UmiJs

什么是UmiJs?

杨大侠

为什么会有框架&解决了什么问题?

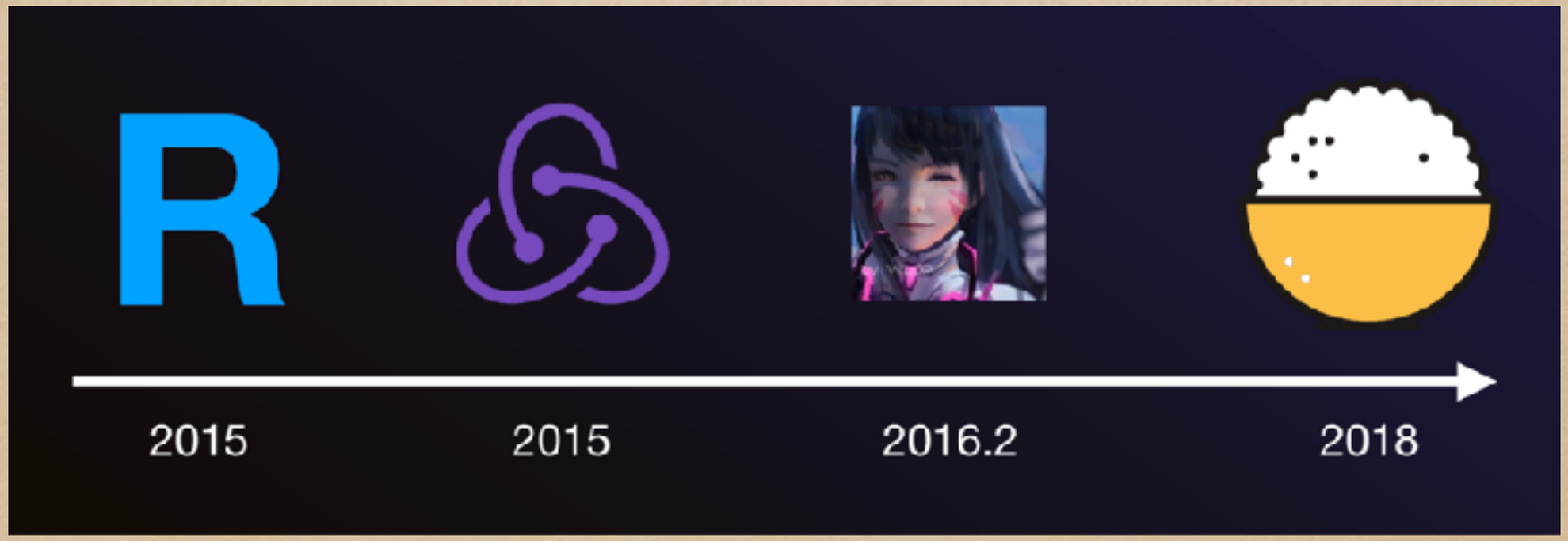
先和我去看几个demo吧

- 开发周期缩短
- 前后端分离
- 组件化开发

Umi的前世今生

阿里巴巴是从2015年从Jq和模板引擎转入React技术栈，就开始了疯狂的对开源社区做贡献。如antd、antV、DvaJS、ant-design-pro（综合后台管理架构模版）、在到今天的主角UmiJS,每一次升级都是对开发效率的一次综合提升。

从 Roof 到 Redux 到 DvaJS，再到 UmiJS之间的演变



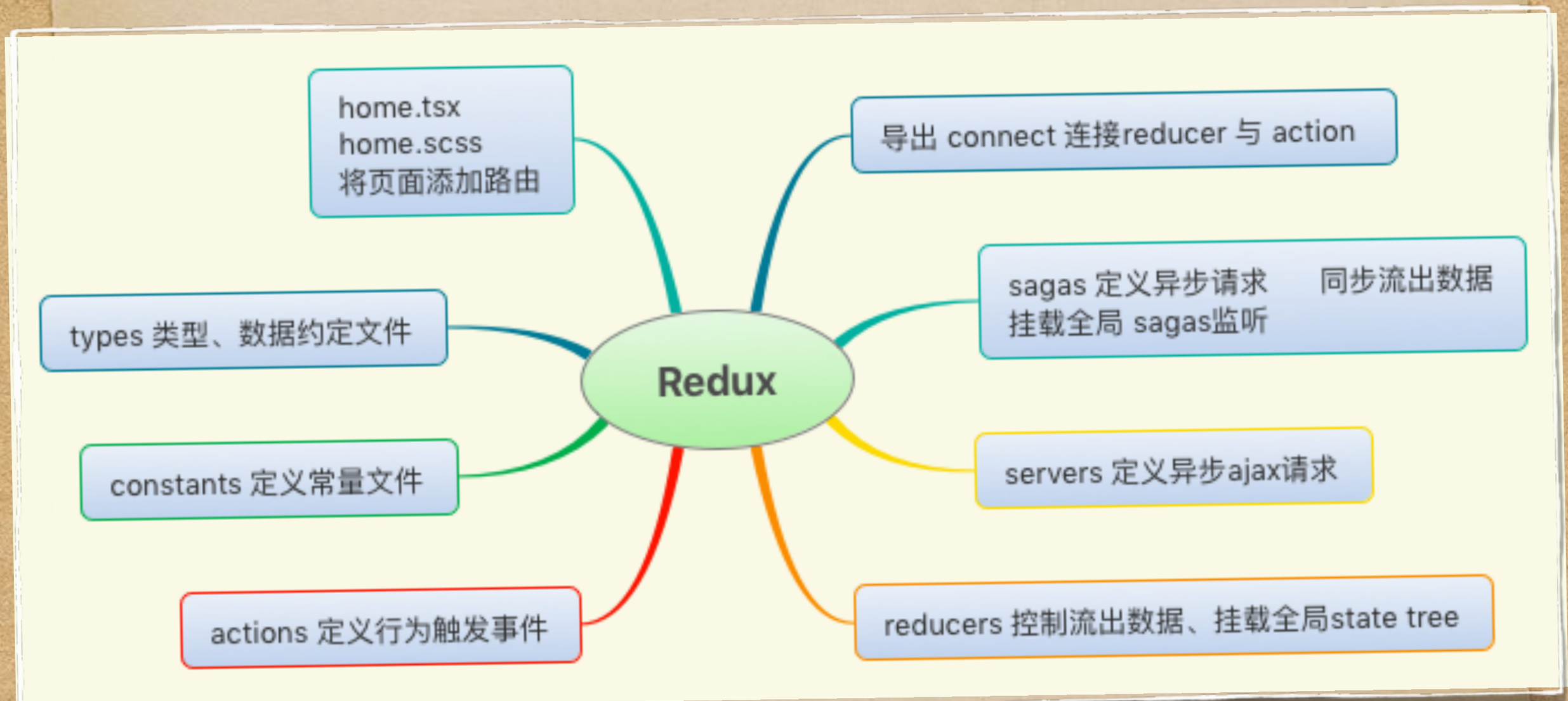
Roof

```
// 定义 state
createRootContainer({
  user: { name: 'chris', age: 30 }
})(App);

// 绑定 state
createContainer({
  myUser: 'user',
})(UserInfo);
```

拥抱社区很重要！！！

Redux基本套路

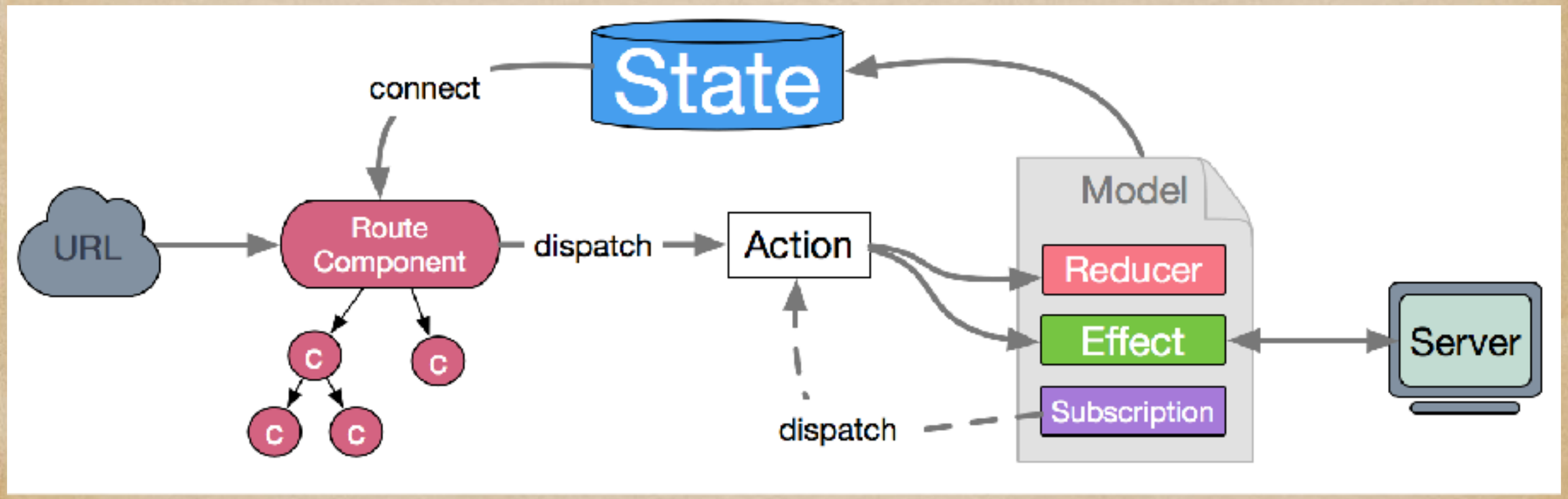

home.tsx
home.scss
将页面添加路由
导出 connect 连接reducer 与 action
types 类型、数据约定文件
sagas 定义异步请求　　同步流出数据
挂载全局 sagas监听
Redux
constants 定义常量文件
servers 定义异步ajax请求
actions 定义行为触发事件
reducers 控制流出数据、挂载全局state tree

Redux痛点

- 文件参差不齐相互依赖共存
- 创建redux-saga繁琐
- 手写Redux配置项麻烦

DvaJs是什么鬼，是不是又要学习一套新的理念！以及API？走跟我去看下面的介绍吧!

Dva.js


State
connect
URL
Route
Component
C
C
C
C
dispatch
Action
dispatch
Model
Reducer
Effect
Subscription
Server

Dva解析

- namespace action名称
- state保存数据
- reducers流入数据
- effects操作异步&复杂逻辑
- subscriptions监听路由的变化

大马路&FCT

```
import { list } from "../Services/index";

export default {
  namespace: 'StoreList',
  state: {
  },
  reducers:{
  },
  effects: {
  },
  // 订阅数据
  subscriptions: {
    //history 监听路由变化、dispatch 触发 effects与reducers 的方法
    setup({ dispatch, history }) {
      history.listen(({ pathname }) => {
        // console.log(pathname)

      });
    },
  }
};
```

ajax 异步封装请求

connect 连接业务模块

业务模块 连接 props 数据

触发 action 流入业务模块数据

触发 异步 ajax 请求

监听 url 变化

connect & action

```
export default connect(({ StoreList }) => {
  return {
    list: StoreList.list,
    total_count: StoreList.total_count,
    page: StoreList.page,
    page_size: StoreList.page_size,
    loaded: StoreList.loaded,
    errored: StoreList.errored,
    store_name: StoreList.store_name,
    status: StoreList.status,
    city: StoreList.city
  }
})(StoreListTable);
```

```
this.props.dispatch({
  type: "StoreList/getStoreList",
  payload: {
    page_size: this.props.page_size,
    page: page,
    store_name: this.props.store_name,
    status: this.props.status,
    city: this.props.city
  }
})
```

连接 connect 导入数据　　触发 action异步请求

namespace & state

```
namespace: 'StoreList',
state: {
  list:[],
  total_count:0,
  page:1,
  page_size: 10,

  loaded: false,
  errored: false,

  store_name: "",
  status: "",
  city: "",

},
```

约定modal name　定义初始化数据

reducers

```
reducers:{
  list(state, { payload: {list, total_count, page, store_name, status, city} }) {
    // 处理之后返回
    return {
      ...state,
      total_count:total_count,
      page: page,
      store_name: store_name,
      status: status,
      city: city,
      list: list.map((item,index) => {
        return {
          ...item,
          key: index
        }
      })
    }
  },

  loaded(state, { payload }) {
    return {
      ...state,
      ...payload
    }
  },
},
```

控制reducer 数据流向

effects

```
effects: {
  *getStoreList({ payload }, { put, call,}) {

    yield put({
      type:"loaded",
      payload:{
        loaded: false,
        errored: false,
        list:[]
      }
    })

    const result = yield call(list, {...payload});

    if (result.data) {
      yield put({
        type:"loaded",
        payload:{
          loaded: true,
          errored: false,
        }
      })
      yield put({
        type: 'list',
        payload: {
          list: result.data.list,
          total_count: result.data.total_count,
          page: payload.page,
          store_name:payload.store_name,
          status:payload.status,
          city: payload.city
        }
      });
    }else{
      yield put({
        type:"loaded",
        payload:{
          loaded: true,
          errored: true,
        }
      })
    }
  }
},
```

subscriptions

```
// 订阅数据
subscriptions: {
  //history 监听路由变化、dispatch 触发 effects与reducers 的方法
  setup({ dispatch, history }) {
    history.listen(({ pathname }) => {
      // console.log(pathname)

    });
  },
}
```

监听路由变化

她到底啥东西?

中文可发音为 “乌米” 、每一种新的框架与技术出现总会带来一次新的学习成本，比如css基础上出现了预编译语言less与sass。而她的出现并不是这样， **她** 的出现只会让我们关注业务层，不会让开发者各种配置才能构建出一套完整的符合项目需求的架构。只需一行代码生成cil即可快速搭建属于公司的框架。

umi 是工具吗？ 是。但不仅仅是工具。作者对 umi 的定位是开发框架，包含工具 + 路由，不包含数据和视图。

umi 以路由为基础搭建，支持约定式路由配置，完善的各种插件供我们选择使用便捷我们的开发。

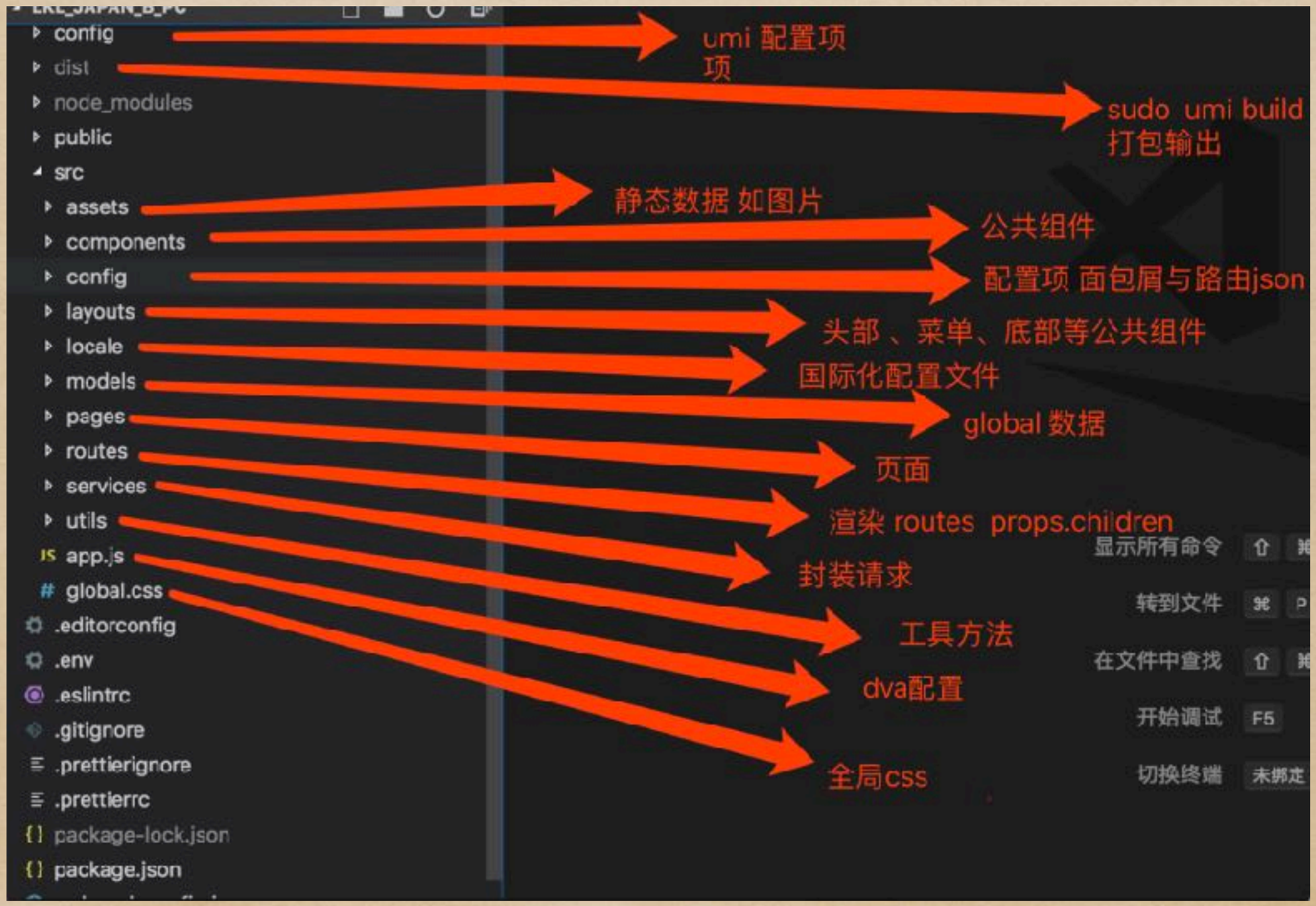
一行命令生成包含DvaJs，React，React-route、Antd等公司开发常用的所有npm，重点提示一下 **所有依赖npm**

核心竞争力

- 基于路由
- HTML是串联业务流程的粘合剂
- 插件机制

这个框架会火，我觉得她的Star能和阿里的股票一样🔥

目录解析


config
dist
node_modules
public
src
assets
components
config
layouts
locale
models
pages
routes
services
utils
app.js
global.css
.editorconfig
.env
.eslintrc
.gitignore
.prettierignore
.prettierrc
package-lock.json
package.json
umi 配置项
项
sudo umi build
打包输出
静态数据 如图片
公共组件
配置项 面包屑与路由json
头部 、菜单、底部等公共组件
国际化配置文件
global 数据
页面
渲染 routes props.children
封装请求
工具方法
dva配置
全局css
显示所有命令
转到文件
在文件中查找
开始调试
F5
切换终端
未绑定

单页面解析


StoreList
Components
Models
Services
StoreList.js
pege 业务模块组件
page 业务逻辑
封装页面请求
page

万物皆套路

以上目录解析，包含umi 的核心，顶层models 包含全局 global ，每个单独页面的page文件里面包含 models，这样的优势就是动态挂载modal，也是umi对性能的一种优化方案，类似于路由按需加载。配置页面子路由可以通过无状态组件渲染填充业务模块 { props.children }

- mock数据
- 一键兼容IE9
- 完善TypeScript支持
- 万物皆路由
- model为核心概念

三克油