师

工作场景离不开桌面端

钉钉桌面版

主要界面的 UI 使用 Web 技术



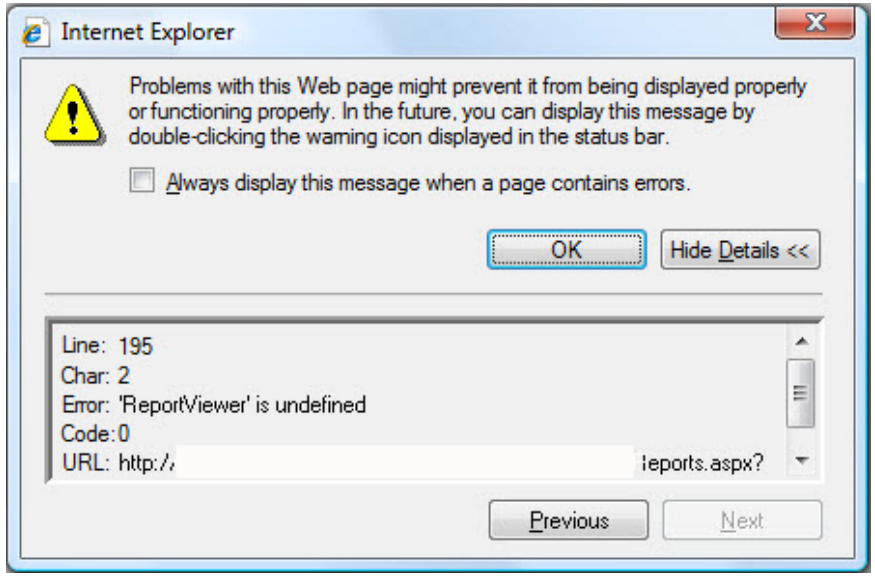
2014 年

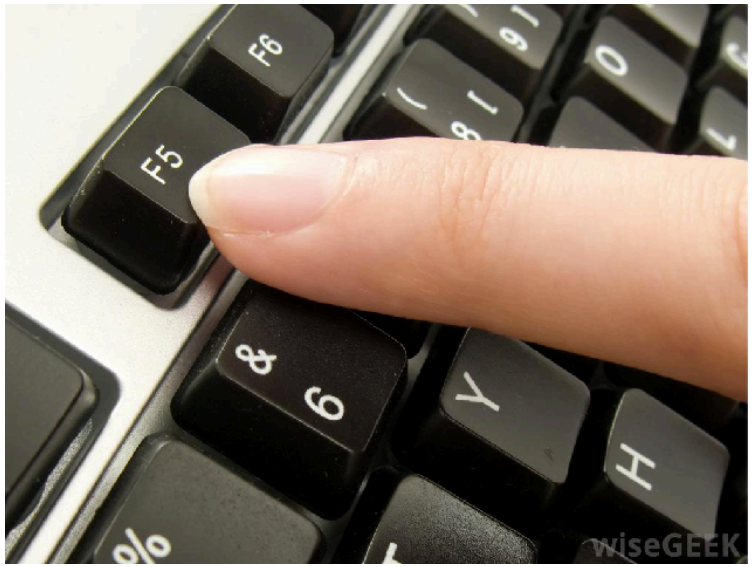
10 月份上线内测 Web 版本，解决了钉钉团队内部沟通的效率

11 月份使用 nw.js 包装了第一个桌面端拿出去找用户共创

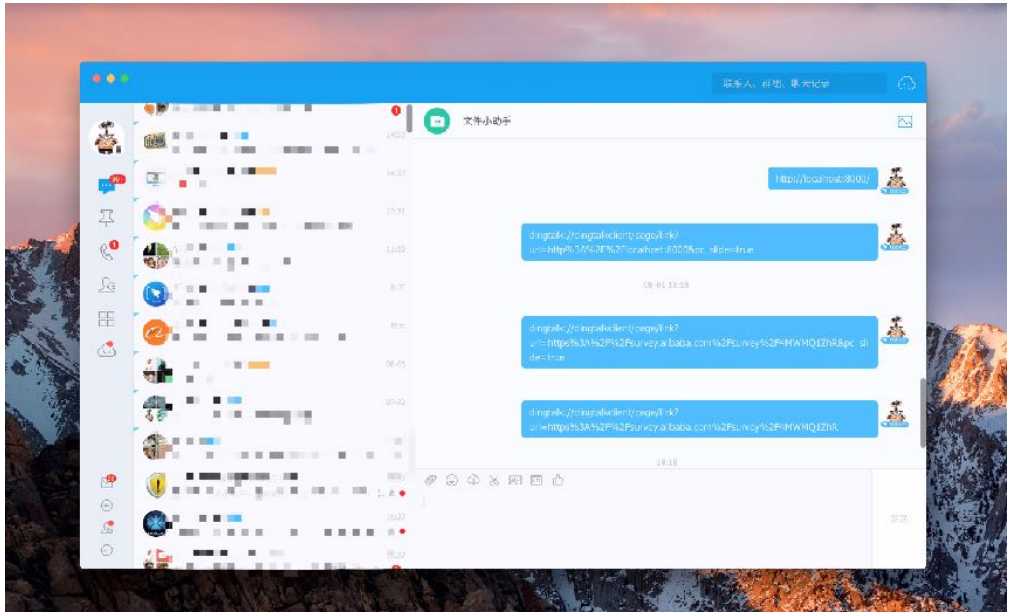
赢得了共创用户的口碑，给共创用户了信心

Web


Internet Explorer
Problems with this Web page might prevent it from being displayed properly or functioning properly. In the future, you can display this message by double-clicking the warning icon displayed in the status bar.
Always display this message when a page contains errors.
OK
Hide Details <<
Line: 195
Char: 2
Error: 'ReportViewer' is undefined
Code: 0
URL: http:/
leports.aspx?
Previous
Next


F6
F5
wiseGEEK

桌面 App


文件小助手


MON TUE WED THU FRI SAT SUN
09:00:00
AM
MON TUE WED THU FRI SAT SUN
18:00:00
PM

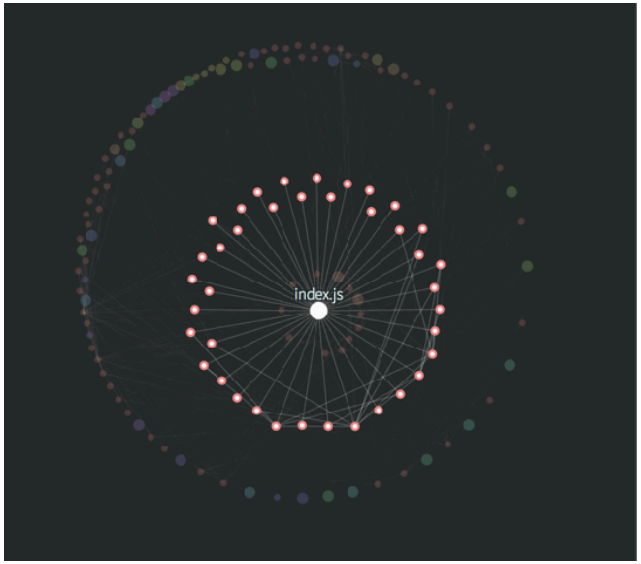
钉钉的特点

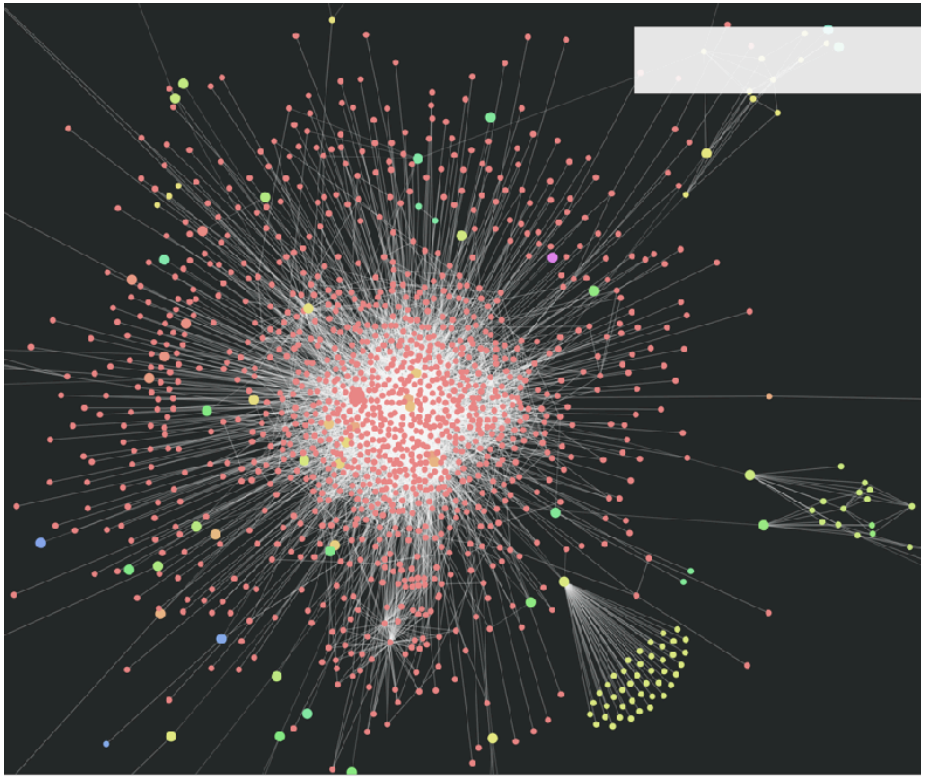
业务复杂

实时性要求高

IM 消息
消息已读未读状态
群禁言
群加人删除人
DING 消息
DING 回复
用户更改头像
用户更改用户名
用户更改状态
...

业务复杂 -> 代码复杂

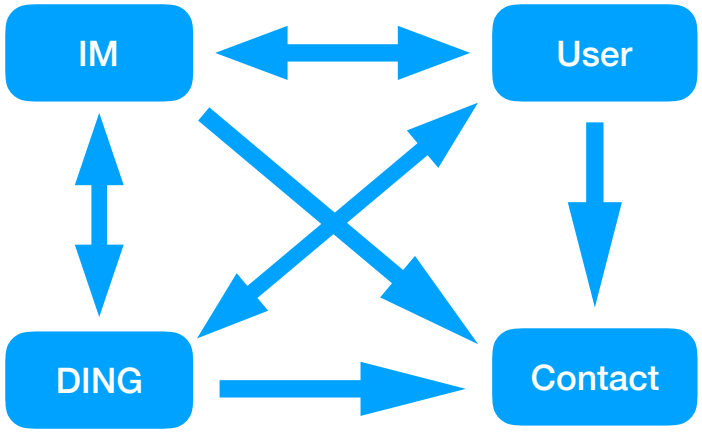




钉钉这几年成长为一个大应用

代码复杂

- 上百个 RPC 接口
- 几十种消息推送
- 10W + 行代码
- 模块依赖关系复杂
- 数据模型复杂



上百个 RPC 接口

- 接口地址是什么?
- 接口的参数是什么?
- 接口的返回值是什么?

接口的返回值

```
import axios from 'axios';
export const getUserByUid = (uid) => {
    return axios.get('/users?id=' + uid).then((response) => {
        return response.data;
    });
};
getUserByUid(123).then((user) => {
    console.log(user.avatar);
    console.log(user.bio);
});
```

User 有哪些属性?

- 注释? 文档?
- 看代码?
- 跑一次，console 出来?

User 数据模型

```
{
  "uid": 12345,
  "nickName": "云际",
  "sex": "M",
  "avatar": "@xfdjj",
  "isActive": true,
  "org": {
    “orgName”: “钉钉”,
    “orgId”: “0001”
 }
}
```

早期结构简单，不容易出错

User 数据模型

```
{
  "uid": 12345,
  "nickName": "云际",
  "sex": "M",
  "avatar": "@xfdjj",
  "bio": "Web Developer",
  "isActive": true,
  "org": {
    "mainOrgId": "00001",
    "orgList": [
      {
        "orgId": "00001",
        "orgName": "钉钉",
        "departmentList": [
          {
            "departId": "0001",
            "departName": "前端"
          }
        ]
      }
    ]
  }
}
```

业务越来越复杂，数据模型也随之复杂

实际的数据模型，比这个还要复杂的多

User 数据模型

```
{
  "uid": 12345,
  "nickName": "云际",
  "sex": "M",
  "avatar": "@xfdjj",
  "bio": "Web Developer",
  "isActive": true,
  "org": {
    "mainOrgId": "00001",
    "orgList": [
      {
        "orgId": "00001",
        "orgName": "钉钉",
        "departmentList": [
          {
            "departId": "0001",
            "departName": "前端"
          }
        ]
      }
    ]
  }
}
```

个人用户没有 “org” 这个字段

```
let mainOrgId = user.org.mainOrgId;
```

Uncaught TypeError: Cannot read property 'mainOrgId' of undefined

钉钉曾经犯过的错

如果使用 TypeScript

User 数据模型

```typescript
// user.ts
import axios from 'axios';

export enum SEX {
    Male = 'M',
    Female = 'F'
}
export interface IDepartment {
    departId: string;
    departName: string
}
export interface IOrgItem {
    orgId: string;
    orgName: string;
    departmentList: IDepartment[];
}
export interface IOrg {
    mainOrgId: string;
    orgList: IOrgItem[];
}
```

```typescript
export interface IUser {
    uid: number;
    nickName: string;
    sex: SEX,
    avatar: string,
    bio: string,
    isActive: boolean,
    org?: IOrg
}
export const getUserByUid = (uid: number): Promise<IUser> => {
    return axios.get('/users?id=' + uid).then((response) => {
        return response.data as IUser;
    })
};

getUserByUid(123).then((user) => {
    console.log(user.avatar);
    console.log(user.bio);
})
```

User 数据模型

有效避免可选属性误用

```
export interface IUser {
    uid: number;
    nickName: string;
    sex: SEX,
    avatar: string,
    bio: string,
    isActive: boolean,
    org?: IOrg
}
export const getUserByUid = (uid: number): Promise<IUser> => {
    return axios.get( url: '/users?id=' + uid).then( onfulfilled: (response) => {
        return response.data as IUser;
    })
};

getUserByUid(123).then( onfulfilled: (user: IUser) => {
    console.log(user.avatar);
    console.log(user.bio);
    console.log(user.org.mainOrgId);
})
```

TS2532:Object is possibly 'undefined'.

自动生成带有 types 的 RPC 代码

- 手写复杂的 types 比较麻烦
- 手写的 types 怎么做到和服务端返回值一致?

客户端和服务端依赖同一套描述文件生成代码

自动生成带有 types 的 RPC 代码

描述接口的 IDL 文件：

类似 Java Interface

user.idl

```
module userProfile;

struct IDepartment {
    1: string departId;
    2: string departName;
}
struct IOrgItem {
    1: string orgId;
    2: string orgName;
    3: list<IDepartment> departmentList;
}
struct IOrg {
    1: string mainOrgId;
    2: list<IOrgItem> orgList;
}
struct IUser {
    1: long uid;
    2: string nickName;
    3: string sex;
    4: string avatar;
    5: string bio;
    6: boolean isActive;
    7: IOrg org;
}

interface UserProfileIService {
    IUser getUserByUid(long uid);
}
```

自动生成带有 types 的 RPC 代码

IDL Parser 本身也要处理复杂的数据模型，也使用 TS 编写

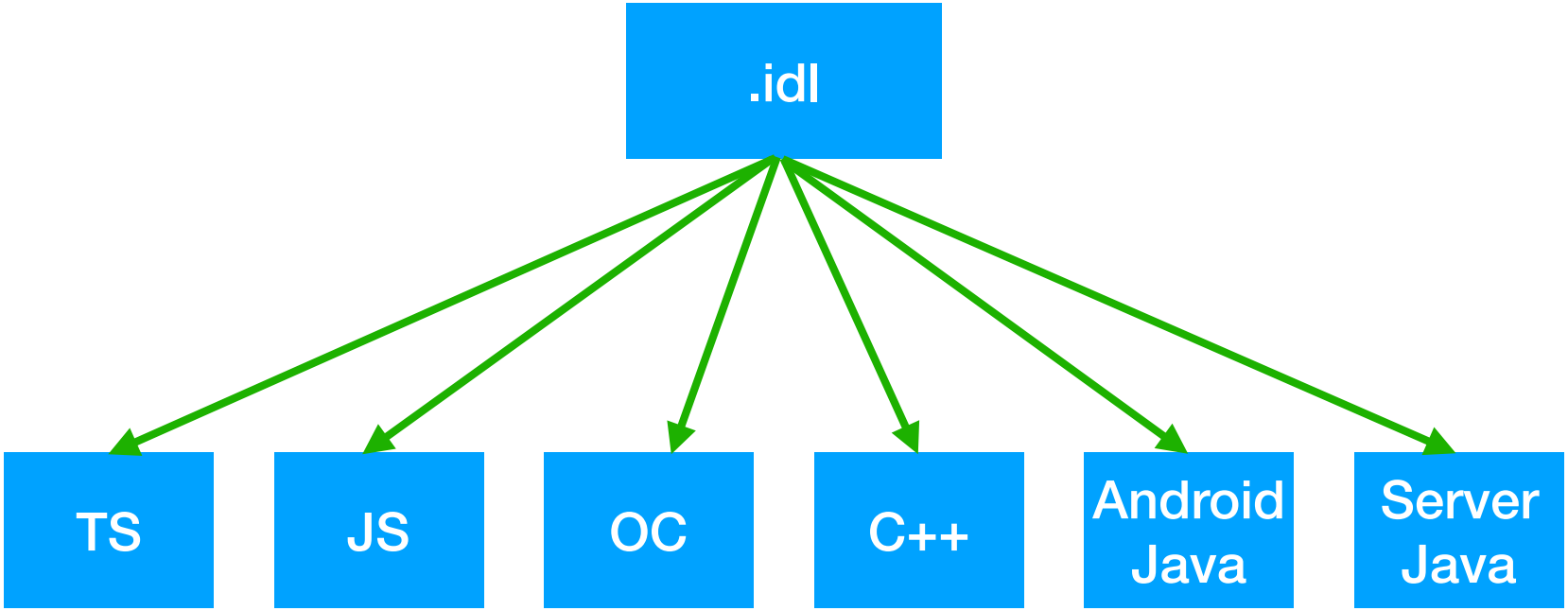
自动生成带有 types 的 RPC 代码

自动生成的代码：

userProfile.ts

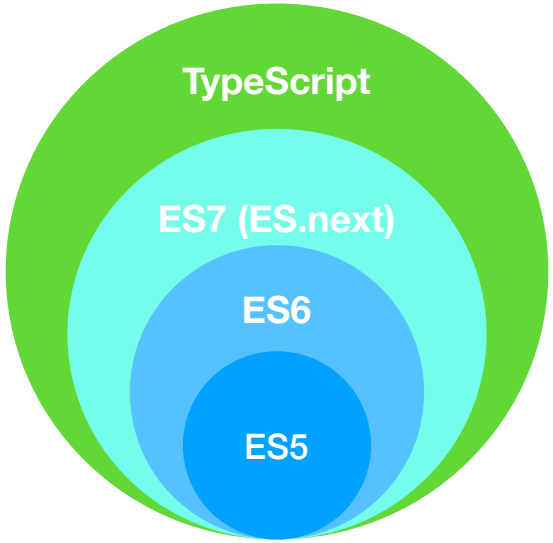
```ts
import {BasicMap} from '../lib/basic-types';
import {sendMsg, IRPCResult} from '../lib/rpc-request';
import {IUser, IOrg, IOrgItem, IDepartment} from './userProfile'
export interface IUser {
    uid?: string;
    nickName?: string;
    sex?: string;
    avatar?: string;
    bio?: string;
    isActive?: boolean;
    org?: IOrg;
}
export interface IOrg {
    mainOrgId?: string;
    orgList?: IOrgItem[];
}
export interface IOrgItem {
    orgId?: string;
    orgName?: string;
    departmentList?: IDepartment[];
}
export interface IDepartment {
    departId?: string;
    departName?: string;
}
export const UserProfileI_getUserByUid = (uid: string): Promise<IRPCResult<IUser>> => {
    return sendMsg<IUser>('/UserProfileI/getUserByUid', {}, [uid]);
};
```

自动生成带有 types 的 RPC 代码



自动生成各个端的代码

什么是 TypeScript



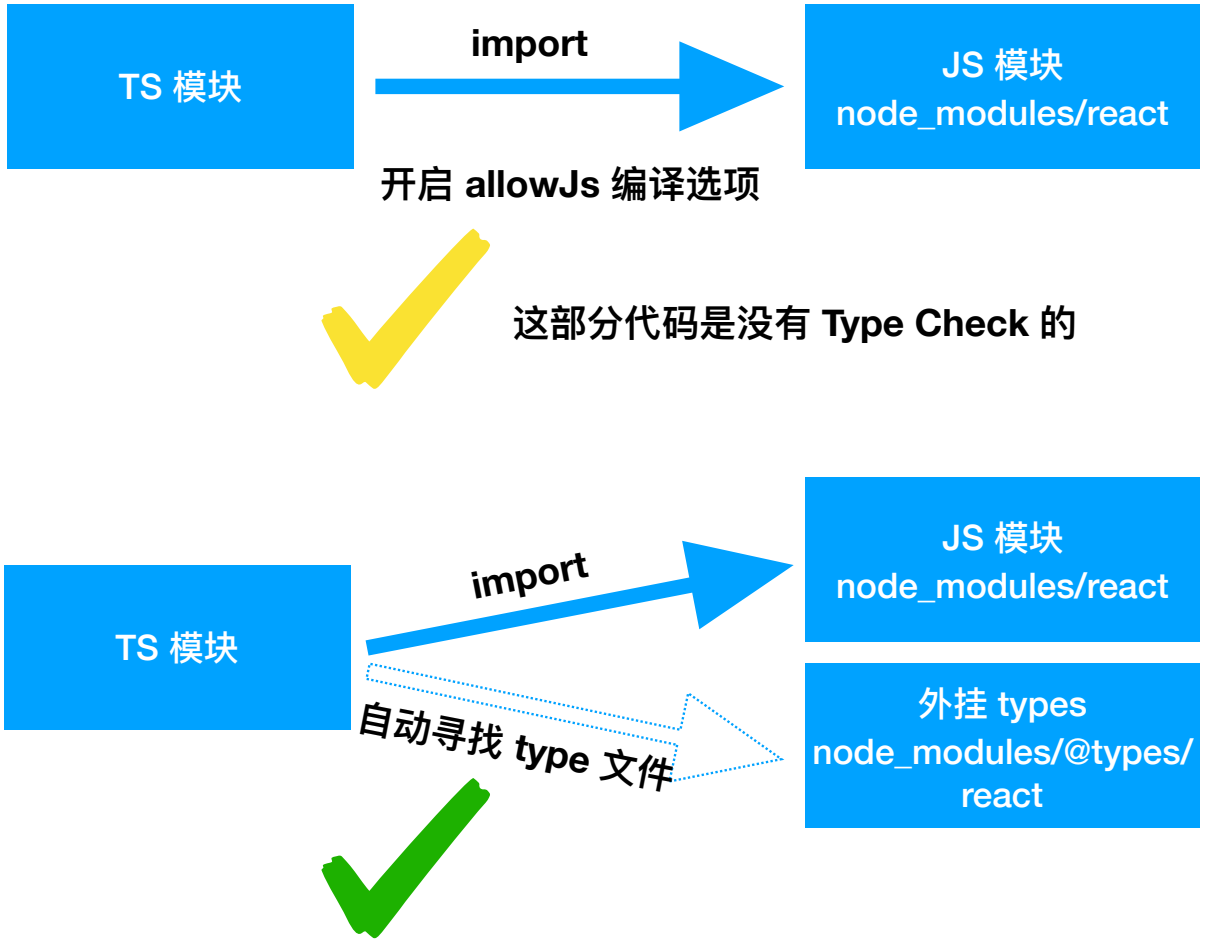
什么是 TypeScript

- 是微软开发的免费，开源的编程语言。
- Anders Hejlsberg 是核心开发者，架构师。他同时也是 C# 的首席架构师。
- 它是 JavaScript 的超集，兼容 ES5, ES2015, ES2016,ES2017 以及未来的一些标准。
- 它的编译结果是 JavaScript ，可以运行在浏览器，Node.js 及其他 JavaScript 环境下。
- 它支持 TS 和 JS 混用，支持逐步迁移你的代码。

TypeScript 社区发展情况

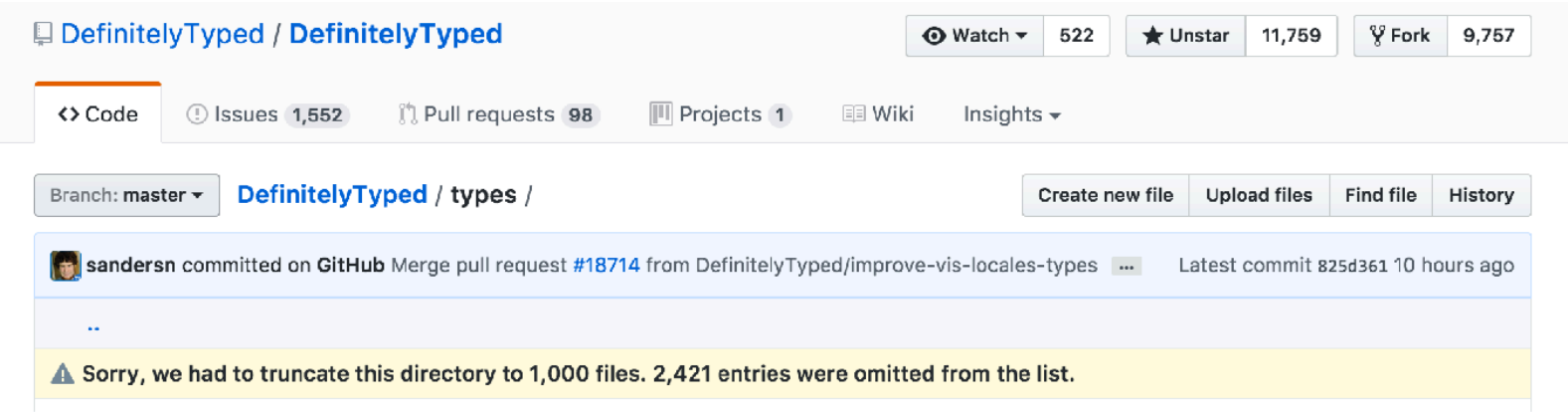
- 大量的使用 JS 写的库，支持 TypeScript。例如 React, Vue, Angular1.x, Redux, React-redux, lodash, moment 等等。
- 很多大型开源项目使用 ts 作为开发语言，例如 Angular 2+, RxJS , cycle.js, vscode, ant-design 等等。
- 主流的 IDE / 编辑器都支持 TypeScript, 例如 WebStorm, VSCode, VSStudio, Sublime, Atom, Eclipse, Emacs, Vim 等等。

TS 工程引用 JS 模块


TS 模块
import
JS 模块
node_modules/react
开启 allowJs 编译选项
这部分代码是没有 Type Check 的
TS 模块
import
JS 模块
node_modules/react
自动寻找 type 文件
外挂 types
node_modules/@types/
react

TS 工程引用 JS 模块

@types 下有 3K+ 的模块



这些模块托管在 DefinitelyTyped/DefinitelyTyped Github 仓库，涵盖了主流的 JS 模块

TS 工程引用 JS 模块

如果你有内部库使用 JS 写的，你可以很方便的为它增加 .d.ts type 文件

entry.d.ts

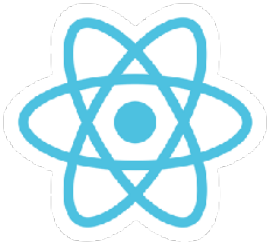
```
interface ICountryDesc {
    countryKey: string;
    countryCode: string;
    countryName_zh: string;
    countryName: string;
    flag: string
}

interface ICNCountryDesc {
    countryKey: string;
    countryCode: string;
    countryName_zh: string;
    countryName: string;
    countryNamePinYin: string;
    flag: string
}

export const en_US_country: ICountryDesc[];
export const zh_CN_country: ICNCountryDesc[];
export const zh_TW_country: ICountryDesc[];
```

package.json 里指定 types 文件

```
{
  “main”: “./entry.js”,
  “types”: “./entry.d.ts”
}
```

主流框架支持情况






```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!
  </h1>
  <ul>
    <li *ngFor=" let country of countryList; index as i;">{{country.code}}</li>
  </ul>
</div>
```

```
import { Component } from '@ar

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppCompo
  title = 'app';
  countryList: string[];
  constructor() {
    this.countryList = ['china', 'USA'];
  }
}
```

```
/Users/allenm/dev/try/ng-test/src/app/app.component.html
Error:(7, 61) Angular: Identifier 'code' is not defined. '<anonymous>' does not contain such a member
```

Angular 扩展了 TS，使得 template 也具备 Type Check 的能力

使用 Angular AOT 编译方式，模板会变成 TS 代码，直接具备了 Type Check

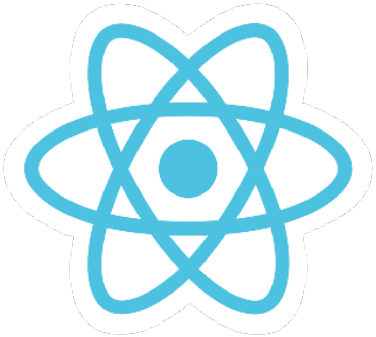
https://vuejs.org/v2/guide/typescript.html

```
import Vue from 'vue';
import Component from 'vue-class-component';

@Component({
    template: re
})
export class HomeComponent extends Vue {
    package: string = 'vue-webpack-typescript';
}
```

```
<div class="container">
  <h1><strong>Thank you for using <span class="text-primary package">{{package.code}}</span></strong></h1>
</div>
```

无报错，dev 模式，build ，Chrome 控制台均无报错

模板：https://github.com/ducksoupdev/vue-webpack-typescript



```
import React from 'react';

interface IHelloProp {
    country: string
}

export class Hello extends React.PureComponent<IHelloProp, {}> {
    public render() {
        let { country } = this.props;
        return (
            <div>{country.code}</div>
        )
    }
}
```

/Users/allenm/dev/pc/pc-quicksilver/src/components/chat/hello.tsx

Error:(11, 27) TS2339:Property 'code' does not exist on type 'string'.

TypeScript 支持 jsx 语法，jsx 不是“模板”，而是 js，所以 Type Check 支持的很好

```typescript
interface ICreateNewTodoItemAction {
    type: 'createNewTodo';
    title: string;
}

interface IDeleteTodoItemAction {
    type: 'deleteTodoItem';
    id: number
}

interface ITodoItem {
    id: number;
    title: string
}
interface IStore {
    maxId: number;
    todoList: ITodoItem[];
}

type Actions = ICreateNewTodoItemAction | IDeleteTodoItemAction;

const reducer = (state: IStore , action: Actions) => {
    switch (action.type) {
        case 'deleteTodoItem':
            console.log(action.title);
            break;
        case 'createNewTodo':
            console.log(action.id);
            break;
    }
};
```

```
/Users/allenm/dev/slide/js-scale-application/switchCase.ts
Error:(26, 32)  TS2339:Property 'title' does not exist on type 'IDeleteTodoItemAction'.
Error:(29, 32)  TS2339:Property 'id' does not exist on type 'ICreateNewTodoItemAction'.
```

可以根据 switch case 的条件，自动收敛到对应的 type

现在就开始使用 TS

- 它支持 TS 和 JS 混用，支持逐步迁移你的代码
- 它是 ES 的超级，兼容 ES 代码
- 方便和各种构建工具集成

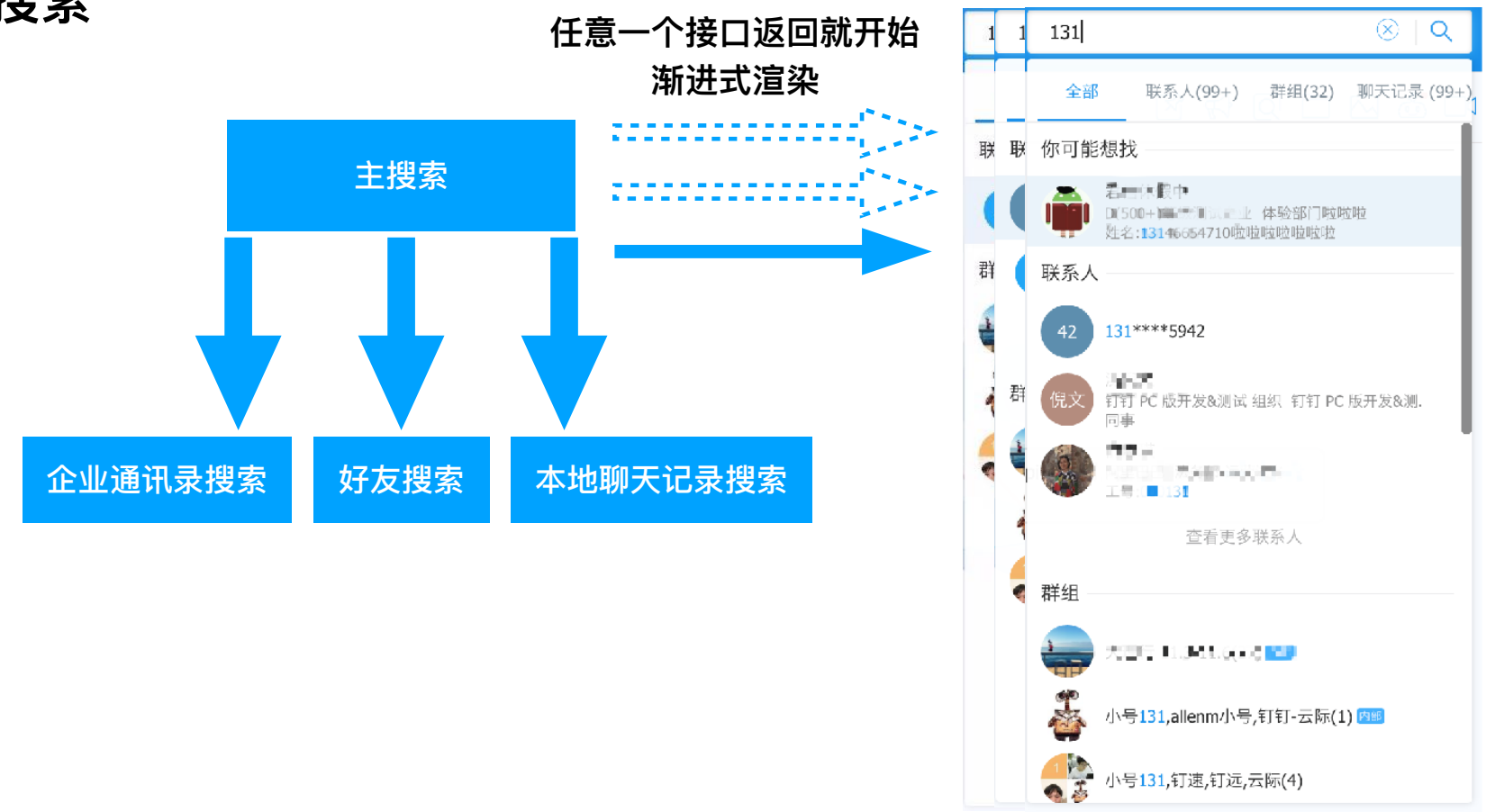
http://www.typescriptlang.org/docs/handbook/migrating-from-javascript.html

http://www.typescriptlang.org/docs/handbook/integrating-with-build-tools.html

异步逻辑的处理

异步逻辑的处理

渐进式搜索



渐进式搜索

```
var mainSearch = function(keyword) {
    var workerList = [
        orgSearch(keyword),
        friendSearch(keyword),
        localMsgSearch(keyword)
    ];
    var result = [];
    workerList.forEach(function(item){
        item.then(function (data) {
            result.push(data);
            if (result.length === workerList.length) {
                searchEventEmitter.emit('finish', rebuildSearchResult(result));
            } else{
                searchEventEmitter.emit('update', rebuildSearchResult(result));
            }
        })
    });
};
```

对于只有一次返回值的异步 API，用 Promise 很适合。

mainSearch 需要有多次 emit value，Promise 无法使用，只能使用事件或者 callback

渐进式搜索

```
var onKeywordChange = function(keyword) {
  mainSearch(keyword);
  this.isSearching = true;
};

searchEventEmitter.on('update', function (result) {
  this.searchResult = result;
});

searchEventEmitter.on('finish', function (result) {
  this.searchResult = result;
  this.isSearching = false;
});
```

Bug: Race Condition

渐进式搜索

```
var mainSearch = function(keyword, taskId) {
    var workerList = [
        orgSearch(keyword),
        friendSearch(keyword),
        localMsgSearch(keyword)
    ];
    var result = [];
    workerList.forEach(function(item){
        item.then(function (data) {
            result.push(data);
            if (result.length === workerList.length) {
                searchEventEmitter.emit('finish', rebuildSearchResult(result), taskId);
            } else{
                searchEventEmitter.emit('update', rebuildSearchResult(result), taskId);
            }
        })
    });
};
```

渐进式搜索

```
var taskId = 0;
var onKeywordChange = function(keyword) {
    taskId++;
    mainSearch(keyword, taskId);
    this.searchResult = {};
    this.isSearching = true;
};

searchEventEmitter.on('update', function (result, _taskId) {
    if (_taskId === taskId) {
        this.searchResult = result;
    }
});

searchEventEmitter.on('finish', function (result, _taskId) {
    if (_taskId === taskId) {
        this.searchResult = result;
        this.isSearching = false;
    }
});
```

增加状态判断，解决 Race Condition

如果使用 RxJS

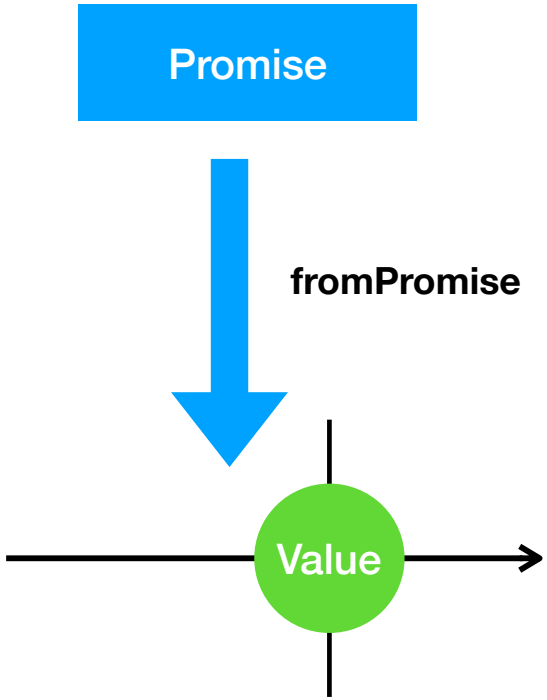
渐进式搜索

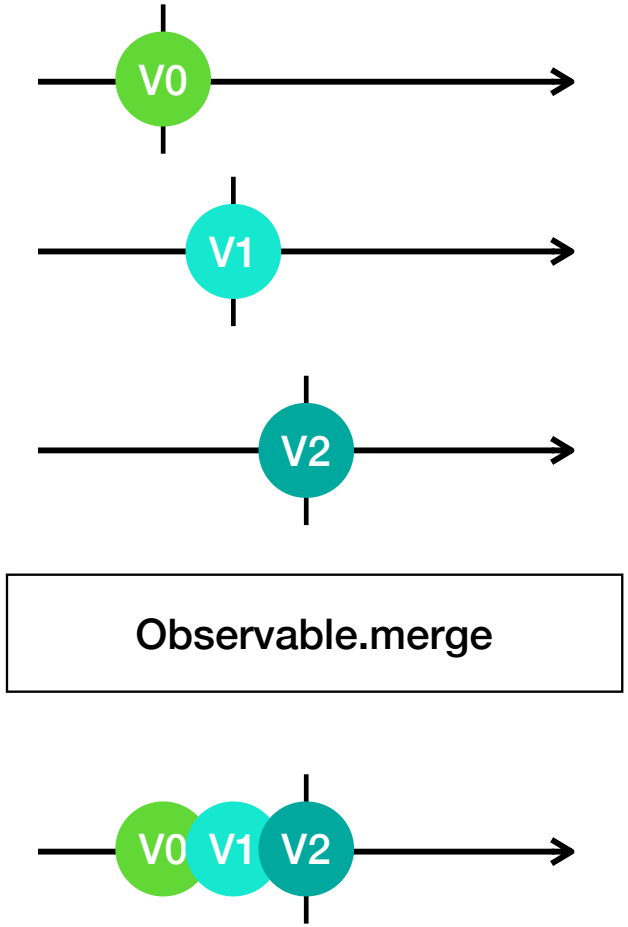
```
const mainSearch = function(keyword){
    return Rx.Observable.merge(
        Rx.Observable.fromPromise(orgSearch(keyword)),
        Rx.Observable.fromPromise(friendSearch(keyword)),
        Rx.Observable.fromPromise(localMsgSearch(keyword))
    ).scan((acc, value) => {
        if (value.type === 'org') {
            acc.org = value.data;
        } else if (value.type === 'friend') {
            acc.friend = value.data;
        } else if (value.type === 'localMsg') {
            acc.localMsg = value.data;
        }
        return acc;
    }, {});
};
```

类似 Array.prototype.reduce,
替换前面例子里的 rebuildSearchResult

Observable 非常好的描述了这种需要多次 emit value 的异步 API

渐进式搜索





圆表示 next value，竖线表示 complete，箭头表示时间

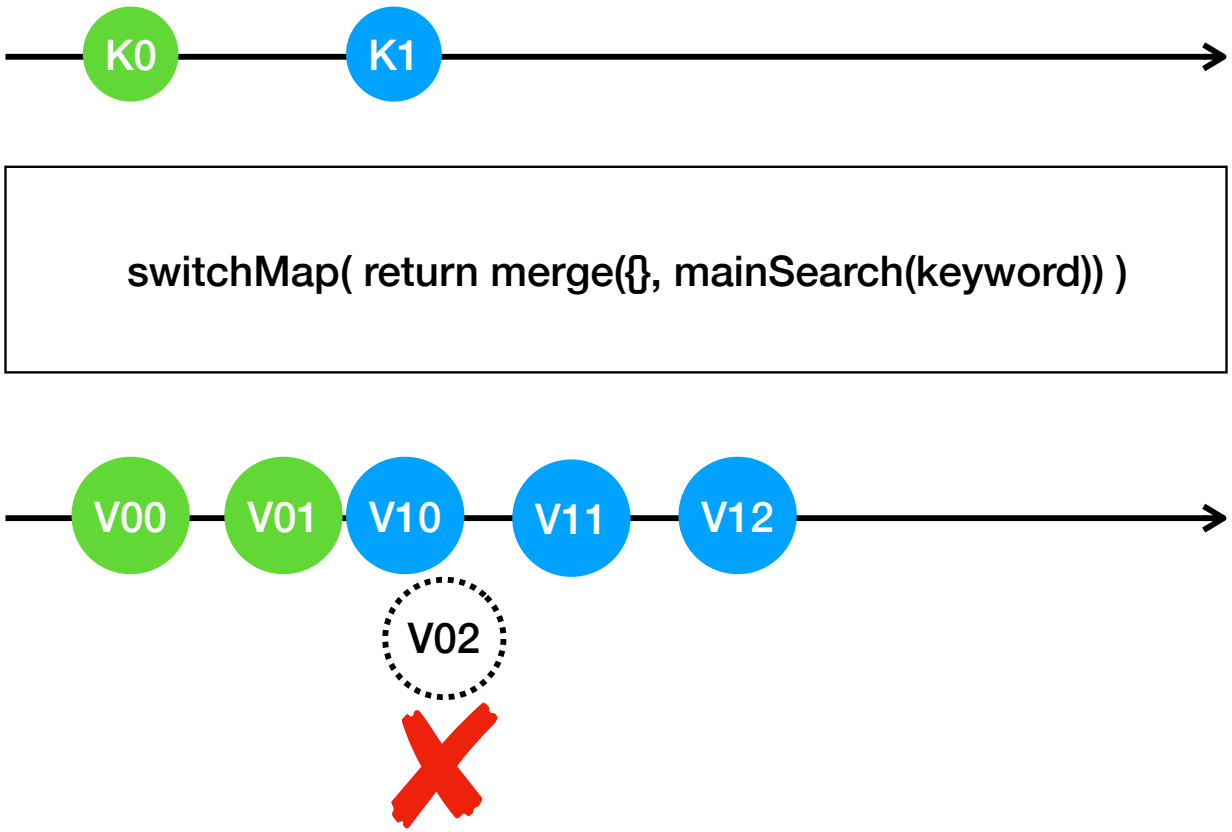
渐进式搜索

```
const keyword$ = new Rx.Subject();
const onKeywordChange = function(keyword) {
    keyword$.next(keyword);
};
```

解决 Race Condition

```
keyword$.switchMap((keyword) => {
    return Rx.Observable.merge(
        Rx.Observable.of({}),
        mainSearch(keyword)
    ).materialize();
}).subscribe((item) => {
    if (item.kind === 'N') {
        this.isSearching = true;
        this.searchResult = item.value;
    } else if (item.kind === 'C') {
        this.isSearching = false;
    }
});
```

渐进式搜索



switchMap 自动忽略之前的 projected Observable 产生的值，是解决 Race Condition 的好帮手

异步调用

渐进式业务

- 搜索
- 文件上传，下载
- Profile 详情。 基本 Profile -> 完整 Profile

需要 Cancel 的异步 API

- 在 Component 生命周期里工作的异步 API
- 用户交互产生的需要 Cancel 的场景

使用 Rx.js Observable 可以更好更直观的描述这些 API,
帮助你在错综复杂的异步 API 调用中少犯错

钉钉

Thanks！

Q&A

钉钉上搜索 allenm56 和我交流