React全家桶2

课堂目标

1. 深入理解react全家桶
2. 掌握redux解决方案--dva
3. 掌握generator
4. 掌握umi

资源

1. umi
2. dva

知识要点

回顾redux

redux中间件机制

```
export function createStore(reducer, enhancer){
    if (enhancer) {
        return enhancer(createStore)(reducer)
    }
    let currentState = {}
    let currentListeners = []

    function getState(){
        return currentState
    }
    function subscribe(listener){
        currentListeners.push(listener)
    }
    function dispatch(action){
        currentState = reducer(currentState, action)
        currentListeners.forEach(v=>v())
        return action
    }
    dispatch({type:'@IMOOC/WONIU-REDUX'})
    return { getState, subscribe, dispatch}
}

export function applyMiddleware(...middlewares){
    return createStore=>(...args)=>{
        const store = createStore(...args)
        let dispatch = store.dispatch

        const midApi = {
            getState:store.getState,
            dispatch:(...args)=>dispatch(...args)
        }
        const middlewareChain = middlewares.map(middleware=>middleware(midApi))
        dispatch = compose(...middlewareChain)(store.dispatch)
        return {
            ...store,
            dispatch
        }

    }
}
```

```
export function compose(...funcs){
    if (funcs.length==0) {
        return arg=>arg
    }
    if (funcs.length==1) {
        return funcs[0]
    }
    return funcs.reduce((ret,item)=> (...args)=>ret(item(...args)))
}
function bindActionCreator(creator, dispatch){
    return (...args) => dispatch(creator(...args))
}
export function bindActionCreators(creators,dispatch){
    return Object.keys(creators).reduce((ret,item)=>{
        ret[item] = bindActionCreator(creators[item],dispatch)
        return ret
    },{})
}
```

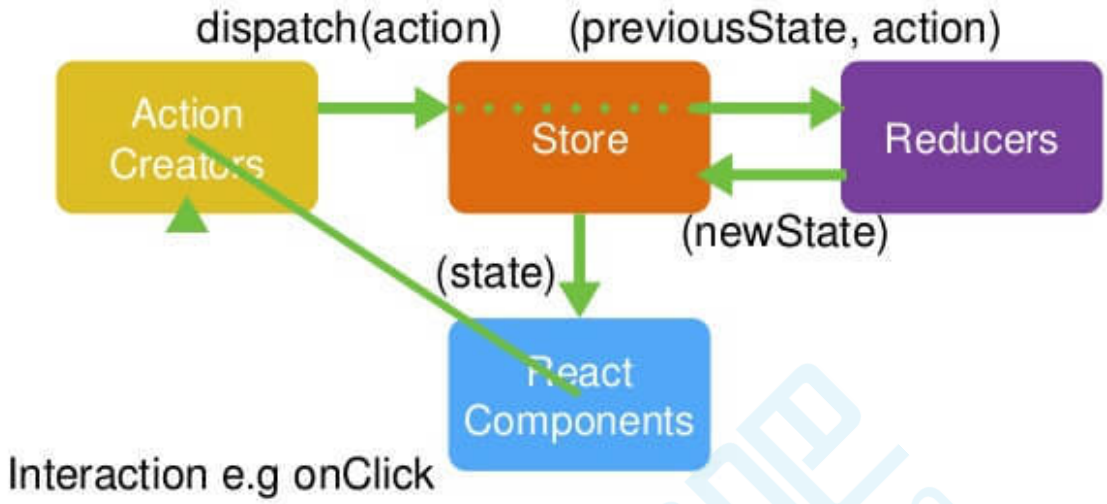
```
import React from 'react'
import PropTypes from 'prop-types'
import {bindActionCreators} from './woniu-redux'

export const connect =
(mapStateToProps=state=>state,mapDispatchToProps={})=>(WrapComponent)=>
{
    return class ConnectComponent extends React.Component{
        static contextTypes = {
            store:PropTypes.object
        }
        constructor(props, context){
            super(props, context)
            this.state = {
                props:{}
            }
        }
        componentDidMount(){
            const {store} = this.context
            store.subscribe(()=>this.update())
            this.update()
        }
        update(){
            const {store} = this.context
            const stateProps = mapStateToProps(store.getState())
            const dispatchProps =
bindActionCreators(mapDispatchToProps, store.dispatch)
            this.setState({
                props:{
                    ...this.state.props,
```

```
                    ...stateProps,
                    ...dispatchProps
                }
            })
        }
        render(){
            return <WrapComponent {...this.state.props}>
</WrapComponent>
        }
    }
}

export class Provider extends React.Component{
    static childContextTypes = {
        store: PropTypes.object
    }
    getChildContext(){
        return {store:this.store}
    }
    constructor(props, context){
        super(props, context)
        this.store = props.store
    }
    render(){
        return this.props.children
    }
}
```

```


const thunk = ({dispatch,getState})=>next=>action=>{
    if (typeof action=='function') {
        return action(dispatch,getState)
    }
    return next(action)
}
export default thunk
```

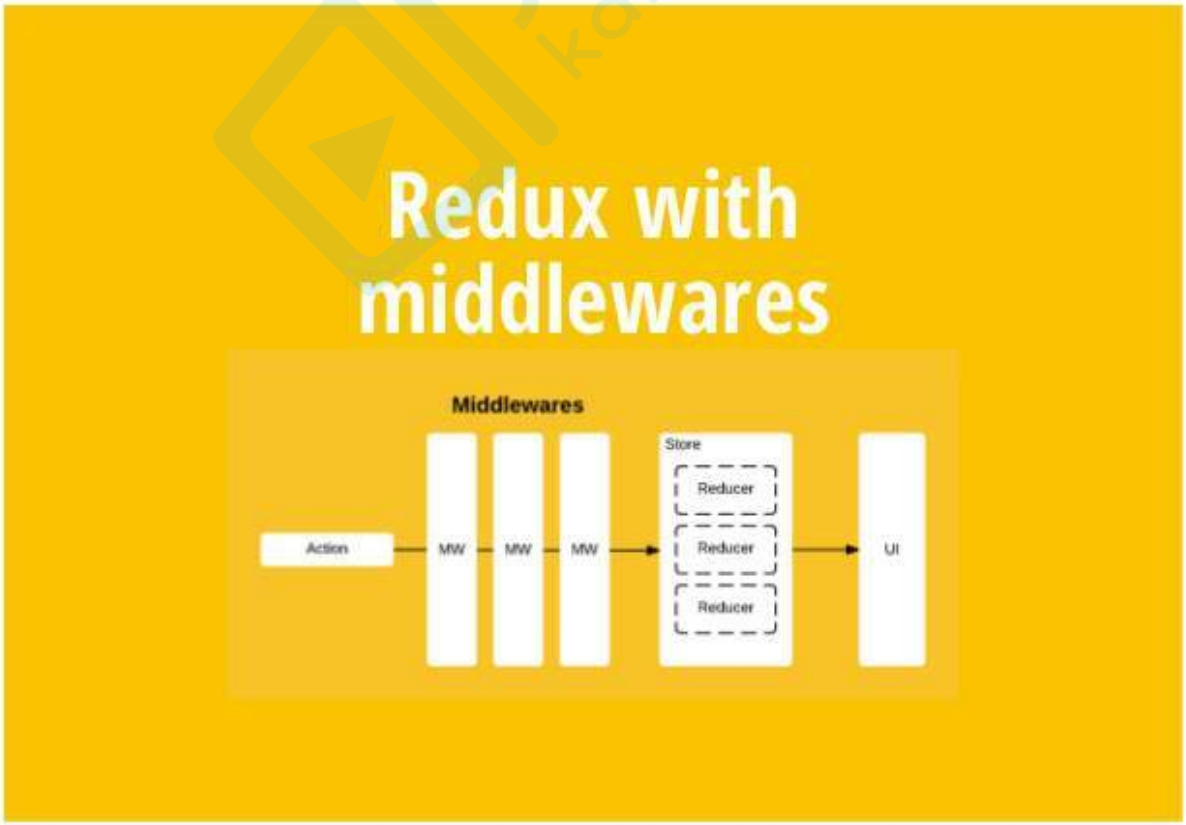
redux流程梳理

Redux Flow



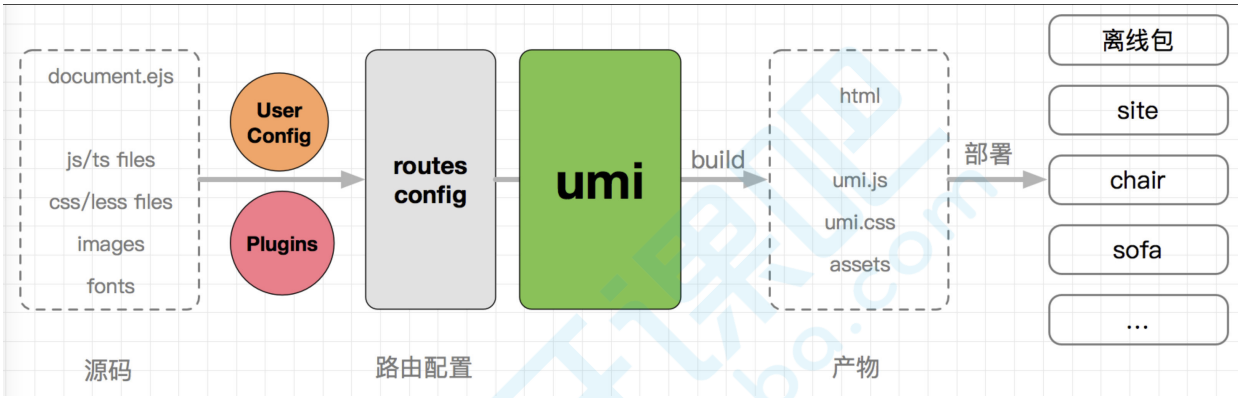
React + Redux

@nikgraf

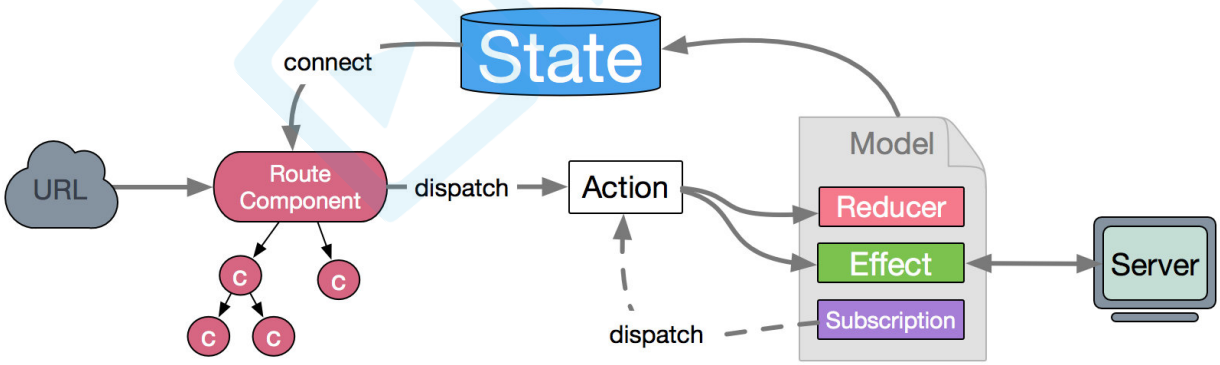


umi

- 📦 **开箱即用**，内置 react、react-router 等
- 🏈 **类 next.js 且功能完备的路由约定**，同时支持配置的路由方式
- 🎉 **完善的插件体系**，覆盖从源码到构建产物的每个生命周期
- 🚀 **高性能**，通过插件支持 PWA、以路由为单元的 code splitting 等
- 💈 **支持静态页面导出**，适配各种环境，比如中台业务、无线业务、egg、支付宝钱包、云凤蝶等
- 🛫 **开发启动快**，支持一键开启 dll 和 hard-source-webpack-plugin 等
- 🐟 **一键兼容到 IE9**，基于 umi-plugin-polyfills
- 🍁 **完善的 TypeScript 支持**，包括 d.ts 定义和 umi test
- 🌴 **与 dva 数据流的深入融合**，支持 duck directory、model 的自动加载、code splitting 等等



dva



dva+umi 的约定

1. src 源码
    1. pages页面
    2. components 组件
    3. layout布局
    4. model
2. config 配置

3. mock 数据模拟
4. test测试等

项目骨架

```
npm init
npm install umi --save
```

新建config/config.js

```
export default {}
```

新建 pages/index.js

```
export default () => {
  return <div>hello world</div>;
}
```

新增pageage.json的scripts

```
"dev": "umi dev",
"build": "umi build"
```

默认路由为声明式，pages下面会自动生成路由，更建议大家用配置，嵌套和参数 适合复杂项目

修改config.config.js

```
export default {

  routes:[
    {
      path:'/',
      component:"./App"
    },
    {
      path:'/about',
      component:"./About"
    },
    {
      component:"./404"
    }
  ]
}
```

对应App.js,About.js， 404为错误页面

添加 umi-plugin-react 和antd插件

`npm install umi-plugin-react antd --save`

修改config.js

```
  plugins: [
    ['umi-plugin-react', {
      antd: true
    }],
  ],
```

```
import {Button} from 'antd'
export default () => {
  return <div>
  <Button type='primary'>123</Button>
  </div>

}
```

布局

新建layout/index.js

```
import React from 'react'
import { Layout } from 'antd'

// Header, Footer, Sider, Content组件在Layout组件•模块下
const { Header, Footer, Sider, Content } = Layout

class BasicLayout extends React.Component {
render() {
  return (
    <Layout>
    <Sider width={256} style={{ minHeight: '100vh', color: 'white' }}>
      Sider
    </Sider>
    <Layout >
      <Header style={{ background: '#fff', textAlign: 'center',
padding: 0 }}>Header</Header>
      <Content style={{ margin: '24px 16px 0' }}>
```

```
        <div style={{ padding: 24, background: '#fff', minHeight: 360 }}>
          {this.props.children}
        </div>
      </Content>
      <Footer style={{ textAlign: 'center' }}>Ant Design ©2018 Created by Ant UED</Footer>
    </Layout>
  </Layout>
  )
}
}

export default BasicLayout
```

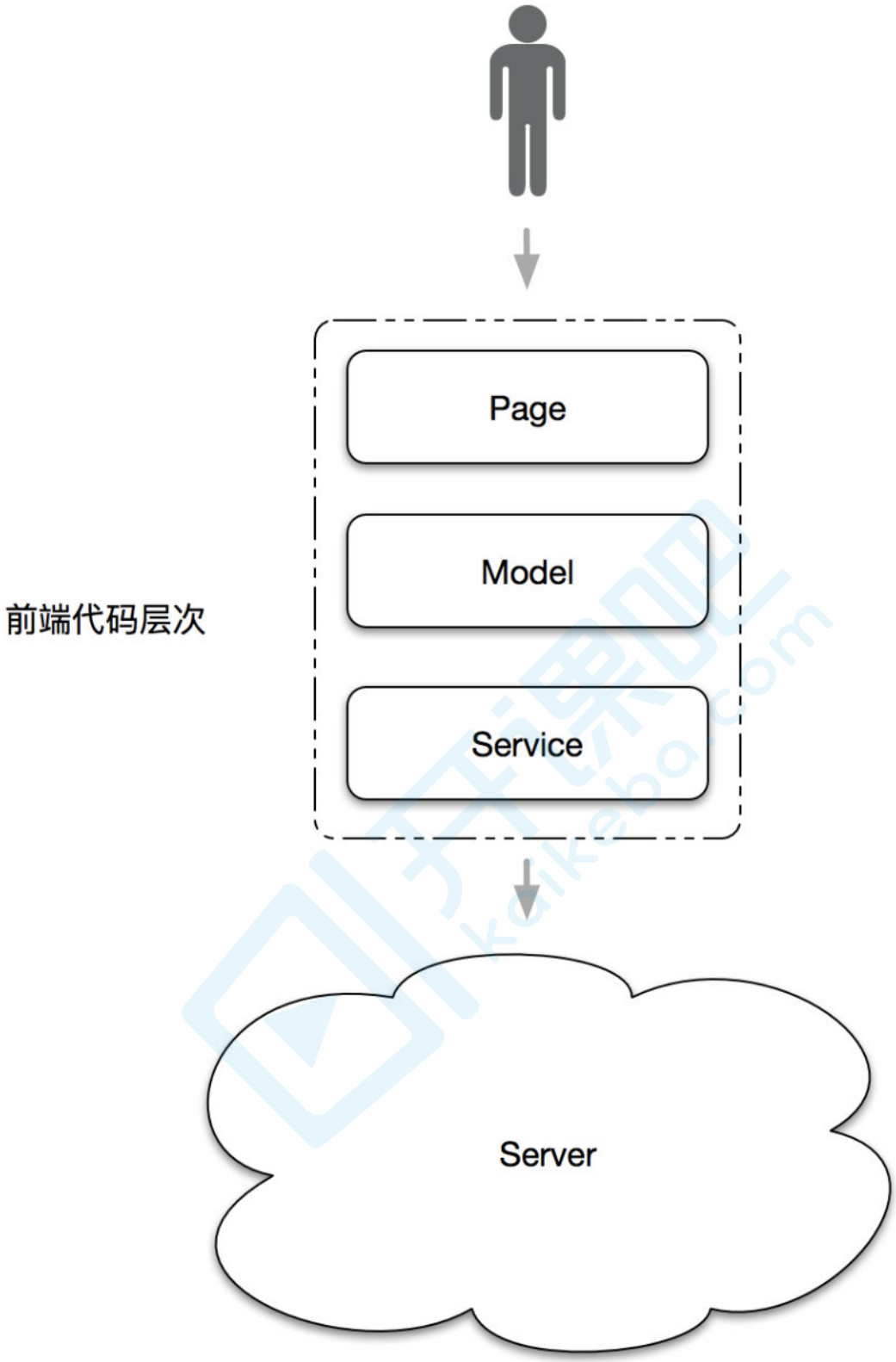
修改config.js

```
export default {
  plugins: [
    ['umi-plugin-react', {
      antd: true
    }],
  ],
  routes: [
    {
      path: '/',
      component: '../layout',
      // 嵌套路由
      routes: [
        {
          path: '/',
          component: "./App"
        },
        {
          path: 'about',
          component: "./About"
        },
        {
          component: "./404"
        }
      ]
    }

  ]
}
```

引入dva

软件分层

回顾react，为了让数据流更易于维护，我们分成了store，reducer，action等模块，各司其职，软件开发也是一样



1. Page 负责与用户直接打交道：渲染页面、接受用户的操作输入，侧重于`展示型交互性逻辑`。
2. Model 负责处理业务逻辑，为 Page 做数据、状态的读写、变换、暂存等。
3. Service 负责与 HTTP 接口对接，进行纯粹的数据读写。

```
export default {
  plugins: [
    ['umi-plugin-react', {
      antd: true,
      dva: true,
    }],
  ],
  // ...
}
```

DVA 是基于 redux、redux-saga 和 react-router 的轻量级前端框架及最佳实践沉淀。核心api如下

1. model
    1. state
    2. action
    3. dispatch
    4. reducer
    5. effect 处理异步 saga中间件
    6. Subscriptions 订阅
    7. router 路由

model

新建src/model/goods.js

```
export default {

  namespace: 'goods',

  state: [],
  effects:{},
  reducers: {
    save(state, { payload: good }) {
      return [...state, good];
    },
  },

}
```

1. `namespace`：model 的命名空间，只能用字符串。一个大型应用可能包含多个 model，通过 `namespace` 区分。

2. `state`：当前 model 状态的初始值，表示当前状态。
3. `reducers`：用于处理同步操作，可以修改 `state`，由 `action` 触发。reducer 是一个纯函数，它接受当前的 state 及一个 action 对象。action 对象里面可以包含数据体（payload）作为入参，需要返回一个新的 state。
4. `effects`：用于处理异步操作（例如：与服务端交互）和业务逻辑，也是由 action 触发。但是，它不可以修改 state，要通过触发 action 调用 reducer 实现对 state 的间接操作。
5. `action`：是 reducers 及 effects 的触发器，一般是一个对象，形如 `{ type: 'add', payload: todo }`，通过 type 属性可以匹配到具体某个 reducer 或者 effect，payload 属性则是数据体，用于传送给 reducer 或 effect。

state+connect

app.js

```
import {Button,Card} from 'antd'
import React from 'react'
import {connect} from 'dva'

@connect(
  state=>({
    goodsList:state.goods
  }),
  dispatch=>{
    return {
      addGood(title){
        dispatch({
          type:"goods/addGood",
          payload:{title}
        })
      }
    }
  }
)
export default class App extends React.Component{

  render(){
    return <div>
        {
          this.props.goodsList.map(good => {
            return (
              <Card key={good.title}>

                <div>
                  <strong>{good.title}</strong>
                </div>
              </Card>
            );
```

```
          })
        }
        <div>
          <Button onClick={()=>this.props.addGood('添加卡片'+new Date().getTime())}> 添加卡片 </Button>
        </div>
    </div>
  }
}


```

Model/goods.js

```
export default {

  namespace: 'goods',

  state: [
    {title:"web全栈"},
    {title:"java架构师"},
    {title:"百万年薪"}
  ],
  effects:{},
  reducers: {
    addGood(state, {payload}) {
      return [...state, {title:payload.title}];
    },
  },

}
```

数据mock

mock目录和src平级，新建mock/goods.js

```
let data = [
  {title:"web全栈"},
  {title:"java架构师"},
  {title:"百万年薪"}
];

export default {
  'get /api/goods': function (req, res, next) {
    setTimeout(() => {
      res.json({
```

```
        result: data,
      })
    }, 250)
  },

}

```

ES6的generator函数

简单的说 就是带一个星星的函数，可以分段执行

```
  function* g() {
    yield 'a';
    yield 'b';
    yield 'c';
    return 'ending';
  }
  console.log(g())
```

g()并不执行g函数 而是返回一个迭代器对象 调用next实际执行

```
  function* g() {
    yield 'a';
    yield 'b';
    yield 'c';
    return 'ending';
  }
  // console.log(g())
  var gen = g()
  console.log(gen.next()) // {value: "a", done: false}
  console.log(gen.next()) // {value: "b", done: false}
  console.log(gen.next()) // {value: "c", done: false}
  console.log(gen.next()) // {value: "ending", done: true}
```

effect处理异步

saga要求使用generator函数来控制异步流程，不能用async+await

```
function request(url) {
  axios.get(url).then(function(response){
    it.next(response);

```

```
  })
}

function* ajaxs() {
    console.log(yield request('a.html'));
    console.log(yield request('b.html'));
    console.log(yield request('c.html'));
}

var it = ajaxs();

it.next();

```

```
import {getGoods} from '../service'
export default {

  namespace: 'goods',

  state: [
    // {title:"web全栈"},
    // {title:"java架构师"},
    // {title:"百万年薪"}
  ],
  effects:{
    *getList(payload,{call,put}){
      const goods = yield call(getGoods)
      yield put({ type: 'initGoods', payload: goods.data.result })
    }
  },
  reducers: {
    initGoods(state,{payload}){
      return [...state,...payload]
    },
    addGood(state, {payload}) {
      // 3. action到来之后，返回新的state
      return [...state, {title:payload.title}];
    },
  },

}
```

```
getList(){
  this.props.dispatch({
    type:"goods/getList",
  })
}
```

回顾

React全家桶2

- 课堂目标
- 资源
- 知识要点
  - 回顾redux
- redux中间件机制
- redux流程梳理
  - umi
  - dva
  - dva+umi 的约定
  - 项目骨架
  - 添加 umi-plugin-react 和antd插件
  - 布局
  - 引入dva
  - model
  - state+connect
  - 数据mock
  - ES6的generator函数
  - effect处理异步
- 回顾