基于STM32节点和阿里云IoT平台的物联网应用开发 系列课程

第四章

服务端的应用开发

life.augmented

阿里云

课程内容下载、观看

- 视频观看：AI电堂、阿里云大学IoT课堂
- 课件胶片下载：STMCU中文官网、阿里云大学IoT课堂
- 课件项目下载：STMCU中文官网、阿里云大学IoT课堂

STM32公众号

STM中文官网

电堂公众号

阿里云大学

第四章内容简介

- 第一节：综合软件架构介绍
  - 软件架构，知识结构梳理
- 第二节：后端服务开发
  - 后端框架，初始化首个后端项目，应用系统开发，部署
- 第三节：前端服务开发体验
  - 前端框架，初始化首个前端项目，组件的使用和数据流转，部署

STM32-阿里云IoT 联合课件开发

第四章．第三节
前端服务开发

life.augmented

阿里云

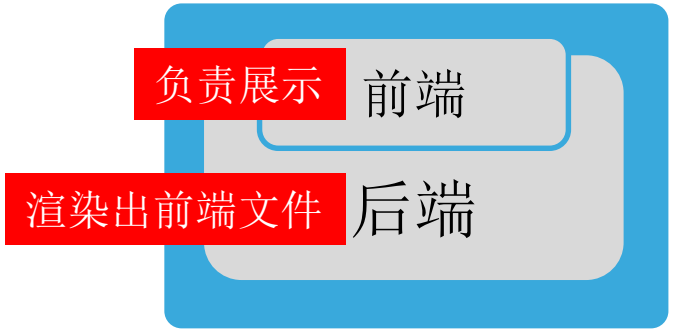
第四章.第三节 内容简介

- 前端基础概念和知识
- 认识前端框架
  - 了解React
  - 学习Umi.js
  - 学习Ant Design
  - 了解dva.js
- 初始化并运行第一个后端项目
  - 安装yarn包管理工具
  - 全局安装umi
  - 使用脚手架初始化
  - Umi路由页面添加
- 创建和使用组件
  - Layout组件
  - 主页面创建
  - 组件引入
  - 组件完善
- 使用dva实现数据流转
  - 数据请求
  - Dva管理数据
- 应用调试与部署
  - 项目打包
  - 项目部署

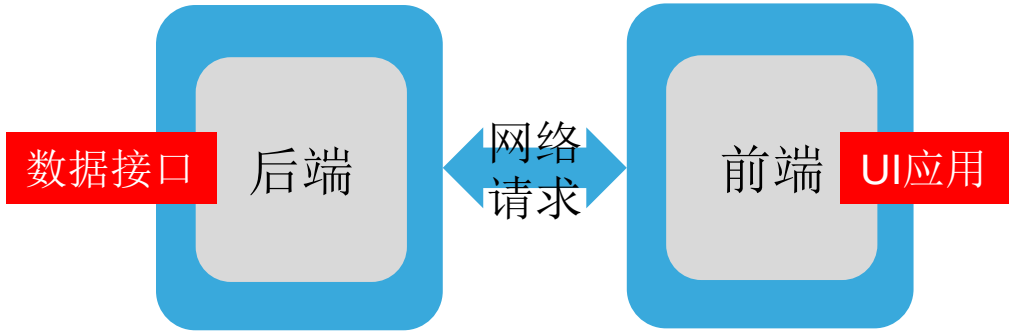
了解前端基础概念和知识技能

- 前端的基本概念

在软体架构和程序设计领域，前端是软件系统中**直接和用户交互的部分**，而后端控制着软件的输出。将软件分为前端和后端是一种将软件不同功能的部分相互分离的抽象。



通过jsp、jinja等技术，后端接收到请求后将前端页面“渲染出来”返回给浏览器。



随着浏览器的性能逐渐强大，运行在浏览器的JavaScript代码通过ajax等技术向后端进行网络请求，实现了前后端的分离。

了解前端基础概念和知识技能

- 前端的开发语言

HTML

HTML即超文本标记语言 ，是用来描述网页的一种语言，与编程语言不同，标记语言用来记录信息而非执行逻辑处理，HTML语言的内容被各类标签所包裹

```
<html>
<body>
<h1>我的第一个标题</h1>
<p>我的第一个段落。</p>
</body>
</html>
```

JavaScript

JavaScript是一种直译式的脚本语言，是一种动态的解释形语言。JavaScript不需要经过编辑为机器码再运行，而是直接可以由解释器运行，它的解释器被称为JavaScript引擎，内置在各类浏览器中，JavaScript最早是在HTML网页上使用，用来给HTML网页增加动态功能，目前也已经拓展到服务端与硬件端。

```
<Script>{JS内容}</Script>
```

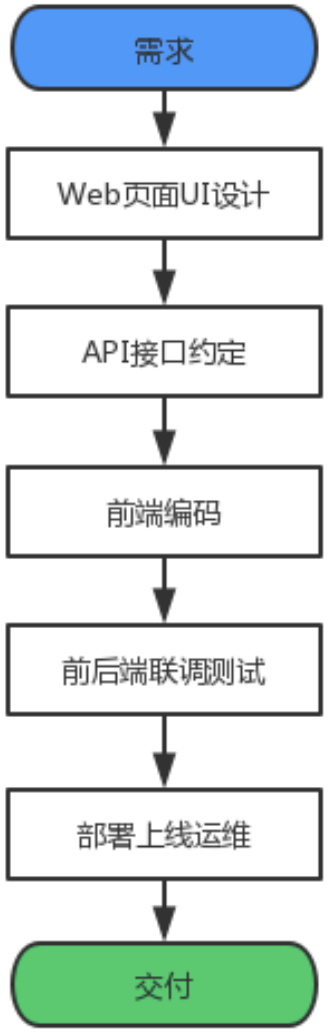
CSS

CSS是 指层叠样式表，定义如何显示 HTML 元素，一般存储在.css后缀的文件中，通过HTML标签中的className以及id属性来进行绑定。

```
h1 {color:red; font-size:14px;}
```

以上CSS代码声明了HTML中<h1>标签的文字颜色为红色，字体大小为14px，凡是在引入此CSS文件的HTML页面，被<h1></h1>标签包裹的字体都会被此CSS所影响。

前端Web应用的开发流程



明确前后端系统需要实现的功能

前端页面具象化，明确页面样式、交互功能

前后端约定API调用接口数据格式

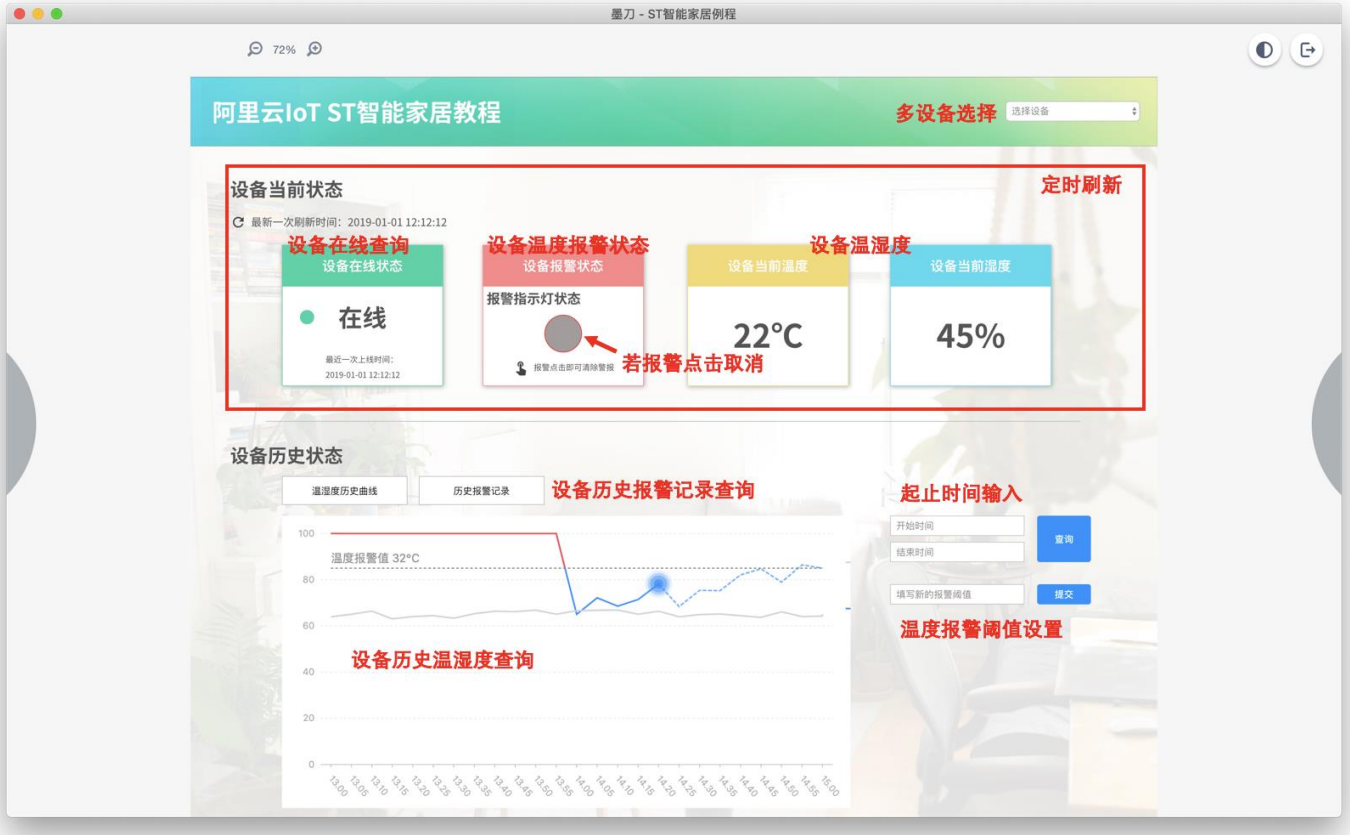
前端代码编写

通过API接口进行前后端功能联调

前端程序部署到服务器，并依据系统能力进行扩容与负载均衡

软件交付验收

前端UI设计



life.augmented

阿里云

前端UI设计

最终编码输出

life.augmented

阿里云

API接口约定

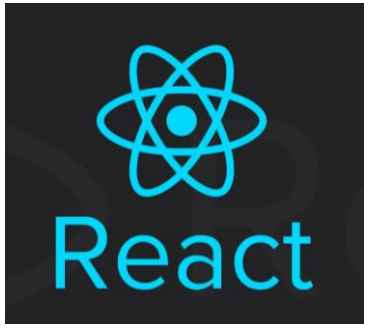
API名称	API内容
clearAlarm	下发命令取消设备的报警状态
listDeviceName	查询设备号，请求设备列表
queryAlarmHistoryLogs	查询设备历史报警记录
queryDeviceProp	查询设备属性（温湿度数据）
queryDevicePropHistoryLogs	查询设备历史属性（温湿度数据）
setDeviceProperty	设定设备属性（温度报警阈值）

第四章.第三节 内容简介

- 前端基础概念和知识
- 认识前端框架
  - 了解React
  - 学习Umi.js
  - 学习Ant Design
  - 了解dva.js
- 初始化并运行第一个前端项目
  - 安装yarn包管理工具
  - 全局安装umi
  - 使用脚手架初始化
  - Umi路由页面添加
- 创建和使用组件
  - Layout组件
  - 主页面创建
  - 组件引入
  - 组件完善
- 使用dva实现数据流转
  - 数据请求
  - Dva管理数据
- 应用调试与部署
  - 项目打包
  - 项目部署

认识前端框架

- React框架
- Umi.js框架
- Ant Design UI组件
- Dva.js数据流方案



Umi.js

Ant Design

dva.js

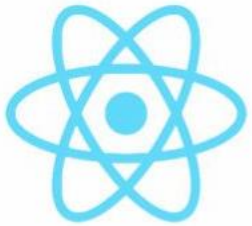
Reactæ¡æ¶æä¾åºå±æ¯æ¯æï¼

Umi.jsæ¡æ¶éæäºé¡µé¢è·¯ç±ãé¡¹ç®æåçå·¥å·ï¼ç®åå¼åèéç½®ä¸ç¹ççæä½ï¼

Ant Designæä¾äºä¸°å¯çé¡µé¢UIç»ä»¶åºï¼

dva.jsæä¾æ°æ®æµæ¹æ¡ï¼å°UIåæ°æ®è§£èï¼æé«åç«¯çå¼åæçã

React技术方案



React

React（React.js或ReactJS）是一个为数据提供渲染为HTML视图的开源 JavaScript 库。由Facebook在2013年instgram项目上进行开源，具有以下特点：

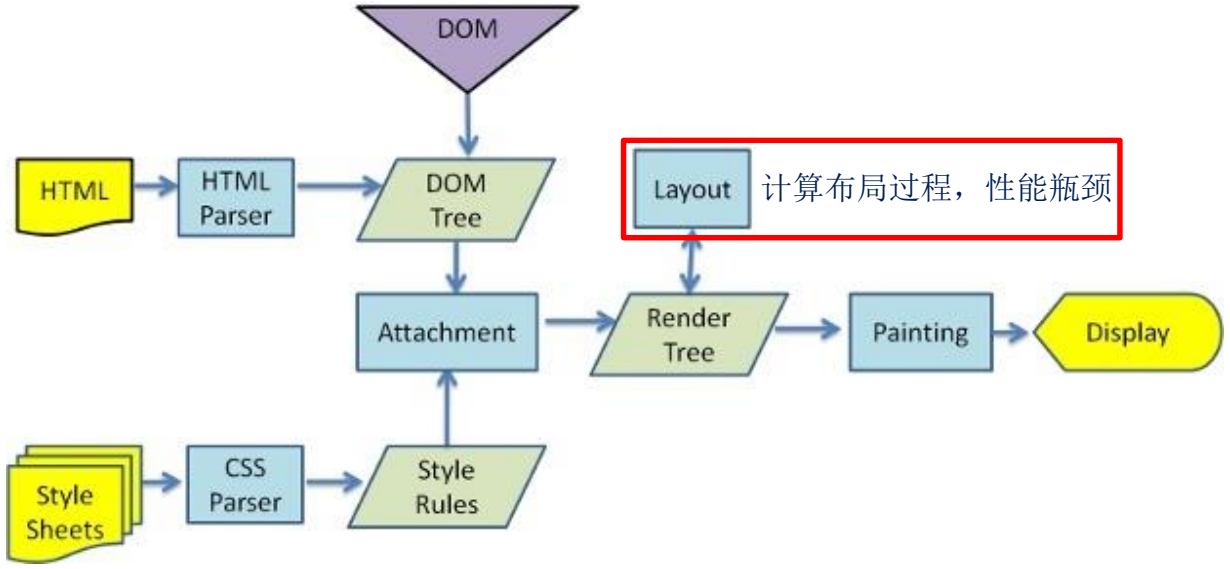
1. 声明式设计：React采用声明范式，屏蔽底层，开发者可以轻松描述构建应用。
2. 高效：React通过对DOM的模拟，最大限度地减少与DOM的交互，从而提升页面性能。
3. 灵活：React可以与已知的库或框架很好地配合。
4. JSX ：JSX 是 JavaScript 语法的扩展，即为JavaScript和XML的混合体。
5. 组件：通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。
6. 单向响应的数据流：React 实现了单向响应的数据流，从而减少了重复代码。

React技术方案—虚拟DOM

浏览器是如何呈现一个页面的

- 解析HTML，并生成一棵DOM tree
- 解析各种样式并结合DOM tree生成一棵Render tree
- 对Render tree的各个节点计算布局信息，比如box的位置与尺寸
- 根据Render tree并利用浏览器的UI层进行绘制

浏览器webkit引擎渲染过程



DOM（文档对象模型）

DOM是HTML、XML文档的JavaScript对象模型，提供丰富的对象操作与查询接口。浏览器通过HTML解析器将HTML文件解析成为DOM树并送到下游功能块渲染，JavaScript脚本可以通过DOM提供的接口直接对浏览器中的DOM树进行重构，从而引发下游模块的重新计算与渲染（Layout）。

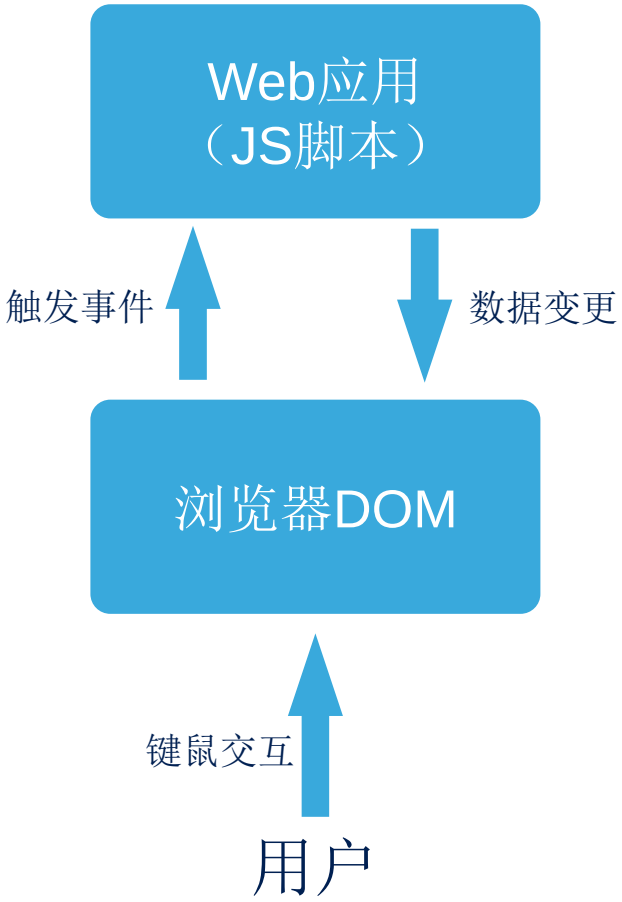
尽量减少对DOM的操作

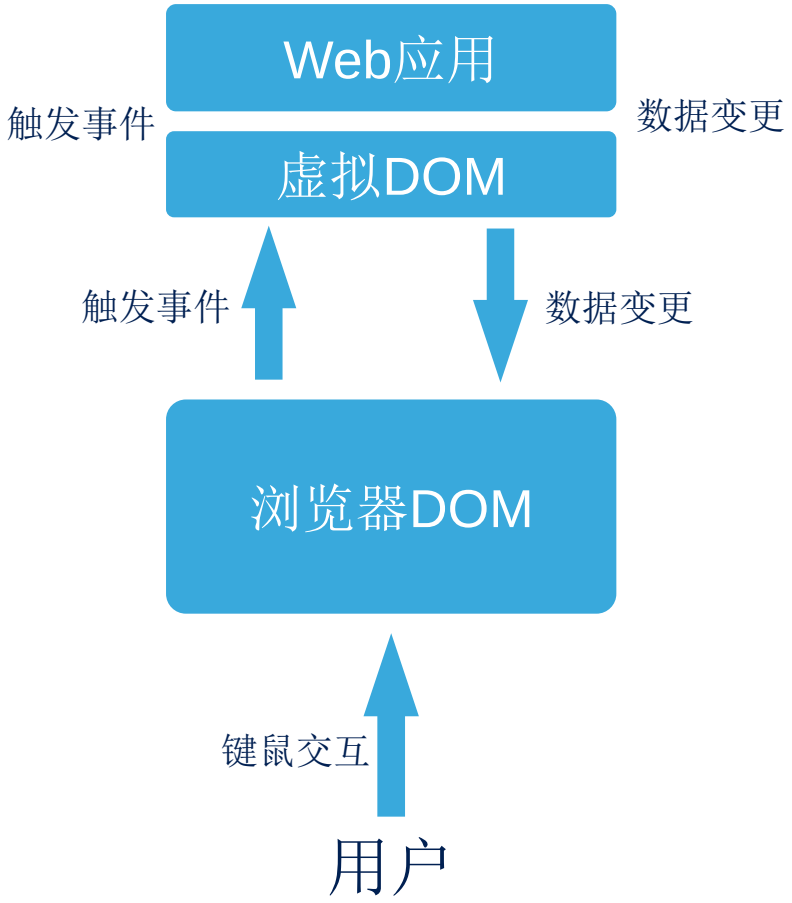
对DOM的每一次修改会引发Render Tree的重新计算，占用大量的浏览器资源，容易造成页面卡顿崩溃的情况，降低了前端应用的体验

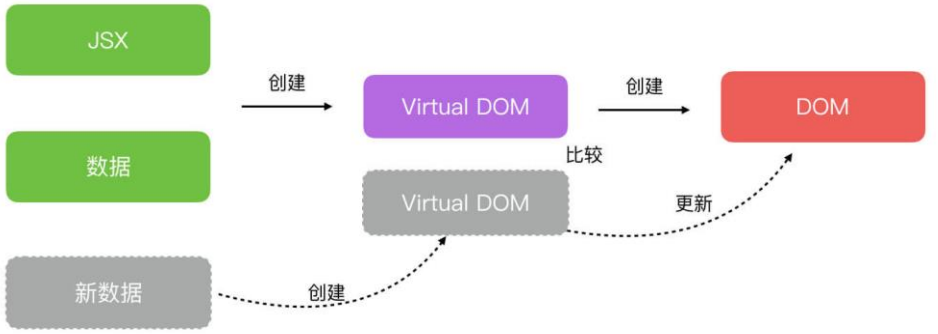
前端请求到新数据
该如何渲染？

虚拟DOM，差分带来最小干扰

React技术方案—虚拟DOM







虚拟DOM的基本原理

通过对DOM的最小干扰提升页面加载运行效率，提升使用体验。

JSX（JavaScript + XML）

JSX

react定义的一种类似于XML的JS扩展语法：XML+JS。 js中直接可以套标签，但标签要套js需要放在{}中，JSX 用来创建react虚拟DOM对象，在{}括号中的元素会自动当做JavaScript执行，而外部的XML（HTML内容）可以给Virtual DOM执行。JSX语言的程序是不能被浏览器识别并直接运行的，需要转化成为浏览器可以识别的HTML + JavaScript + CSS内容，React提供Babel.js来进行处理，而开发者无需考虑和感知。

```
const element = <h1>Hello, world!</h1>;
```

特点：

- JSX 执行更快，它在编译为 JavaScript 代码后进行了优化。
- 它是类型安全的，在编译过程中就能发现错误。
- 使用 JSX 编写模板更加简单快速。

React的核心思想—组件

Component（组件）

组件是React的核心思想。

React是面向组件编程的(组件化编码开发)，一切都是基于组件的。可以通过定义一个组件，然后在其他的组件中，可以像HTML标签一样引用它。

可以通过<ComponentName></ComponentName>的方式进行组合

```
import React, { Component } from 'react';
import { render } from 'react-dom';

class HelloMessage extends Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}

// 加载组件到 DOM 元素 mountNode
render(<HelloMessage name="John" />, mountNode);
```

通过组件的props传参

输出：Hello John

React前端项目，是大量组件组合的产物

将庞大的前端项目功能分散成若干个小的组件进行组装，增强各组件的可移植性和复用开发效率，可一定程度上理解成“搭积木”。

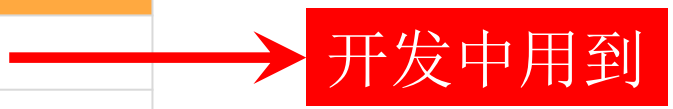
React的核心思想—组件

Component（组件）

React的组件都要继承React提供的Component类，并可以实现组件的生命周期函数，以供在页面渲染时调用，React也为组件提供丰富的API接口供调用。

React将组件看做状态机，当组件的状态发生变化时会引起组件的重新渲染。在组件中有**内部状态（state）和外部状态（属性，props）**。均**可通过this.state和this.props访问**，在render()函数中绑定state或props数据后，待**数据发生变化对引起render函数的重新执行，组件执行重新渲染。**

组件API	
设置状态	setState
替换状态	replaceState
替换属性	replaceProps
强制更新	forceUpdate
获取DOM节点	findDOMNode
判断组件挂载状态：	isMounted



React的核心思想—组件

开发中用到

Component（组件）生命周期函数

主要用到的组件生命周期函数	
componentWillMount	在渲染前调用，常用在预先请求数据或执行复杂渲染。
componentDidMount	在第一次渲染后调用，之后组件已经生成了对应的DOM结构，可以通过this.getDOMNode()来进行访问。
componentWillReceiveProps	在组件接收新的 prop (更新后)时被调用。
shouldComponentUpdate	返回一个布尔值，在组件接收到新的props或者state时被调用，返回true则组件执行渲染，false为不渲染。在初始化时或者使用forceUpdate时不被调用。
componentWillUpdate	在组件接收到新的props或者state但还没有render时被调用。在初始化时不会被调用。
componentDidUpdate	在组件完成更新后立即调用。在初始化时不会被调用。
componentWillUnmount	在组件从 DOM 中移除之前立刻被调用。

```
import React, { Component } from 'react';
import { render } from 'react-dom';

class Hello extends React.Component {

  constructor(props) {
      super(props);
      this.state = {opacity: 1.0};
  }
```

组件在浏览器加载完执行

```
  componentDidMount() {
    this.timer = setInterval(function () {
      var opacity = this.state.opacity;
      opacity -= .05;
      if (opacity < 0.1) {
        opacity = 1.0;
      }
      this.setState({
        opacity: opacity
      });
    }.bind(this), 100);
  }
```

state调整亮度

```
  render () {
    return (
      <div style={{opacity: this.state.opacity}}>
        Hello {this.props.name}
      </div>
    );
  }
}
```

触发重新渲染

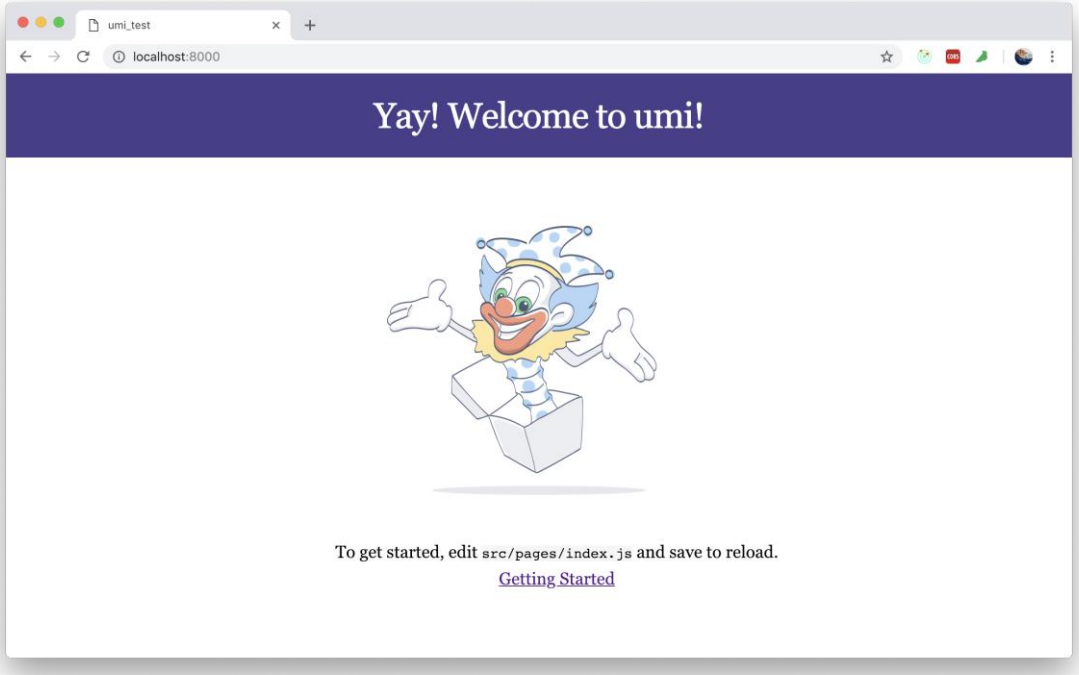
life.augmented 阿里云

化繁为简的umi框架

umi

中文可发音为乌米，是一个可插拔的企业级 react 应用框架。umi 以路由为基础的，支持类 next.js 的约定式路由（根据文件自动生成路由），比如支持路由级的按需加载，可以帮助开发者快速搭建应用而无需关心复杂的路由配置，umi 是蚂蚁金服的底层前端框架，官方文档地址：https://umijs.org/zh/

Umi.js



化繁为简的umi框架

Umi安装过程：

- 安装Node.js环境：Node.js下载地址：https://nodejs.org/zh-cn/
- 安装yarn：命令行执行 npm i yarn tyarn -g
- 安装umi：命令行执行 yarn global add umi

丰富高质组件库Ant Design

Ant Design

Ant Design 是专门用来开发和服务于企业级后台产品的UI组件库，简单来说就是提供了大量的UI组件，例如输入框、动效、按钮、表格等等，每个UI组件都有稳定的API可进行调用，开发者只要在各组件API基础上进行开发，即可快速搭建自己的前端页面。官网：https://ant.design/index-cn

life.augmented

阿里云

丰富高质组件库Ant Design

代码演示

Primary Default Dashed Danger

按钮类型

按钮有四种类型：主按钮、次按钮、虚线按钮、危险按钮。

主按钮在同一个操作区域最多出现一次。

```
import { Button } from 'antd';

ReactDOM.render(
  <div>
    <Button type="primary">Primary</Button>
    <Button>Default</Button>
    <Button type="dashed">Dashed</Button>
    <Button type="danger">Danger</Button>
  </div>,
  mountNode
);
```

API

通过设置 Button 的属性来产生不同的按钮样式，推荐顺序为：type -> shape -> size -> loading -> disabled。

按钮的属性说明如下：

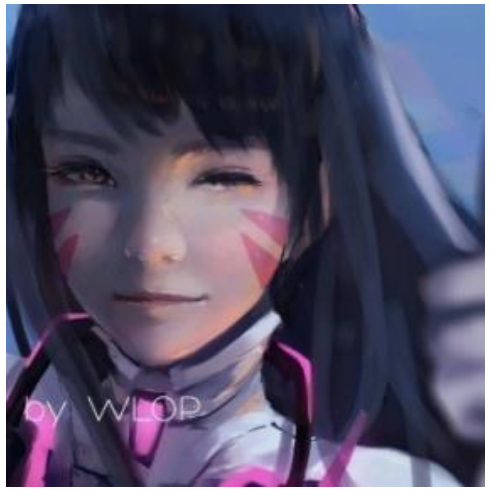
属性	说明	类型	默认值
disabled	按钮失效状态	boolean	false
ghost	幽灵属性，使按钮背景透明，版本 2.7 中增加	boolean	false
href	点击跳转的地址，指定此属性 button 的行为和 a 链接一致	string	-

对于每个组件，Ant Design官网都给出了详细的API列表和参考示例，开发者只需查阅文档即可快速上手

Umi在初始化时选择antd插件即可集成antd组件库，也可通过命令行执行npm install -g antd进行安装

数据流方案：dva

React框架将各个模块组件化，存在不足在于**React只支持单向的数据流**，即父组件可以将数据通过子组件的props属性传入，但是父组件确很难获取到子组件的状态。

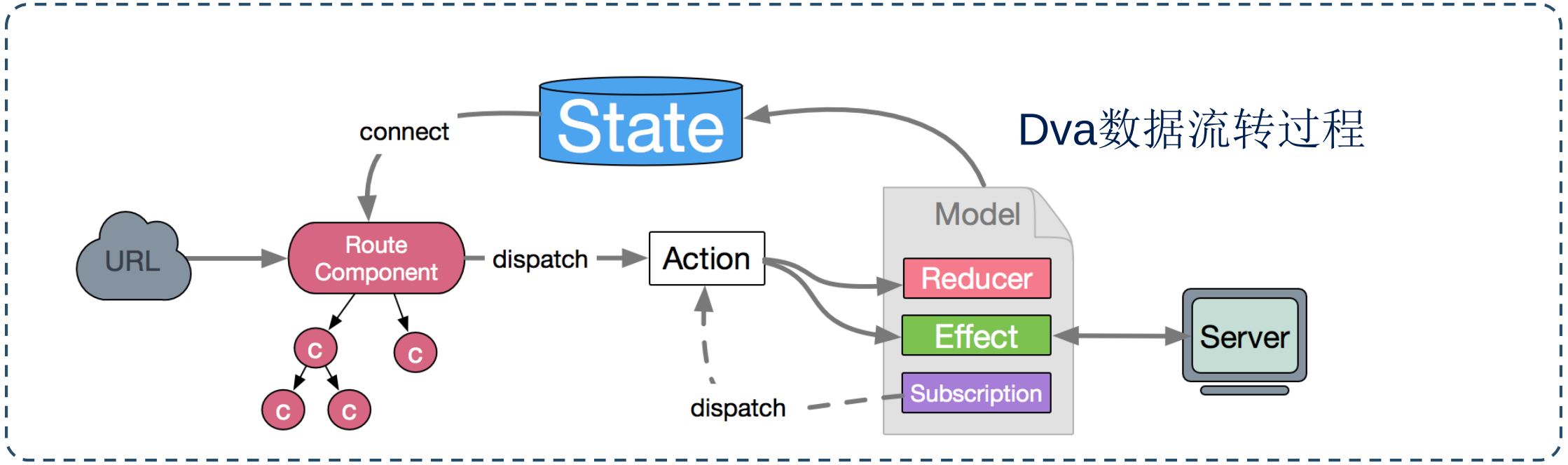


dva.js

Dva，基于Redux的方案

Redux这样的数据流框架，核心思想是**创建一个数据状态管理库（store）**，所有组件的状态state都以一个对象树的形式储存在一个单一的store中，**store中的数据通过组件的props传入，当props关联的数据发生改变时将会触发组件的重新渲染**。dva是对Redux的进一步封装，对许多配置进行了简化。dva官网地址：https://dvajs.com/

数据流方案：dva



Umi在初始化时选择dva插件即可集成，也可通过命令行执行

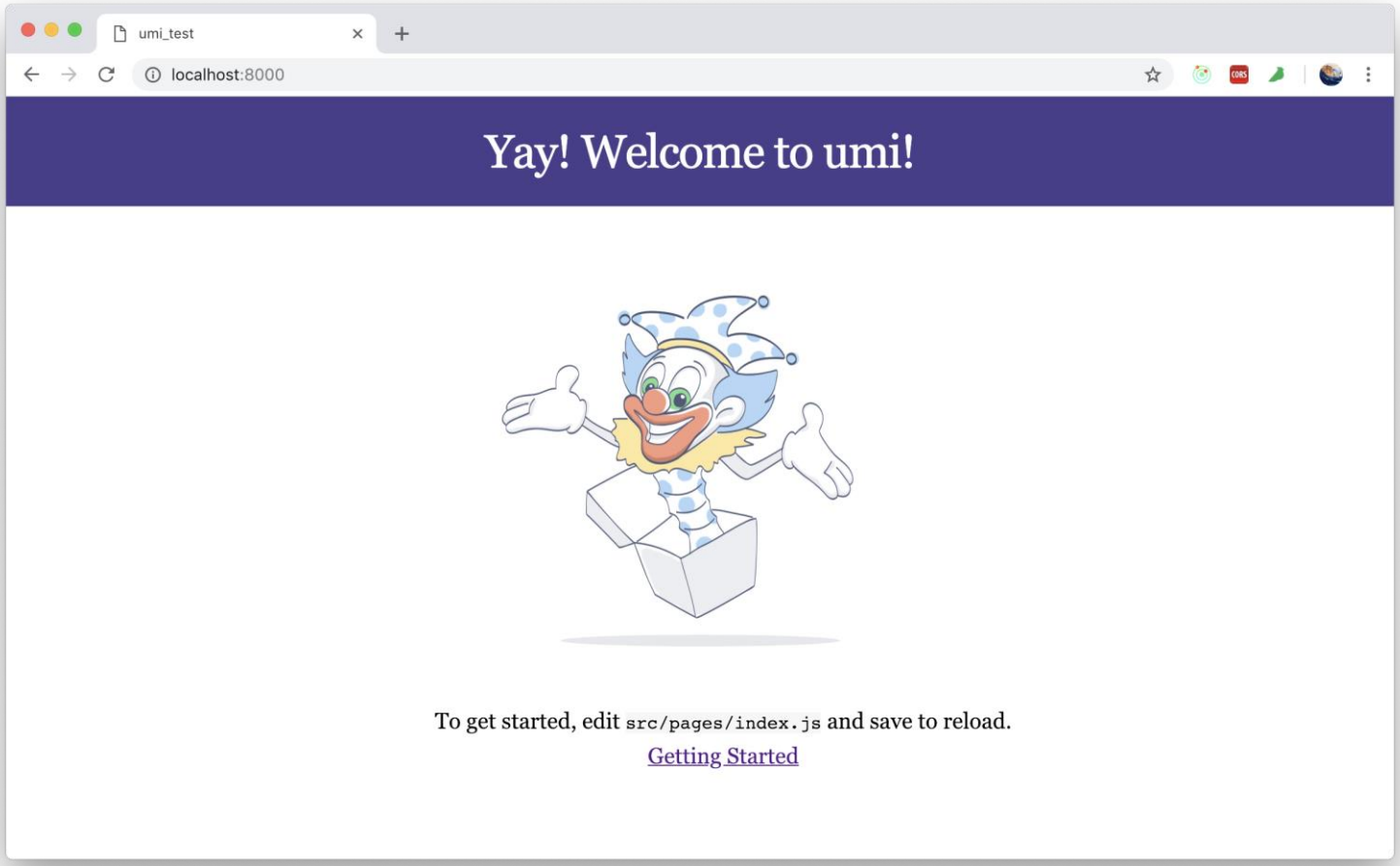
```
npm install -g dva
```

进行安装

第四章.第三节 内容简介

- 前端基础概念和知识
- 认识前端框架
  - 了解React
  - 学习Umi.js
  - 学习Ant Design
  - 了解dva.js
- 初始化并运行第一个前端项目
  - 安装yarn包管理工具
  - 全局安装umi
  - 使用脚手架初始化
  - Umi路由页面添加
- 创建和使用组件
  - Layout组件
  - 主页面创建
  - 组件引入
  - 组件完善
- 使用dva实现数据流转
  - 数据请求
  - Dva管理数据
- 应用调试与部署
  - 项目打包
  - 项目部署

初始化前端项目


umi_test
localhost:8000
Yay! Welcome to umi!
To get started, edit src/pages/index.js and save to reload.
Getting Started

软件环境安装

在正式开始之前，需要系统已经安装以下软件环境：

Node.js

Node.js是一个高性能的JavaScript运行环境，基于强大的Chrome V8引擎打造

安装地址：
https://nodejs.org/zh-cn/

yarn

yarn是流行的包管理工具，Yarn 会缓存它下载的每个包，所以无需重复下载。它还能并行化操作以最大化资源利用率，安装速度之快前所未有。

安装地址：
https://yarnpkg.com/zh-Hans/docs/install#windows-stable
或：npm install yarn tyarn -g

umi

Umi是一款基于路由的前端开发框架，目前为蚂蚁金服底层前端框架。

安装方式：

命令行执行：
yarn global add umi

初始化前端项目

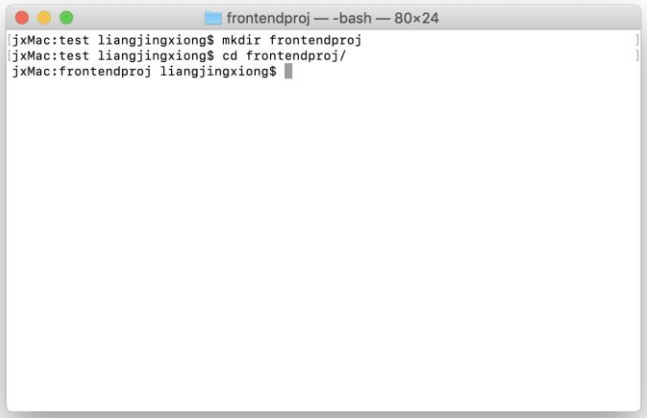
0. 新建项目文件夹

Linux&macOS

在保存项目的文件夹下采用terminal执行：

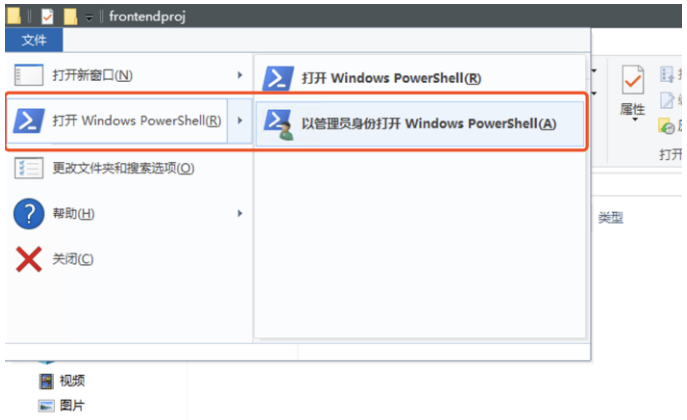
mkdir frontendproj

cd frontendproj



Windows

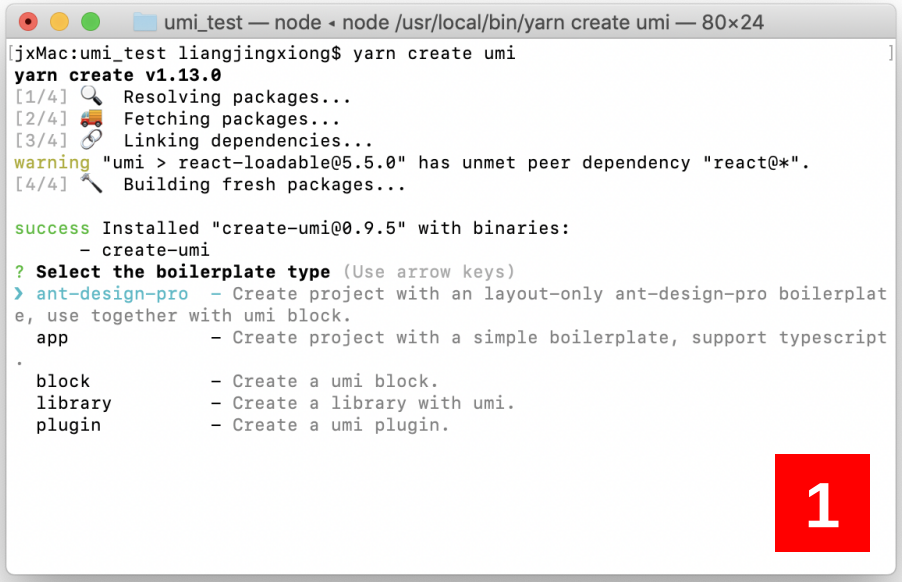
在保存项目的文件夹下右键新建文件夹frontendproj，进入文件夹，在文件通过powershell进入shell终端：

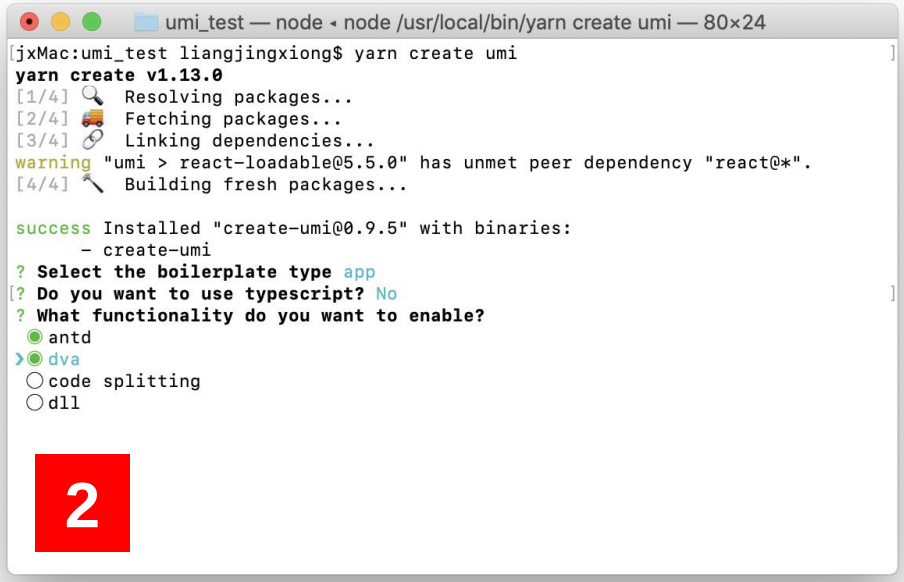


初始化前端项目

1. 在项目文件夹内命令行执行：**yarn create umi**

脚手架工具将会自动安装必备的软件包依赖，通过按钮选择**引用类型为app->不使用TypeScript（n）->选中antd和dva功能块**，选择结束后，umi将通过脚手架生成项目模版。





初始化前端项目

2. 在项目文件夹内命令行执行：yarn 安装依赖

yarn命令会依照package. json中定义的依赖包内容自动安装依赖到文件夹下node_modules子文件夹

```
.
├── mock/                          // mock 文件所在目录，基于 express
├── node_modules/                  // 依赖包安装文件夹
└── src/                           // 源码目录，可选
    ├── layouts                    // 全局布局
    ├── models                     // dva models文件存放目录，存放global的models
    ├── pages/                     // 页面目录，里面的文件即路由
        ├── .umi/                  // dev 临时目录，需添加到 .gitignore
        ├── index.js               // HTML 模板
        ├── index.css              // 404 页面
    ├── global.css                 // 约定的全局样式文件，自动引入，也可以用 global.less
    ├── app.js                   // 可以在这里加入 polyfill
├── .umirc.js                      // umi 配置，同 config/config.js，二选一
├── .env                           // 环境变量
└── package.json
```

开发中主要编辑的文件

项目初始化目录结构

初始化前端项目

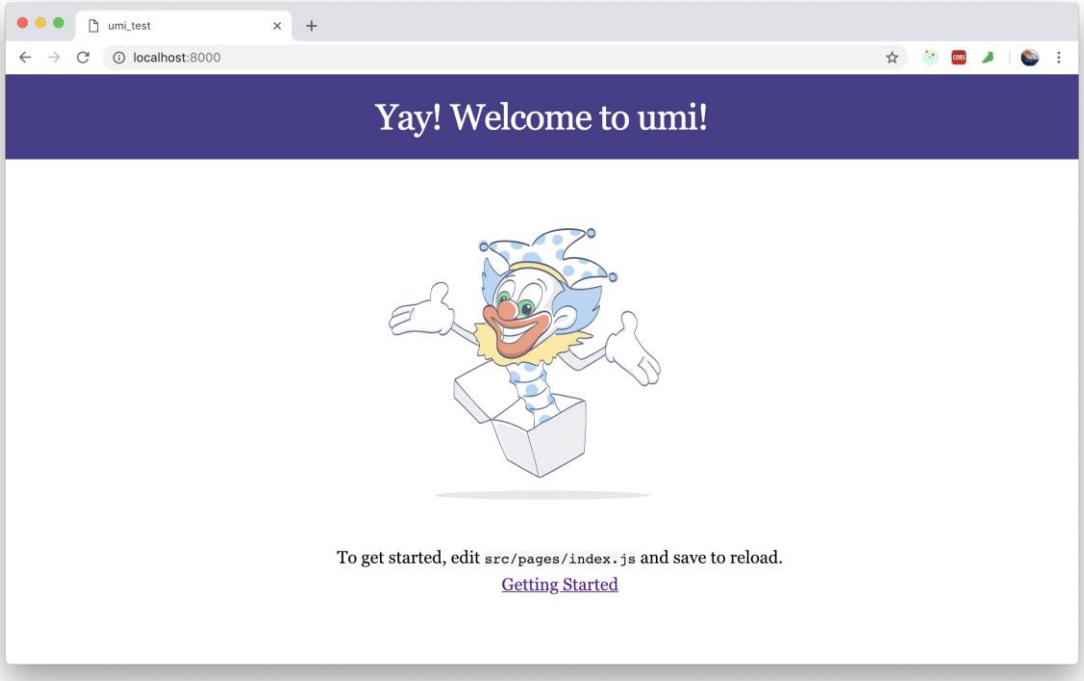
3. 在项目文件夹内命令行执行：**yarn start** 启动项目

webpack会自动编译当前项目文件，并通过本地网络地址8000端口开放访问，访问到以下页面即通过脚手架工具初始化成功。

脚手架：

可以理解为“**项目模版**”

访问localhost:8000



4. 修改组件内容和Layout

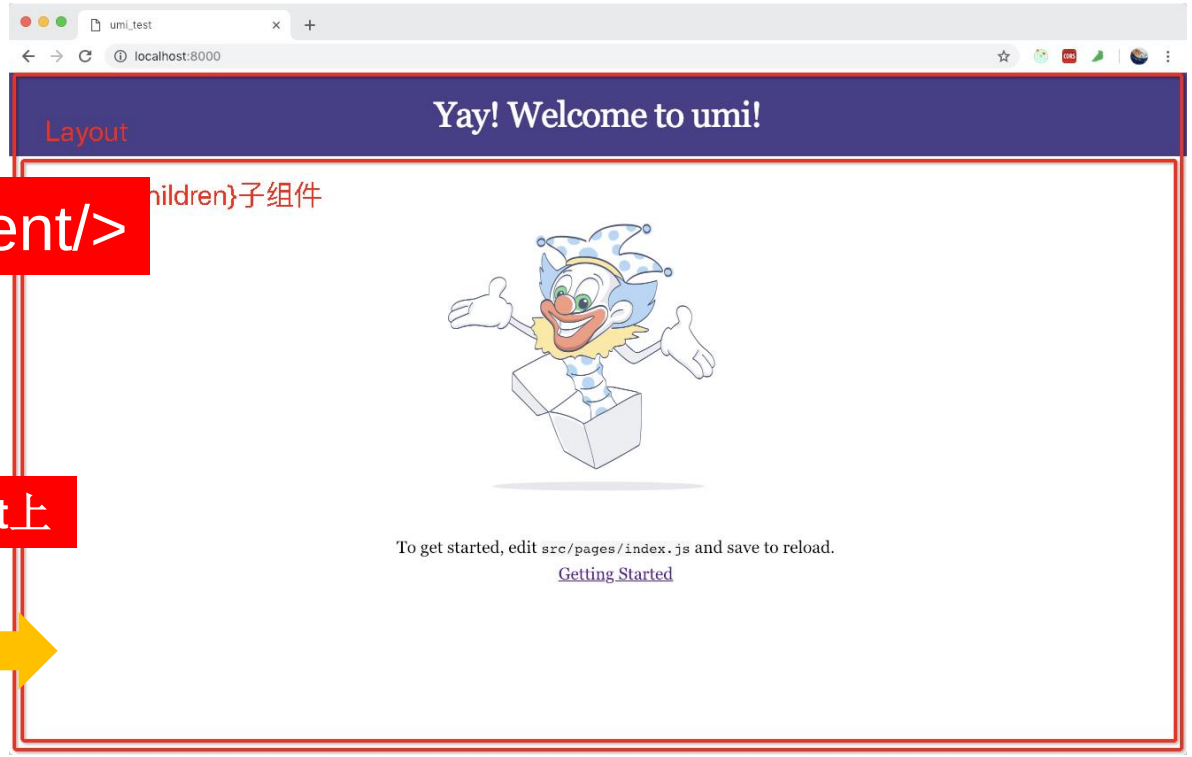
pages/index.js文件即为首页小丑页面组件，layout/index.js即为页面布局（layout）组件，分别修改其中内容观察组件内容变化。

```
<Layout>
    {this.props.children}
</Layout>
```

<Content/>

Umi将Layout组件嵌套在Content上

访问content下路由 → Layout+content

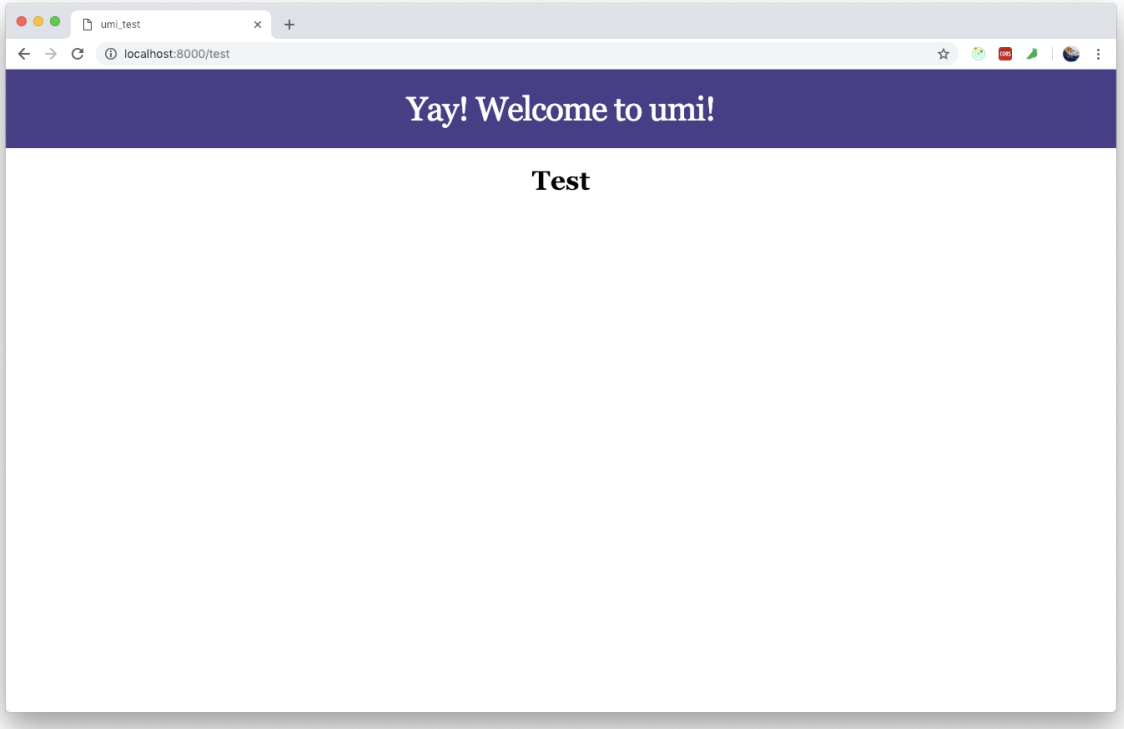


5. 添加新的页面

Umi采用约定式路由，即会**通过pages下新增的文件名或文件夹名称创建组件的路由**，在pages下新建test.js或test/index.js，写入组件内容后保存，修改浏览器路由即可访问新的组件内容

```
import React from 'react';

export default class Test extends React.Component {
    render() {
        return (
            <div>
                <h1>Test</h1>
            </div>
        );
    }
}
```

浏览器访问localhost:8000/test



第四章.第三节 内容简介

- 前端基础概念和知识
- 认识前端框架
  - 了解React
  - 学习Umi.js
  - 学习Ant Design
  - 了解dva.js
- 初始化并运行第一个前端项目
  - 安装yarn包管理工具
  - 全局安装umi
  - 使用脚手架初始化
  - Umi路由页面添加
- 创建和使用组件
  - Layout组件
  - 主页面创建
  - 组件引入
  - 组件完善
- 使用dva实现数据流转
  - 数据请求
  - Dva管理数据
- 应用调试与部署
  - 项目打包
  - 项目部署

Layout组件

使用Layout组件

Ant Design提供了丰富的layout组件以及示例代码参考：
https://ant.design/components/layout-cn/

Antd提供的Layout布局样式

对应的示例代码

基本结构

典型的页面布局。

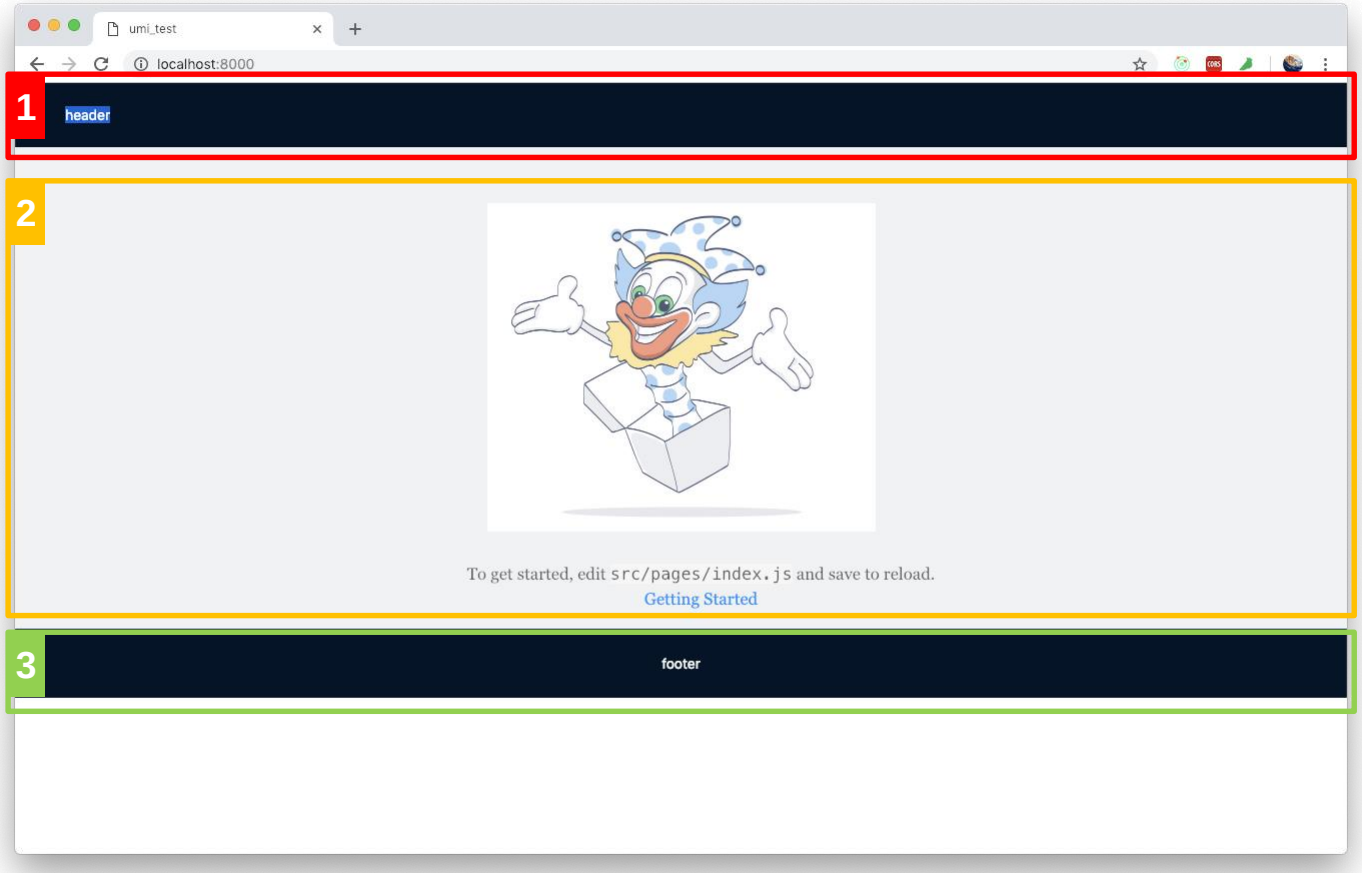
```
import { Layout } from 'antd';

const {
  Header, Footer, Sider, Content,
} = Layout;

ReactDOM.render(
  <div>
    <Layout>
      <Header>Header</Header>
      <Content>Content</Content>
      <Footer>Footer</Footer>
    </Layout>

    <Layout>
      <Header>Header</Header>
      <Layout>
        <Sider>Sider</Sider>
        <Content>Content</Content>
      </Layout>
      <Footer>Footer</Footer>
    </Layout>

    <Layout>
      <Header>Header</Header>
      <Layout>
        <Content>Content</Content>
        <Sider>Sider</Sider>
      </Layout>
```

使用Layout组件

Layout组件引入

修改项目layout/index.js组件内容，引入Antd提供的layout组件内容，保存后新的layout内容会自动替换。

```
import React from 'react';
import 'antd/dist/antd.css';
import { Layout } from 'antd';

const { Header, Footer, Content } = Layout;

class BasicLayout extends React.Component {

  render() {
    return (
      <div>
        <Layout>
          <Header>
            header
          </Header>
          <Content>
            {this.props.children}
          </Content>
          <Footer style={{ textAlign: 'center', backgroundColor: '#001529' }}>
            <div style={{ color: "white" }}>
            footer
          </div>
          </Footer>
        </Layout>
      </div>
    );
  }
}
export default BasicLayout;
```



使用Layout组件

完善组件内容

根据设计稿，我们需要在Header添加Logo内容以及设备选择下拉框，前端需要通过用户选择的deviceName参数对后端进行数据请求，下拉框使用Ant Design提供的「Select组件」，修改BasicLayout内容为下：

```
import React from 'react';
import 'antd/dist/antd.css';
import { Layout, Select } from 'antd';
const Option = Select.Option;
const { Header, Footer, Content } = Layout;

class BasicLayout extends React.Component {

  render() {
    //Select组件的选择回调函数
    const onSelect = (value) => {
      console.log("选中: " + value);
    }
    return (
      <div>
        <Layout>
          <Header style={{ display: 'flex', justifyContent: 'space-between' }}>
            <div style={{ color: "white", fontSize: '24px', fontWeight: 'bold', float: 'left' }}>阿里云IoT | ST 智能家居教程</div>
            <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', color: 'white' }}>请选择设备deviceName:
               
              <Select style={{ width: '200px', float: 'right' }}
                placeholder="当前返回无设备"       //当默认没有Option或者未选中Option(注释defaultValue)时显示
                onChange={onSelect}               //Select组件的选择回调，输出选中Option的value属性，参见Select组件API文档
                defaultValue={"默认设备[待替换]"} //默认选中的选项，应填充设备
              >
                <Option value={"device1"}>device1</Option>
                <Option value={"device2"}>device2</Option>
                <Option value={"device3"}>device3</Option>
              </Select>
            </div>
          </Header>
          <Content>
            {this.props.children}
          </Content>
          <Footer style={{ textAlign: 'center', backgroundColor: '#001529' }}>
            <div style={{ color: "white" }}>
              Powered By [illegible]abits.cn
            </div>
          </Footer>
        </Layout>
      </div>
    );
  }
}
export default BasicLayout;
```

1

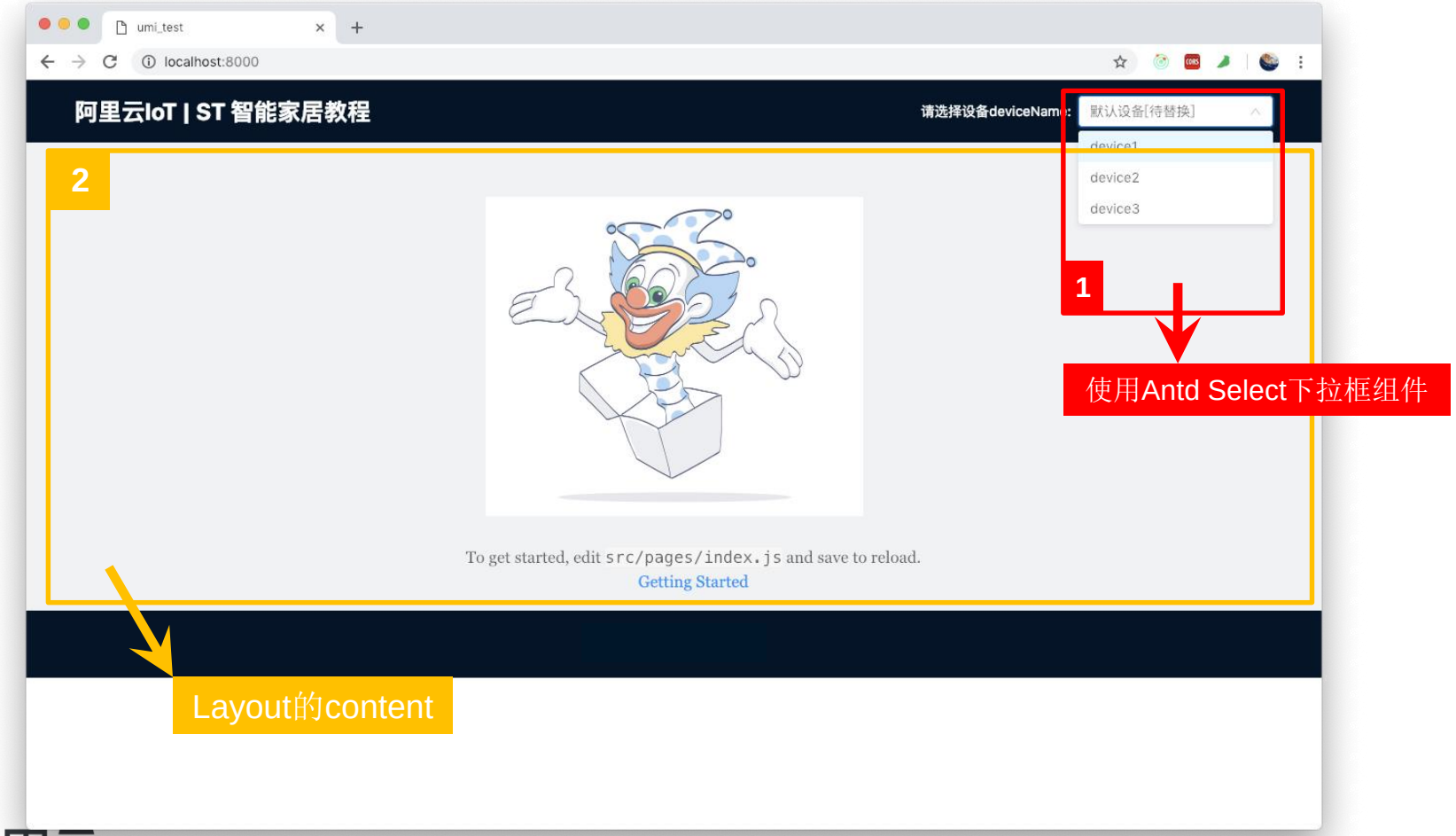
使用Antd Select下拉框组件

2

Layout注入content

完善组件内容

保存后，layou更新为以下内容：



待完善的内容

1.组件联动

在Header中加入设备选择后，BasicLayout的作用不仅仅是给页面提供样式嵌套，还需要通过**deviceName的选择动态更新Content中包裹的设备数据的相应变化**，实现多个设备数据在页面上有序的切换。

2.数据请求

在Select组件的defaultValue={“默认设备[待替换]”} 属性中，组件应提前向后端服务请求设备的列表，将列表的第一个设备deviceName填充进defaultValue属性，当Select选择新的deviceName时，需要完成新的deviceName下的设备数据请求，替换已有下拉框内容。

组件的数据部分介绍将在下一节dva数据流内容进行讲解。

页面拆解

根据设计稿，在主页面中需要展示设备当前状态和设备历史状态（历史温湿度曲线、温度报警表格）两个块，为了使得项目代码清晰易懂可维护，我们**使用React的组件化实现复杂页面的组件化拆解**，拆解出来的组件作为一个独立文件，通过组件组合出目标页面。

页面拆解

Card组件

在主页面需要确定各子组件的布局及位置，各组件位置可以通过字符进行占位。使用Ant Design 「Card组件」快速实现主页面的框架。「Card组件」提供内容块隔离的样式，可通过配置Card.Grid快速实现，免去写复杂的CSS样式。「Card组件」同时提供可切换的Tab页，通过TabList API即可快速实现Tab页的切换，同时也和Card.Grid保持风格上的一致。

页面拆解

依据拆解划分的组件内容修改项目目录如下：

```
.
├── mock/                          // mock 文件所在目录，基于 express
├── node_modules/                  // 依赖包安装文件夹
└── src/                           // 源码目录，可选
    ├── layouts                    // 全局布局
    ├── models                     // dva models文件存放目录，存放global的models
    ├── pages/                     // 页面目录，里面的文件即路由
        ├── .umi/                  // dev 临时目录，需添加到 .gitignore
        ├── main/                  // 主页组件目录
            ├── index.js           // main主页组件
            ├── component/         // 存放子组件的目录
                ├── Card/          // 存放设备实时状态的四个卡片
                    ├── AlarmStatus.js    // 报警状态
                    ├── HumiStatus.js     // 湿度状态
                    ├── TempStatus.js     // 温度状态
                    ├── OnlineStatus.js   // 在线状态
                ├── Chart.js       // 曲线图组件
                ├── Table.js       // 表格组件
        ├── index.js               // umi默认欢迎页
        ├── index.css              // 欢迎页css文件
    ├── global.css                 // 约定的全局样式文件，自动引入，也可以用 global.less
    ├── app.js                     // 可以在这里加入 polyfill
├── .umirc.js                      // umi 配置，同 config/config.js，二选一
├── .env                           // 环境变量
└── package.json
```

开发中主要编辑的文件

主页拆解UI组件

4张卡片组件

曲线图和报警表格组件

框架搭建

修改main/index.js文件内容如下：

```
import React from 'react'
import 'antd/dist/antd.css';
import { Card, Icon, Tooltip } from 'antd';

export default class Main extends React.Component {

  constructor(props) {
    super(props);
    //记录当前选中的Tab
    this.state = {
      key: 'tab1',
    }
  }

  //Tab之间切换
  onTabChange = (key, type) => {
    this.setState({ [type]: key });
  }

  render() {
    //显示设备历史状态的TabList
    const tabList = [{
      key: 'tab1',
      tab: (
        <Tooltip title="设备历史温湿度数据查询">
          设备历史状态
        </Tooltip>
      ),
    }, {
      key: 'tab2',
      tab: (
        <Tooltip title="设备历史温度报警记录查询">
          设备历史报警
        </Tooltip>
      ),
    }];
    const contentList = {
      tab1: (
        <div>
          这里是历史温湿度曲线图组件
        </div>),
      tab2: (
        <div>
          这里是温湿度历史报警组件
        </div>),
    };
    //第一Row的Style内容
    const gridStyle = {
      width: '25%',
      height: '200px',
      textAlign: 'center',
      padding: '10px'
    };
    return (
      <div style={{backgroundRepeat:'no-repeat',backgroundSize:'100% 100%'}}>
        <div style={{ padding: '30px' }}>
          <Card
            title={
              <div>
                <Tooltip title="显示设备当前的运行状态">
                  设备当前状态
                </Tooltip>
              </div>}
            extra={
              <div>
                <Icon type="reload" style={{ paddingRight: '5px' }} />
                <Tooltip style={{ fontSize: '12px' }} title="设备最新一次上报属性的时间">
                  最新一次上报时间 ： 这里待填充设备最新一次上报时间
                </Tooltip>
              </div>
            }
            style={{ width: '100%' }}
          >
            <Card.Grid style={gridStyle}>
              <div>这里是在线状态卡片</div>
            </Card.Grid>
            <Card.Grid style={gridStyle}>
              <div>这里是报警状态卡片</div>
            </Card.Grid>
            <Card.Grid style={gridStyle}>
              <div>这里是温度卡片</div>
            </Card.Grid>
            <Card.Grid style={gridStyle}>
              <div>这里是湿度卡片</div>
            </Card.Grid>
          </Card>
        </div>
        <div style={{ padding: '30px', paddingTop: 0 }}>
          <Card
            style={{ width: '100%' }}
            tabList={tabList}
            activeTabKey={this.state.key}
            onTabChange={(key) => { this.onTabChange(key, 'key'); }}
          >
            {contentList[this.state.key]}
          </Card>
        </div>
      </div>
    )
  }
}
```

曲线图和报警表格组件

四张卡片组件

1

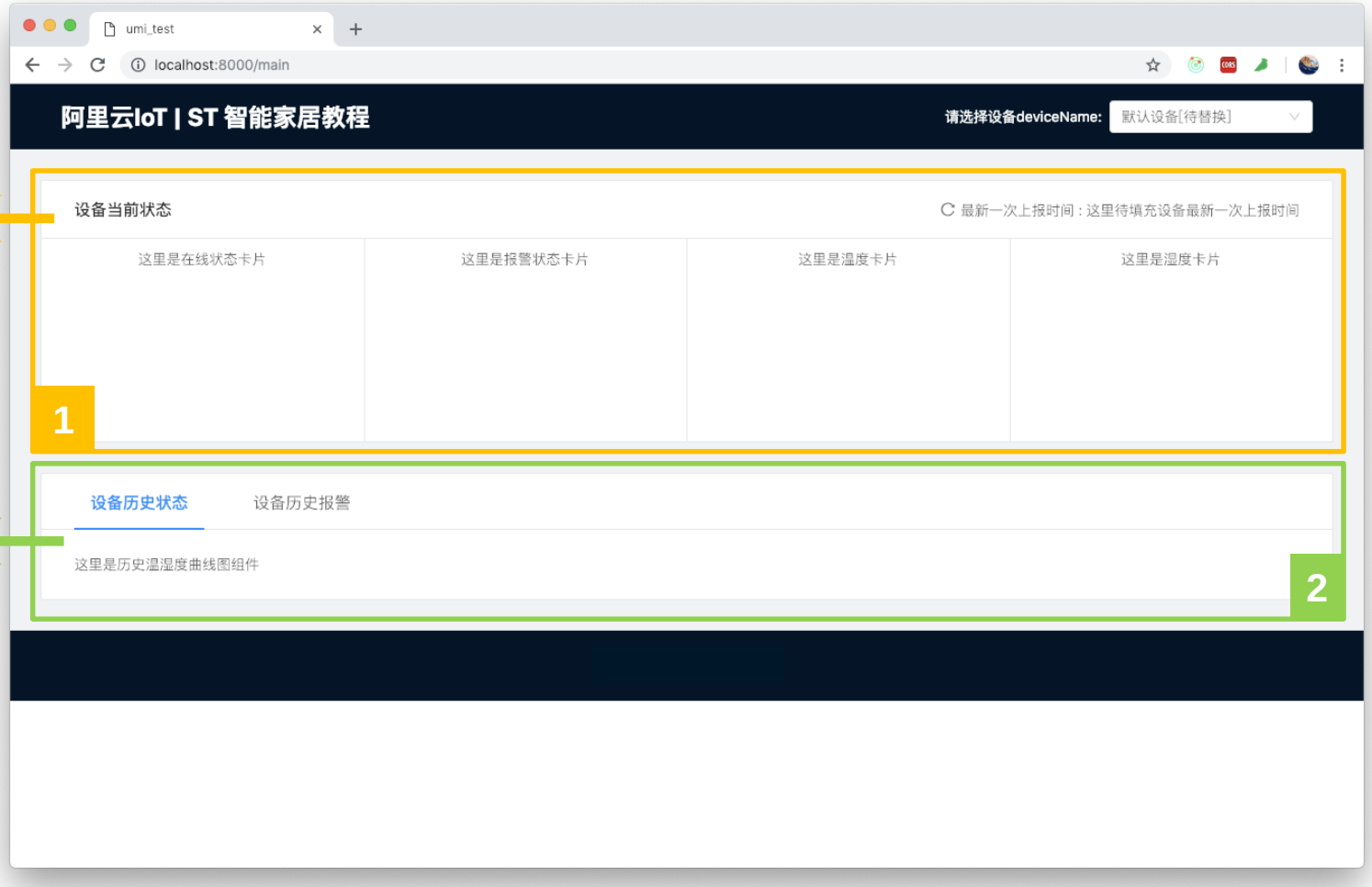
2

2

框架搭建

主页面

保存修改main/index.js后，访问浏览器localhost:8000/main，页面输出如下，至此以初步搭建好主页的页面框架。



四张卡片组件

曲线图和报警表格组件

框架搭建

在main组件中，我们使用了Ant Design提供的两个组件，「Icon图标」和「Tooltip文字提示」。

Tooltip

Tooltip组件实现简单的文字提示起泡框，通过<Tooltip>组件包裹目标组件即可，通过Tooltip的title API进行提示文字设定。

```
<Tooltip title="显示设备当前的运行状态">
    设备当前状态
</Tooltip>
```

Icon

Icon是Ant Desgin提供的语义化矢量图形组件，可以通过配置Icon的Type属性快速集成Ant Design提供的图标库中的图标。

```
<Icon type="reload" style={{ paddingRight: '5px' }} />
<Tooltip style={{ fontSize: '12px' }} title="设备最新一次上报属性的时间">
    最新一次上报时间 : 这里待填充设备最新一次上报时间
</Tooltip>
```

组件引入

我们还需要将Card组件中各个状态内容块中，将定义的main/component 文件夹下的组件引入进来，例如修改**OnlineStatusCard组件**内容为以下，并插入到**main/index.js主页面**的Card.Grid位置。同理引入其它组件。

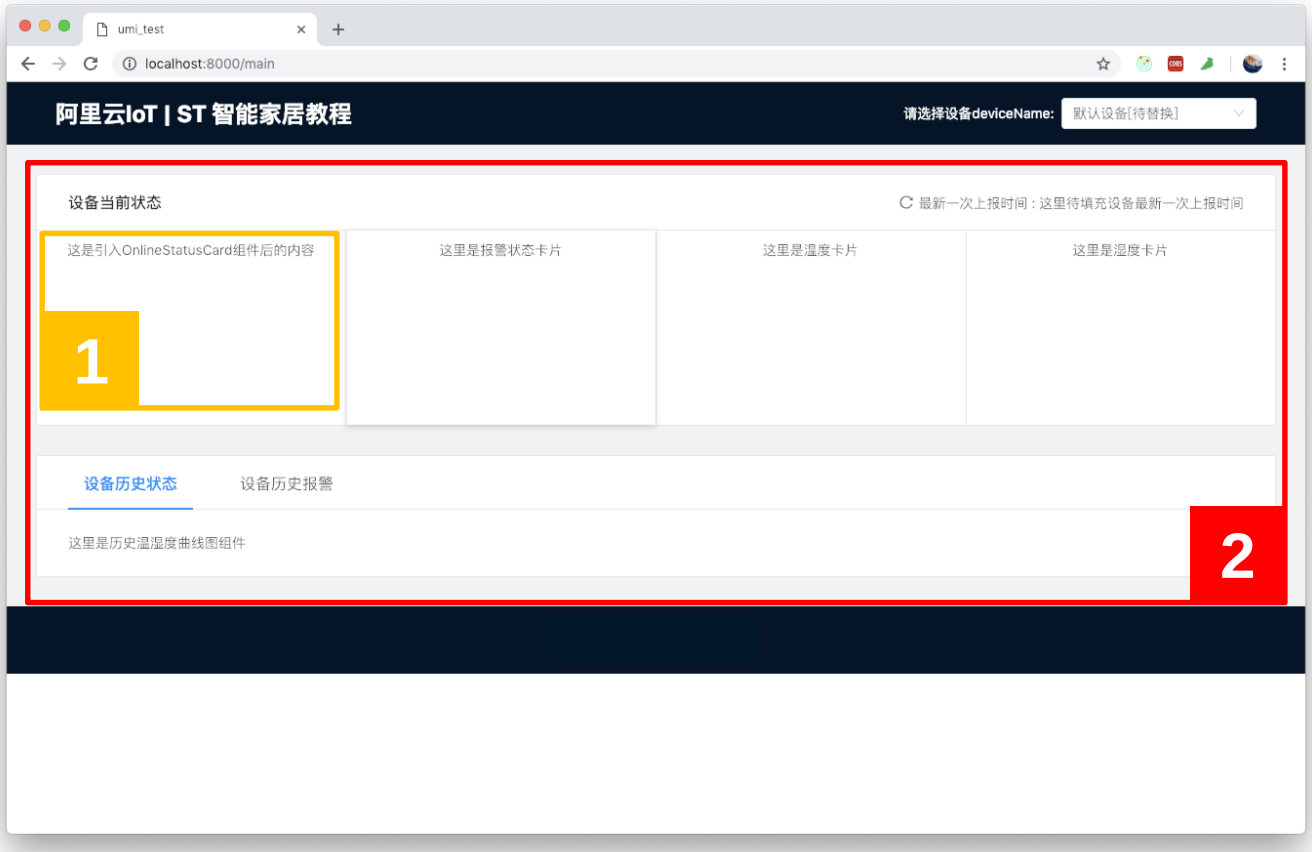
```
//component/card/OnlineStatus.js
import React from 'react'
import 'antd/dist/antd.css';

export default class OnlineStatusCard extends React.Component {
    render() {
        return (
            <div>
                这是引入OnlineStatusCard组件后的内容
            </div>
        );
    }
}
```

1

```
<Card.Grid style={gridStyle}>
  {/* 组件嵌套在需要的位置 */}
  <OnlineStatusCard/>
</Card.Grid>
<Card.Grid style={gridStyle}>
  <div>这里是报警状态卡片</div>
</Card.Grid>
<Card.Grid style={gridStyle}>
  <div>这里是温度卡片</div>
</Card.Grid>
<Card.Grid style={gridStyle}>
  <div>这里是湿度卡片</div>
</Card.Grid>
```

1

2



项目参考代码：**仓库/exercis1**，执行yarn安装依赖

内容简介

- Card卡片组件
- BizCharts曲线图组件
- Table表格组件

OnlineStatusCard

组件需要使用请求到的设备在线状态数据，显示设备不同的在线状态

卡片组件的使用

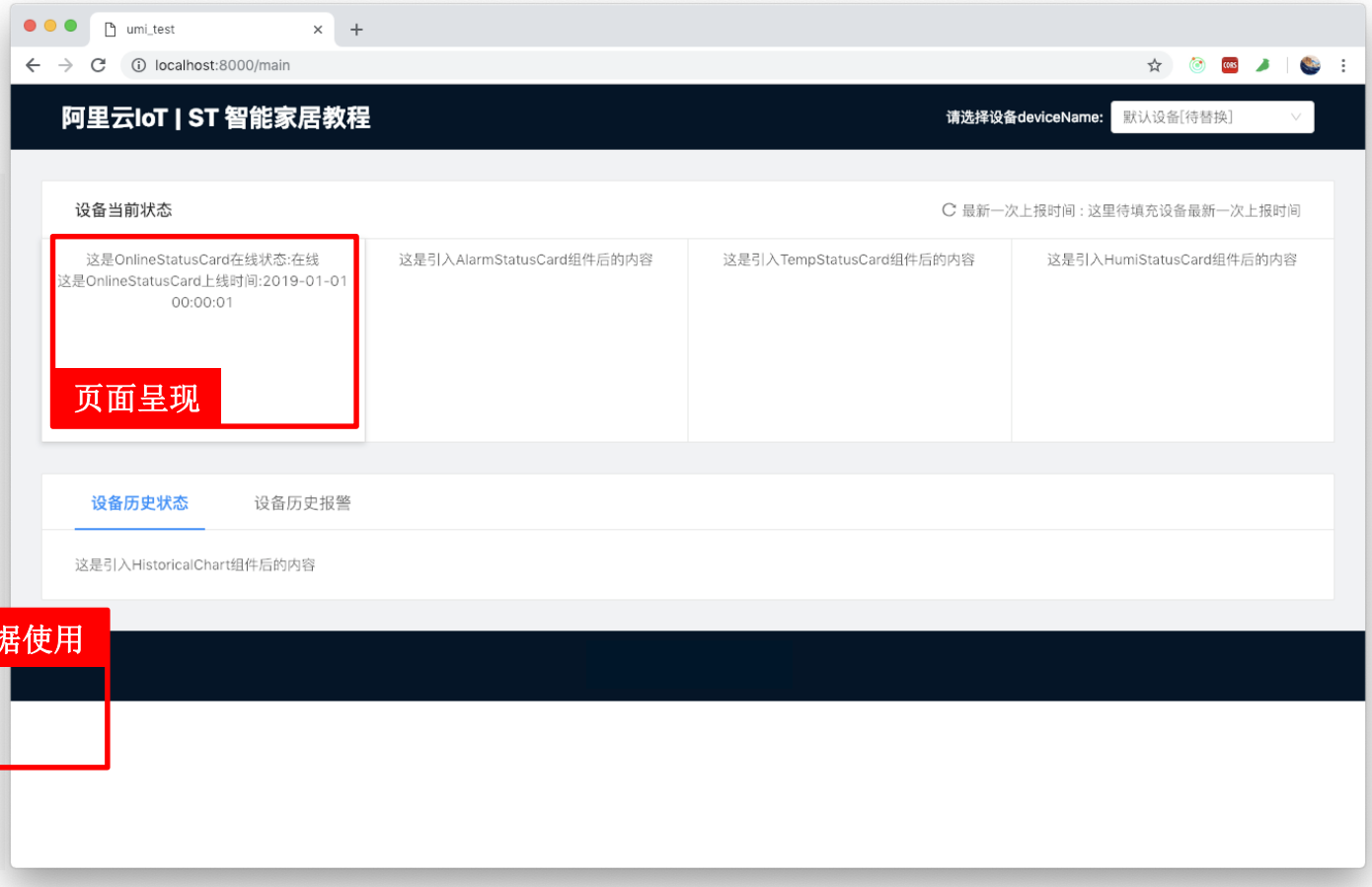
OnlineStatusCard数据

- 在线状态：state
- 最近一次上线时间：

```
import React from 'react'
import 'antd/dist/antd.css';

//组件所需数据
const onlineCard = {
  state: "在线",
  recentOnlineTime: "2019-01-01 00:00:01"
};

export default class OnlineStatusCard extends React.Component {

    render() {
        return (
            <div>
                这是OnlineStatusCard在线状态:{onlineCard.state}
                <br/>
                这是OnlineStatusCard上线时间:{onlineCard.recentOnlineTime}
            </div>
        );
    }
}
```

模拟数据内容

模拟数据使用



OnlineStatusCard样式

通过Antd提供Icon，判断在线状态，显示不同的图标。美观度主要依靠css功底

```
import React from 'react'
import 'antd/dist/antd.css';
import { Icon, Divider } from 'antd';

const onlineCard = {
    state: "离线",
    recentOnlineTime: "2019-01-01 00:00:01"
};

export default class OnlineStatusCard extends React.Component {

    render() {
        return (
            <div style={{ display: 'flex', alignItems: 'center', paddingTop: '5%', justifyContent: 'center', height: '100%', width: '100%' }}>
                <div style={{ position: 'relative', border: '1px solid #e8e8e8', height: "100%", width: "100%", borderRadius: '4px', }}>
                    <div style={{
                        position: 'absolute', top: -20, left: 40,
                        height: '100px', width: '100px',
                        backgroundColor: '#ed7010', borderRadius: '4px', boxShadow: '0 0 10px grey',
                        display: 'flex', alignItems: 'center', justifyContent: 'center',
                    }}
                    >
                        <Icon
                            type={onlineCard.state === "在线" ? "link" : "disconnect"}
                            style={{ color: 'white', fontSize: '60px' }}
                        />
                    </div>
                    <div style={{ textAlign: 'right', paddingRight: '50px' }}>
                        <div style={{ fontSize: '16px', marginTop: '0px' }}>设备在线状态</div>
                         
                         
                        <div style={{ fontSize: '30px', fontWeight: 'bold' }}>{onlineCard.state}</div>
                    </div>
                     
                     
                    <Divider style={{ marginTop: 0, marginBottom: 0, fontSize: '12px', padding: '10px 5px 0 5px' }}>最近上线时间</Divider>
                    <div style={{ fontSize: '12px' }}> {onlineCard.recentOnlineTime ? onlineCard.recentOnlineTime : "暂未上线"} </div>
                </div>
            </div>
        );
    }
}
```

模拟数据内容

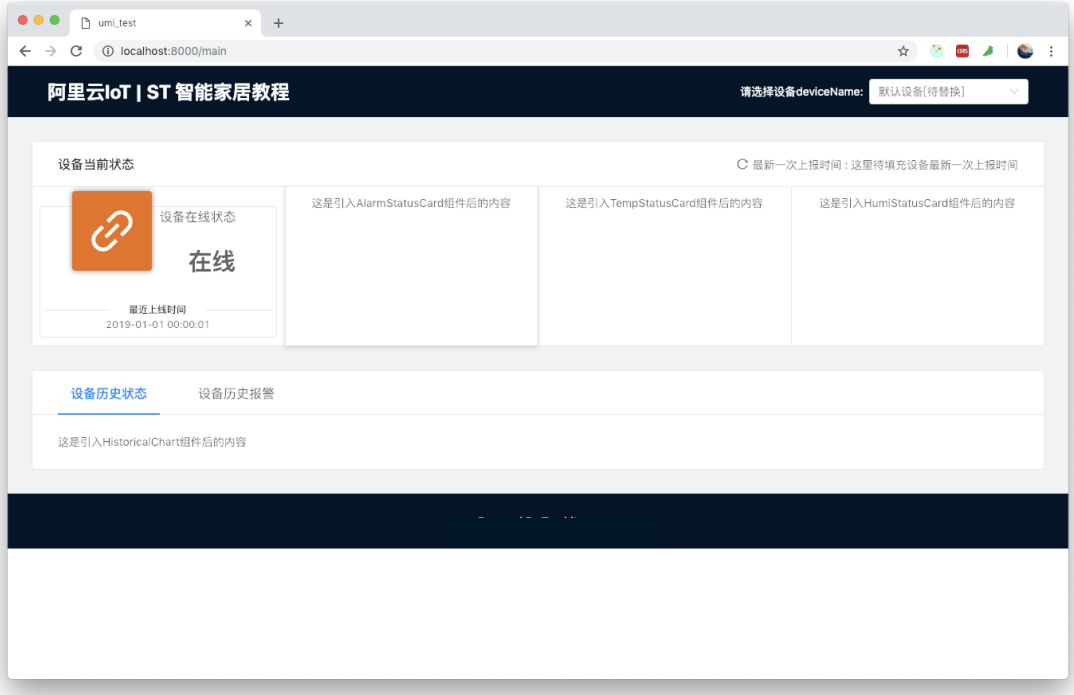
判断在线状态分别渲染内容

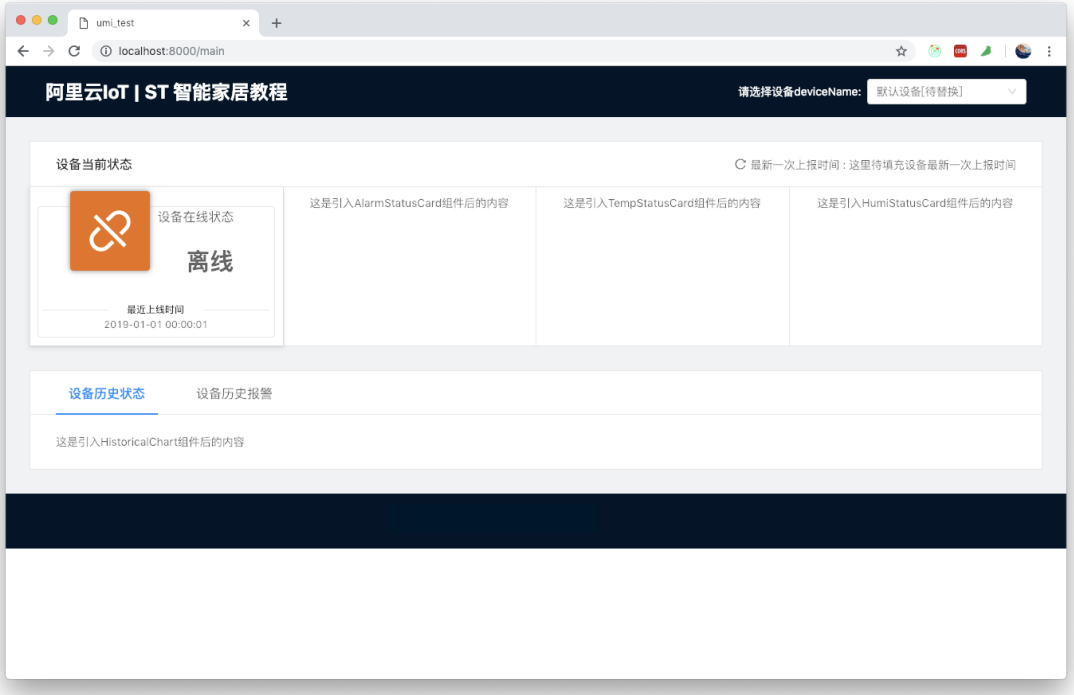
判断在线状态分别渲染内容

life.augmented 阿里云

OnlineStatusCard样式

通过Antd提供Icon，判断在线状态，显示不同的图标。美观度主要依靠css功底

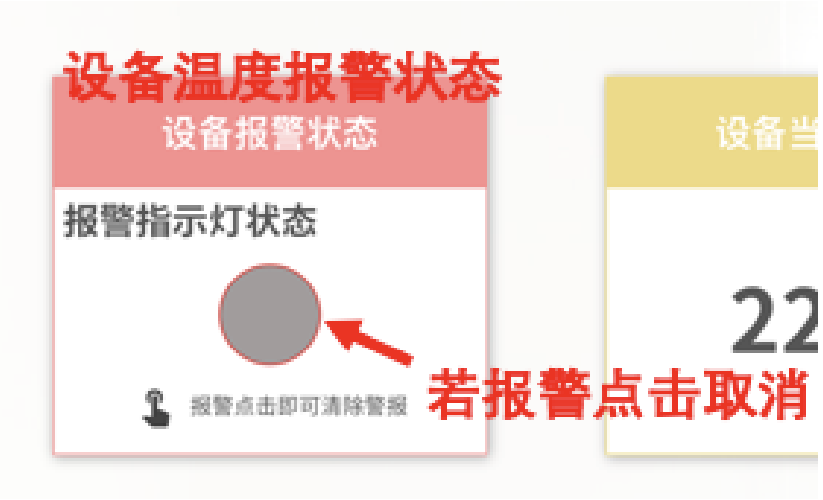




判断在线状态分别渲染不同内容

AlarmStatusCard

报警组件需要显示设备的温度报警状态，以及最近报警时间，在报警后，需要点击图标下发报警解除（向后端进行请求）



OnlineStatusCard数据

- 报警状态：state
- 最近报警时间：recentAlarmTime

```
const alarmCard = {
    state:false,//false为未报警，true为报警
    recentAlarmTime:'2019-01-01 00:00:01'
}
```

模拟数据内容

AlarmStatusCard样式

通过Antd提供Icon，判断报警状态，显示不同内容，点击报警旋转图标后进行网络请求消除报警。美观度主要依靠css功底

卡片组件的使用


```
import React from 'react'
import 'antd/dist/antd.css';
import { Icon, Divider, Tooltip } from 'antd';

const alarmCard = {
    state:false,
    recentAlarmTime:'2019-01-01 00:00:01'
}

export default class AlarmStatusCard extends React.Component {
    render() {
        //清除警报点击事件回调
        const clearAlarm = () => console.log("点击了清除警报");
        return (
            <div style={{ display: 'flex', alignItems: 'center', paddingTop: '5%', justifyCont
                <div style={{
                    position: 'relative',
                    border: '1px solid #e8e8e8',
                    borderRadius: '4px',
                    height: "100%", width: "100%",
                }}>
                    <div style={{
                        position: 'absolute', top: -20, left: 40,
                        height: '100px', width: '100px',
                        backgroundColor: '#e25858',
                        borderRadius: '4px',
                        boxShadow: '0 0 10px grey',
                        display: 'flex', alignItems: 'center', justifyContent: 'center',
                    }}
                    >
                        {
                            alarmCard.state === true ? (
                                //发生报警
                                <Tooltip title="点击按钮即可清除警报">
                                    <Icon
                                        onClick={clearAlarm}      //点击后的回调函数
                                        spin={true}               //转动
                                        type={"exclamation-circle"}
                                        style={{ color: 'white', fontSize: '60px' }}
                                    />
                                </Tooltip>
                            )
                                : (
                                    <Icon
                                        type={"check-circle"}
                                        style={{ color: 'white', fontSize: '60px' }}
                                    />
                                )
                        }
                    </div>
                    <div style={{ textAlign: 'right', paddingRight: '50px' }}>
                        <div style={{ fontSize: '16px', marginTop: '0px' }}>设备报警状态</div>
                         
                         
                        <div style={{ fontSize: '30px', fontWeight: 'bold' }}>{alarmCard.state === true ? "报警" : "正常"}</div>
                    </div>
                     
                     
                    <Divider
                        style={{
                            marginTop: 0,
                            marginBottom: 0,
                            fontSize: '12px',
                            padding: '10px 5px 0 5px'
                        }}
                    >最近报警时间</Divider>
                    <div style={{ fontSize: '12px' }}>{alarmCard.recentAlarmTime ? alarmCard.recentAlarmTime : "暂未报警"}</div>
                </div>
            </div>
        );
    }
}
```

模拟数据内容

报警

正常

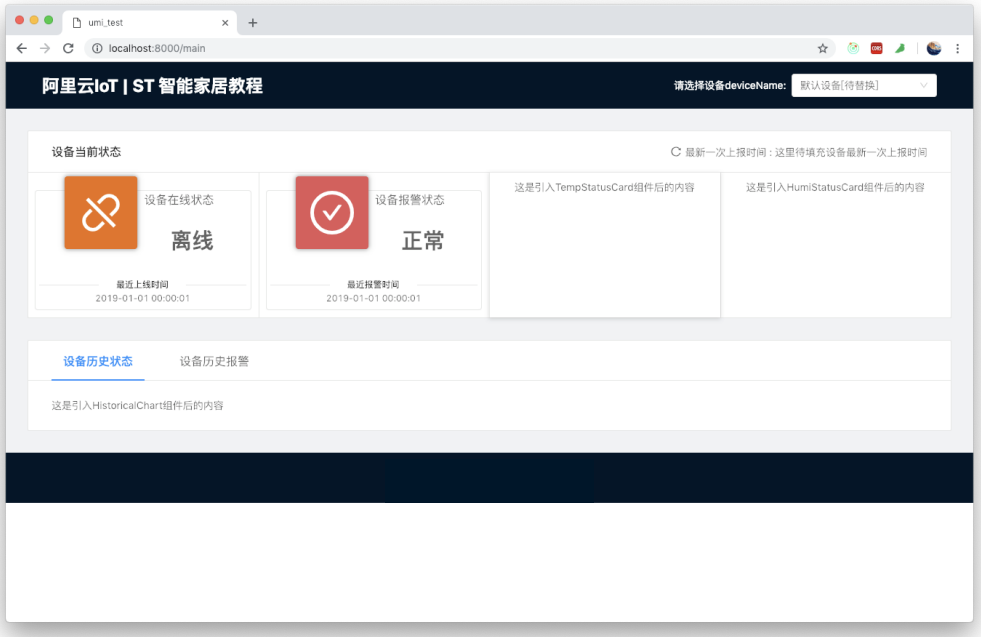
判断报警状态
分别渲染内容

图标点击
清除报警

AlarmStatusCard样式

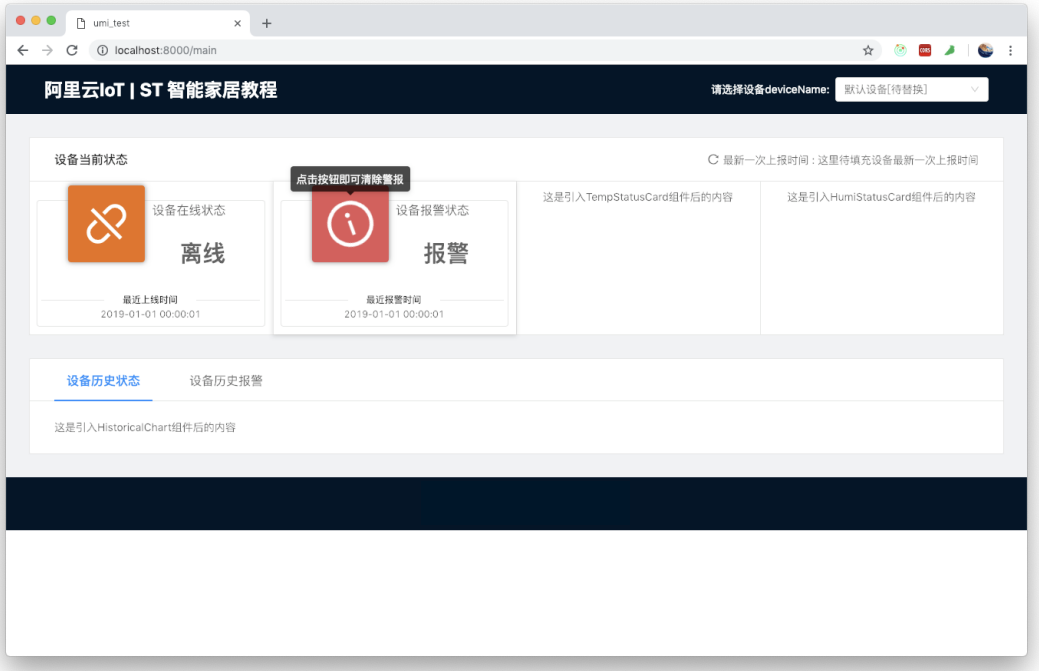
通过Antd提供Icon，判断报警状态，显示不同内容，点击报警旋转图标后进行网络请求消除报警。美观度主要依靠css功底

卡片组件的使用




```
<Icon
    type={"check-circle"}
    style={{ color: 'white', fontSize: '60px' }}
/>
```

正常




```
<Tooltip title="点击按钮即可清除警报">
  <Icon
    onClick={clearAlarm}     //点击后的回调函数
    spin={true}              //转动
    type={"exclamation-circle"}
    style={{ color: 'white', fontSize: '60px' }}
  />
</Tooltip>
```

报警

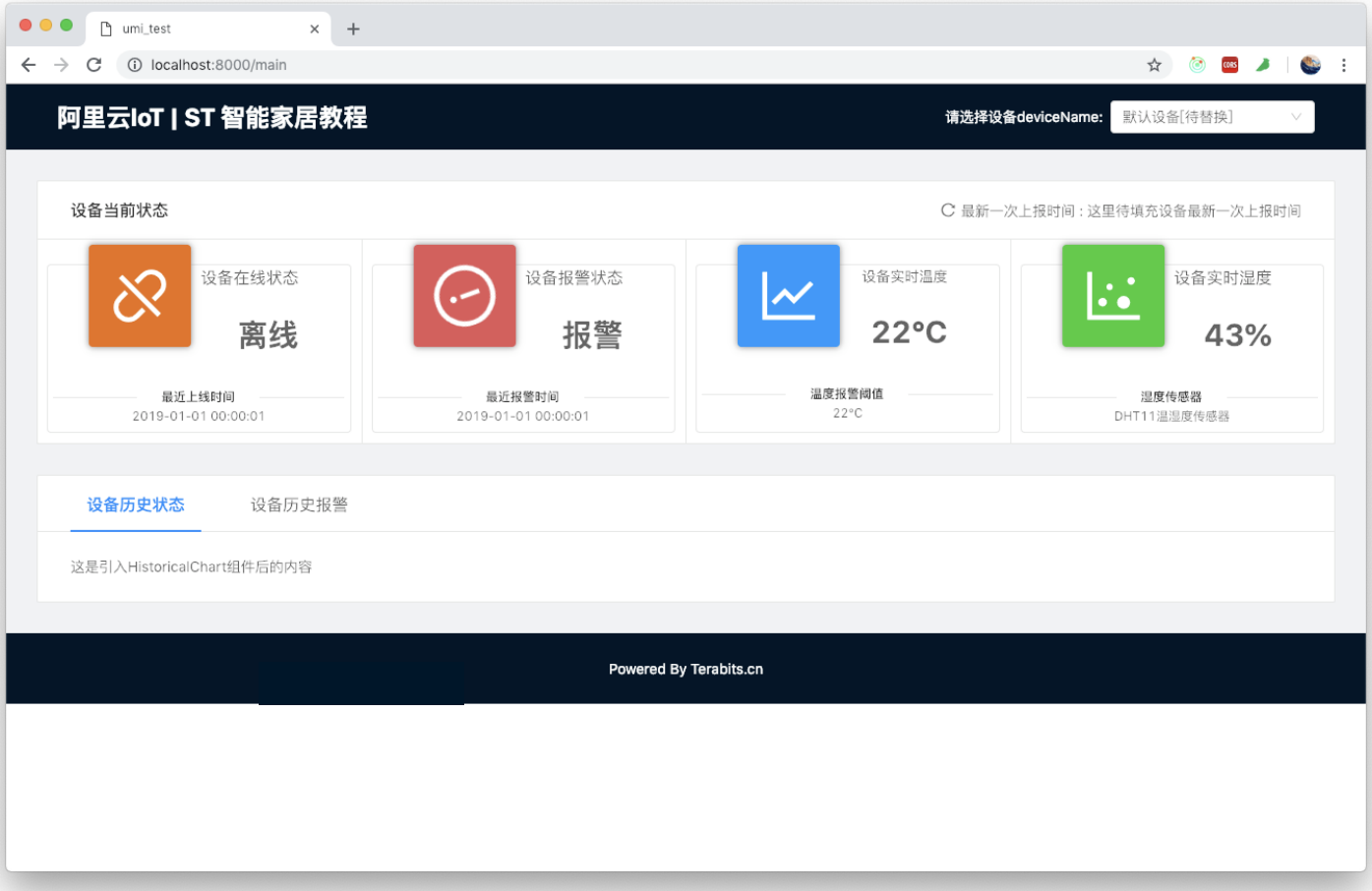
life.augmented

阿里云

温湿度卡片

温湿度卡片较为简单，请参考 课程提供的源码


```
├── README.md
├── mock
├── package.json
├── src
│   ├── assets
│   │   └── yay.jpg
│   ├── axios
│   │   ├── index.js
│   │   └── tools.js
│   ├── global.css
│   ├── layouts
│   │   ├── index.css
│   │   └── index.js
│   └── pages
│       ├── index.css
│       ├── index.js
│       └── main
│           ├── components
│           │   ├── Card
│           │   │   ├── AlarmStatus.js
│           │   │   ├── HumiStatus.js
│           │   │   ├── OnlineStatus.js
│           │   │   └── TempStatus.js
│           │   ├── Chart
│           │   │   └── index.js
│           │   └── Table
│           │       └── index.js
│           ├── index.js
│           └── models
│               └── main.js
├── tslint.yml
├── webpack.config.js
├── yarn-error.log
└── yarn.lock
```



life.augmented 阿里云

曲线图组件

HistoricalChart

HistoricalChart实现历史温湿度数据的展示，通过起止时间请求特定时间段的数据，此外还需要在表格绘制温度报警界限，同时通过输入框提交温度报警阈值。

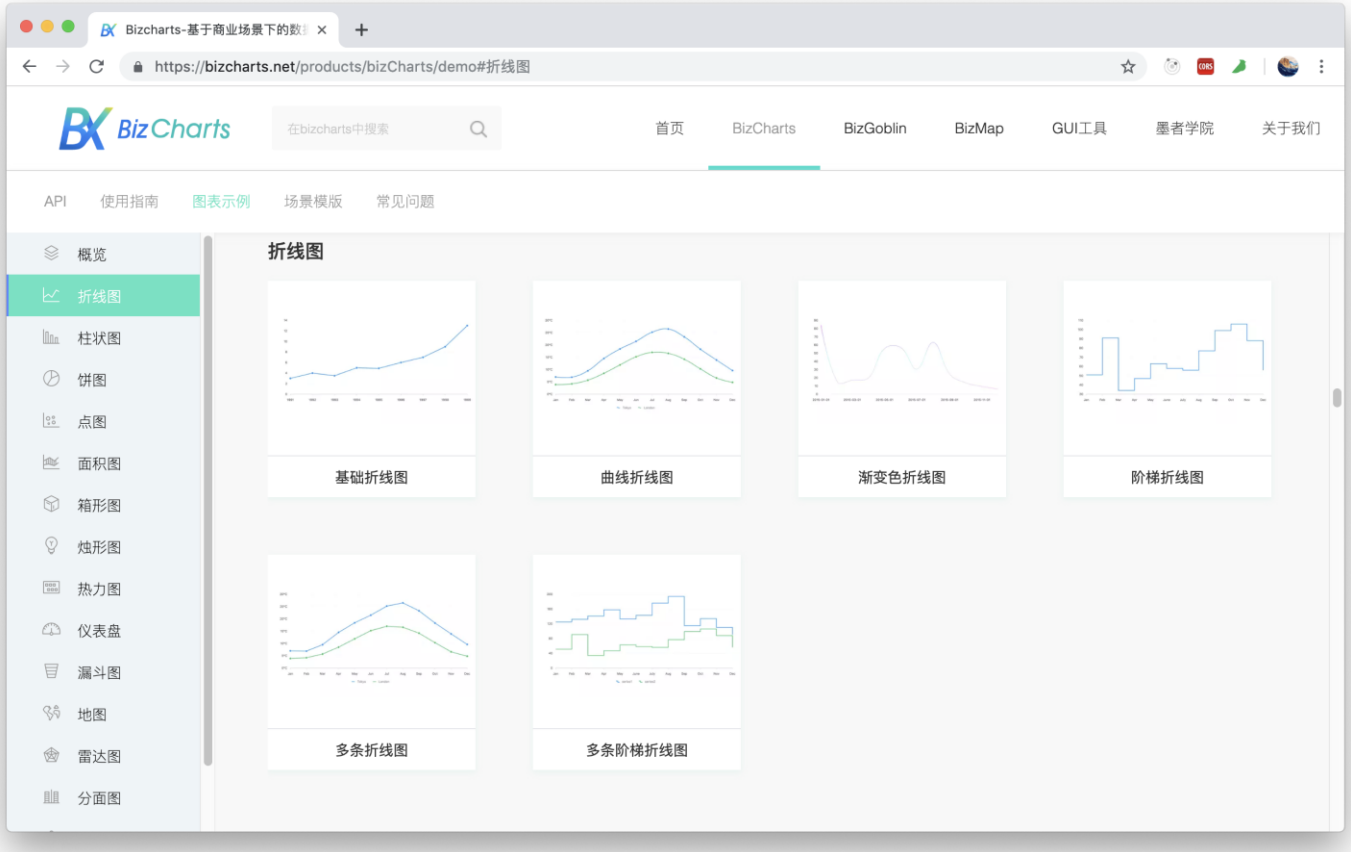
曲线图组件

BizCharts

「BizCharts」是阿里巴巴提供的开源的图表类组件库，是基于AntV的「G2」在React下的实现，支持丰富的数据图形化表达。温湿度曲线，使用bizchart提供的「折线图」绘制，温湿度各绘制一条，采用双坐标轴。

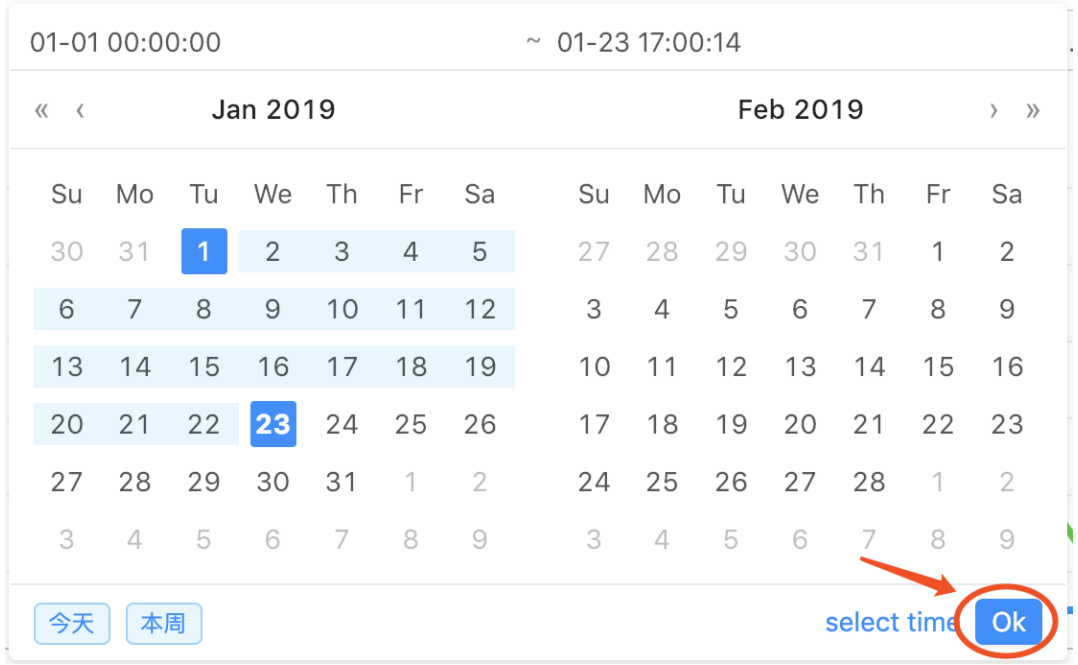
由于umi默认没有集成Bizcharts，可以通过yarn或npm命令在项目内安装：

```
yarn add bizcharts
或
npm install --save bizcharts
```



DatePicker

「DatePicker日期选择框」组件是Antd中用于选择时间，内部包含DatePicker、WeekPicker、MonthPicker等子类，「RangerPicker」是DatePicker中选择时间段的组件，通过API中onOk()提供的回调函数，可以获得点击组件弹窗确认（ok）按钮后的起止日期参数：



RangerPicker可以通过defaultValue属性设定默认值，通过ranges属性可以快速选择时间段，为了方便生成各类时间，我们使用「moment.js」工具包加快效率，moment.js提供丰富的各类数据获取接口，方便快速调用，在项目内执行以下命令安装：

```
yarn add moment
或
npm install --save moment
```

曲线图组件

Input

填写温度报警阈值需要输入框提供输入，Ant Design提供「InputNumber数字输入框」组件进行输入，并对输入变化提供onChange() API接口方便实时获取到最新的输入值，提供最大最小值步长限定，提供小数点后6位的输入精度。

Button

「Button按钮」组件用于执行用户点击回调后的操作，需要取得时间选择框或者温度阈值输入框中的数值，将数值作为请求参数调用后端服务请求数据，通过提供的onClick()回调函数API响应用户的点击。

图表数据

使用Bizcharts图表展示设备的温湿度时序数据，在组件前面加入与API文档一致的数据，用data标识，此外报警阈值使用threshold标识：

```
exercise2
├── README.md
├── mock
├── package.json
├── src
│   ├── app.js
│   ├── assets
│   │   └── yay.jpg
│   ├── global.css
│   ├── layouts
│   │   ├── __tests__
│   │   │   └── index.test.js
│   │   ├── index.css
│   │   └── index.js
│   ├── models
│   └── pages
│       ├── __tests__
│       │   └── index.test.js
│       ├── index.css
│       ├── index.js
│       └── main
│           ├── components
│           │   ├── Card
│           │   │   ├── AlarmStatus.js
│           │   │   ├── HumiStatus.js
│           │   │   ├── OnlineStatus.js
│           │   │   └── TempStatus.js
│           │   ├── Chart.js
│           │   └── Table.js
│           └── index.js
├── tslint.yml
├── webpack.config.js
├── yarn-error.log
└── yarn.lock
```

温度阈值

```
const threshold = 11;
const data = [{
    "currentTemperature": 10.0,
    "currentHumidity": 20.0,
    "gmtCreate": "2019-01-14 18:16"
}, {
    "currentTemperature": 12.0,
    "currentHumidity": 23.0,
    "gmtCreate": "2019-01-15 09:47"
}, {
    "currentTemperature": 1.0,
    "currentHumidity": 56.0,
    "gmtCreate": "2019-01-16 13:47"
}, {
    "currentTemperature": 1.0,
    "currentHumidity": 80.0,
    "gmtCreate": "2019-01-16 14:20"
}, {
    "currentTemperature": 1.0,
    "currentHumidity": 34.0,
    "gmtCreate": "2019-01-16 14:32"
}, {
    "currentTemperature": 10.0,
    "currentHumidity": 50.0,
    "gmtCreate": "2019-01-16 15:57"
}, {
    "currentTemperature": 1.0,
    "currentHumidity": 40.0,
    "gmtCreate": "2019-01-16 16:57"
}, {
    "currentTemperature": 1.0,
    "currentHumidity": 40.0,
    "gmtCreate": "2019-01-16 17:02"
}, {
    "currentTemperature": 2.0,
    "currentHumidity": 40.0,
    "gmtCreate": "2019-01-17 14:18"
}];
```

温度、湿度、时间

图表样式

Bizcharts提供了丰富的demo源码可供参考学习，缺少双坐标轴的例程，只需要添加一个<Axis>标签用于展示设备湿度属性的坐标轴即可，示例如下（详细参见exercise2…/component/Chart.js）

```
<Chart padding={[60, 100, 100, 'auto']} height={400} data={data} scale={cols} forceFit>
  <Legend/>
  <Axis name="gmtCreate" title />                                          时间横轴
  <Axis
    name="currentTemperature"                                              温度竖轴
    title
    label={{ formatter: val => `${val}°C` }}
  />
  <Axis
    name="currentHumidity"                                                 湿度竖轴
    title
    label={{ formatter: val => `${val}%` }}
  />
  <Tooltip                                                                 提示栏
    // title={"hh"}
    crosshairs={{
      type: "y"
    }}
  />
  <Geom
    type="line"                                                            温度点
    position="gmtCreate*currentTemperature"
    size={4}
    // shape={"smooth"}
    color={"#0099ff"}
  />
  <Geom
    type="line"                                                            湿度点
    position="gmtCreate*currentHumidity"
    size={4}
    color={"#33cc33"}
// shape={"smooth"}
  />
  <Guide>
    <Line                                                                  温度阈值
      top={true} // 指定 guide 是否绘制在 canvas 最上层，默认为 false，即绘制在最下层      辅助线
      start={['min', threshold]} // 辅助线起始位置，值为原始数据值，支持 callback
      end={['max', threshold]} // 辅助线起始位置，值为原始数据值，支持 callback
      lineStyle={{
        stroke: '#FF5C68', // 线的颜色
        lineDash: [0, 2, 2], // 虚线的设置
        lineWidth: 3 // 线的宽度
      }} // 图形样式配置
      text={{
        position: 'start', // 文本的显示位置
        autoRotate: true, // 是否沿线的角度排布，默认为 true
        style: { fill: "red" }, // 文本图形样式配置
        content: "温度报警阈值:" + threshold + "℃", // 文本的内容
        offsetY: -5 // y 方向的偏移量
      }}
    />
  </Guide>
</Chart>
```

图表样式

Bizcharts提供了丰富的demo源码可供参考学习，缺少双坐标轴的例程，只需要添加一个<Axis>标签用于展示设备湿度属性的坐标轴即可，示例如下（详细参见component/Chart.js）

最终呈现

HistoricalAlarmTable

设备历史报警数据表格在设计草图中没有完整体现，使用Ant Design提供的表格组件即可展现历史报警的列表，通过RangePicker组件选择起止时间对报警数据进行查询。

Table组件

「Table表格」组件提供行列的数据展示，并且提供丰富的API功能，通过column属性配置列的样式和内容，通过dataSource配置数据源，通过pagination配置分页大小。

- 「Column」配置

Column配置项较多，较为简单的配置实现**dataIndex（数据索引）**、**title（标题显示）**属性即可，Table组件会依据dataIndex中配置的字段解析dataSource中的数据，title为该dataIndex需要显示的名称，为渲染出的表格的头部信息。在本期例程中，配置的Column属性如下：

```
const columns = [{
  title: '设备名',
  dataIndex: 'deviceName',
}, {
  title: '报警温度值',
  dataIndex: 'alarmTemperature',
}, {
  title: '报警时温度阈值',
  dataIndex: 'tempThreshold',
}, {
  title: '创建时间',
  dataIndex: 'gmtCreate'
}];
```

Table组件

「dataSource」配置

dateSource为Table的渲染内容，送入JSONArray类型的数据即可，JSONArray的每个单元中的字段与Column中的dataIndex字段对应。

「pagination」配置

pagination属性配置与pagination组件一致（实际渲染也是pagination组件），可通过pageSize属性设定每页显示的数据条数。

表格数据

```
const data = [{
    "id": 1,
    "deviceName": "test_zhinengjiaju",
    "alarmTemperature": 10.0,
    "tempThreshold": 30.0,
    "gmtCreate": "2019-01-14 18:22"
}, {
    "id": 2,
    "deviceName": "test_zhinengjiaju",
    "alarmTemperature": 10.0,
    "tempThreshold": 15.0,
    "gmtCreate": "2019-01-15 10:32"
}, {
    "id": 3,
    "deviceName": "test_zhinengjiaju",
    "alarmTemperature": 10.0,
    "tempThreshold": 15.0,
    "gmtCreate": "2019-01-15 14:46"
}, {
    "id": 4,
    "deviceName": "test_zhinengjiaju",
    "alarmTemperature": 10.0,
    "tempThreshold": 15.0,
    "gmtCreate": "2019-01-15 14:47"
}];
```

表格组件 详细参考Table.js文件

表格组件


```
import React from 'react'
import 'antd/dist/antd.css';
import { Table, DatePicker, Button } from 'antd';
import moment from 'moment';

const { RangePicker } = DatePicker;
const dateFormat = "MM-DD HH:mm:ss"

export default class HistoricalAlarmTable extends React.Component {

    constructor(props) {
      super(props);
      this.state = {
        beginTime: moment().startOf('month').format('YYYY-MM-DD HH:mm:ss'),
        endTime: moment().format('YYYY-MM-DD HH:mm:ss'),
      }
    }

    render() {
    const datePickerOnOk = (dates) => {
      this.setState({
        beginTime: dates[0].format('YYYY-MM-DD HH:mm:ss'),
        endTime: dates[1].format('YYYY-MM-DD HH:mm:ss'),
      });
    }

    const rangerOnClick = () => console.log("点击起止时间提交");

    return (
      <div>
        <div style={{ paddingBottom: 20 }}>
          选择查询时间  
          <RangePicker
            defaultValue={[moment().startOf('month'), moment()]}
            ranges={{ '今天': [moment().startOf('day'), moment()], '本周': [moment().startOf
            onOk={datePickerOnOk}
            format={dateFormat}
            showTime={true}
          />
           
          <Button type="primary" style={{ display: 'abslute', top: 0 }} onClick={rangerOnCl
            提交
          </Button>
        </div>
        <Table
          columns={columns}
          dataSource={data}
          pagination={{ pageSize: 5 }}
        />
      </div>
    )
  }
}
```

初始化起止时间

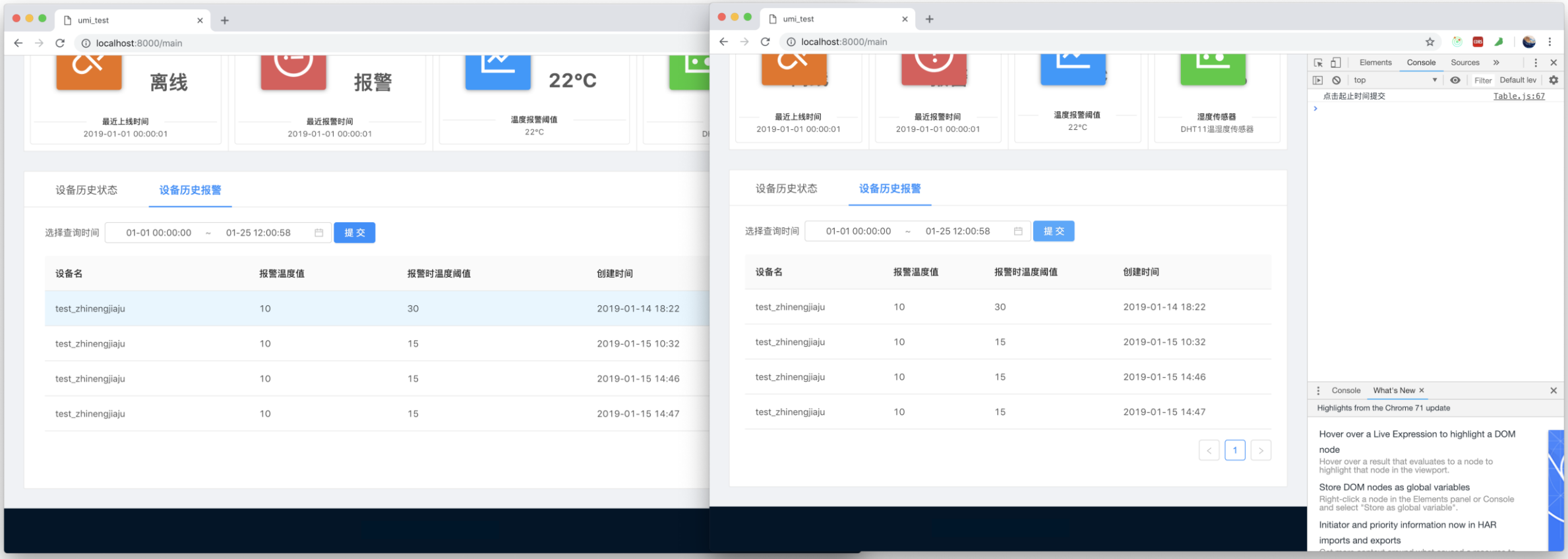
选择起止时间回调函数

起止时间选择与提交

表格组件渲染

表格组件

表格组件



到目前为止，**完成了UI层的开发**，阶段参考代码在**仓库地址/exercise2文件夹**下运行前命令行执行yarn安装项目依赖

life.augmented 阿里云

第四章.第三节 内容简介

- 前端基础概念和知识
- 认识前端框架
  - 了解React
  - 学习Umi.js
  - 学习Ant Design
  - 了解dva.js
- 初始化并运行第一个前端项目
  - 安装yarn包管理工具
  - 全局安装umi
  - 使用脚手架初始化
  - Umi路由页面添加
- 创建和使用组件
  - Layout组件
  - 主页面创建
  - 组件引入
  - 组件完善
- 使用dva实现数据流转
  - 数据请求
  - Dva管理数据
- 应用调试与部署
  - 项目打包
  - 项目部署

Axios请求库

「axios」是流行的HTTP请求库，可快速集成在项目中，可以通过以下命令快速安装axios：

```
yarn add axios
或
npm install --save axios
```

axios请求可以通过async await取得最终的请求返回数据。基于axios封装get和post的函数工具，在一级目录下新建axios文件夹，新建tools.js文件：

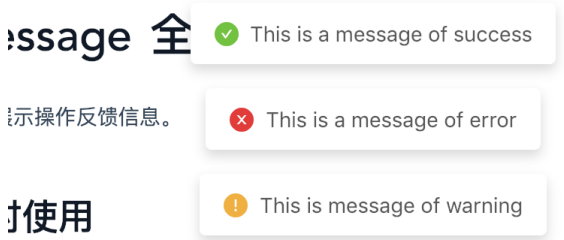
```
import axios from 'axios'
//通过async和await进行promise对象的处理
export default {
  async post(url, data) {
    return await axios({
      method: 'post',
      url,
      data: data,   //post请求参数
      timeout: 10000, //超时时间
    })
  },
  async get(url, data) {
    return await axios({
      method: 'get',
      url,
      params: data, //get请求时带的参数
      timeout: 10000, //超时时间
    })
  }
}
```

post方法

get方法

Message弹窗组件

Message是页面全局提示组件，可以用来对于请求异常返回码进行弹窗，提示网络状况。编写函数对请求返回code码进行判断。



判断返回码的函数

```
const checkStatus = (res) => {
  if (res.status !== 200) {
    message.error('请求失败或网络异常，请检查···');
  } else {
    if (res.data.code !== 0) {
      message.error('提交失败或服务器异常，请检查···');
    }
  }
}
```

API封装

在tool.js中定义了请求的方法，将各请求写入组件将变得难以维护和管理，统一将API请求写进axios/index.js。

```
import http from './tools';
import { message } from 'antd';
const url = 'http://xxxx:xxxx/api/v1/device'; //请换成你的请求地址，可用请求地址http://47.110.
```

检查返回码并进行错误弹窗

```
const checkStatus = (res) => {
  if (res.status !== 200) {
    message.error('请求失败或网络异常，请检查··
  } else {
    if (res.data.code !== 0) {
      message.error('提交失败或服务器异常，请检查···');
    }
  }
}
```

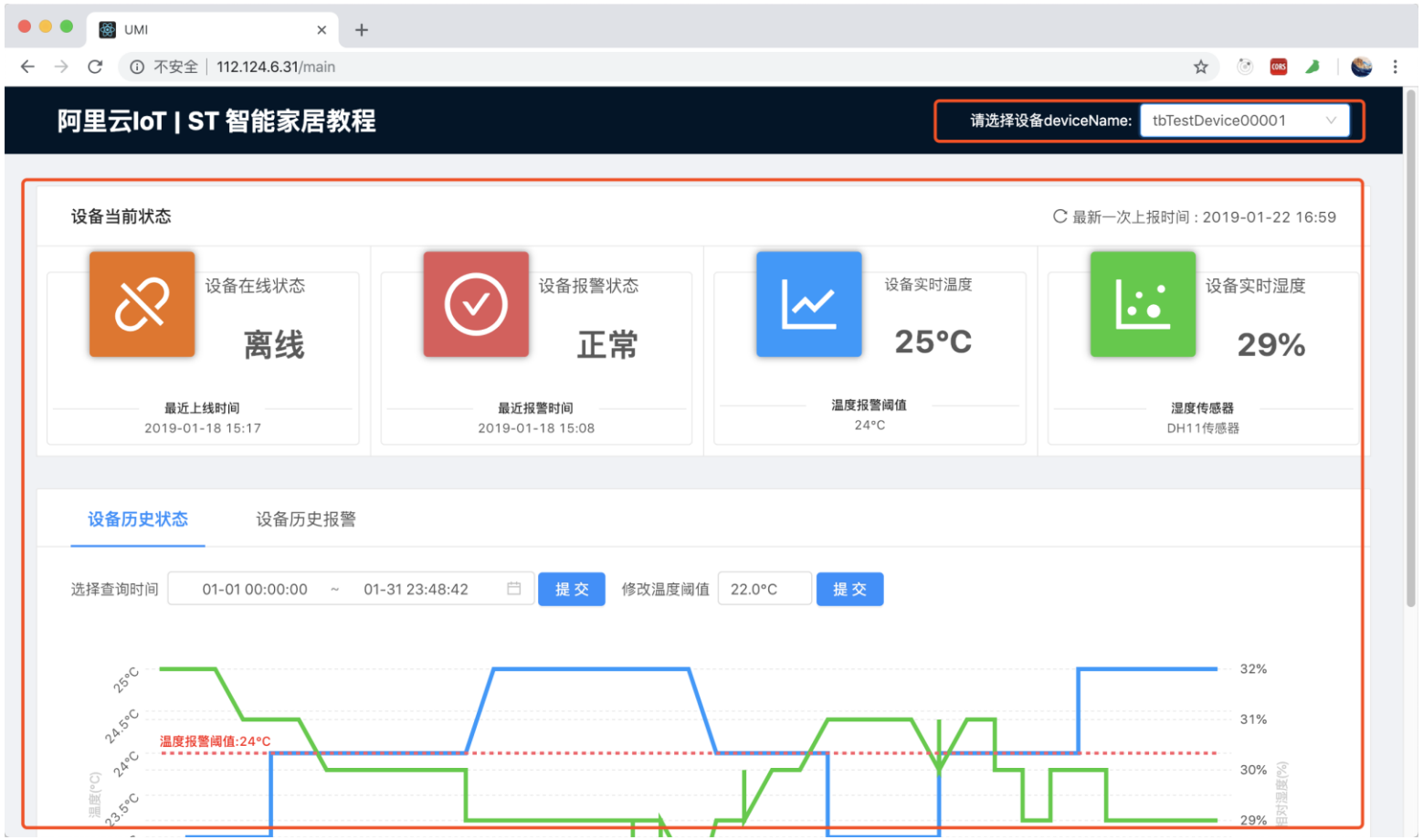
API请求封装

```
//查询设备的列表
export const getDeviceList = async () => {
  //得到原始输出
  let res = await http.get(url + '/listDeviceName', null);
  checkStatus(res);
  return res.data;
};

//查询设备的属性
export const queryDeviceProp = async (params) => {
  let res = await http.get(url + '/queryDeviceProp', params);
  checkStatus(res);
  return res.data;
}

//查询设备的属性
export const selectChartDataByTime = async (params) => {
  let res = await http.get(url + '/queryDevicePropHistoryLogs', params);
  checkStatus(res);
  return res.data;
}
```

API请求封装

```
//设置报警阈值
export const setDeviceProperty = async (params) => {
  let res = await http.get(url + '/setDeviceProperty', params);
  checkStatus(res);
  return res.data
}

//查询历史报警记录
export const queryAlarmHistoryLogs = async (params) => {
  let res = await http.get(url + '/queryAlarmHistoryLogs', params);
  checkStatus(res);
  return res.data
}

//清除报警效果
export const clearAlarm = async (params) => {
  let res = await http.get(url + '/clearAlarm', params);
  checkStatus(res);
  return res.data
}
```

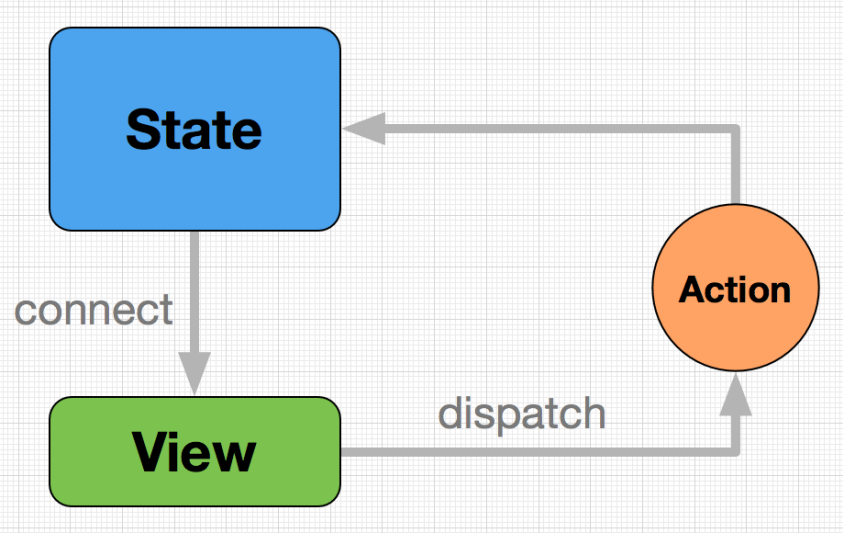
dva管理数据



问题：**如何在header组件内选择不同设备，主页自动请求跳转此设备相关的数据？**

dva提供了简单易用的解决方案。**所有的 组件存储在一个状态对象state中，各组件通过请求去改变state。**

dva管理数据



State是中心数据仓库，通过组件的props连接到组件内部，组件发生操作会dispatch一个action给state，从而通知state完成改变。

随着前端组件增多，功能划分分散，所有的组件维护同一个state是非常危险的。Model是dva中state的单元，每个前端功能块一个model实现解耦。

Model，state的单元

多个小的 model 的 state 以 namespace 为 key 合成，开发者可以依照业务逻辑，通过namespace来对state进行划分，最终交给dva处理

model的常用配置有namespace、state、reducers、effects：

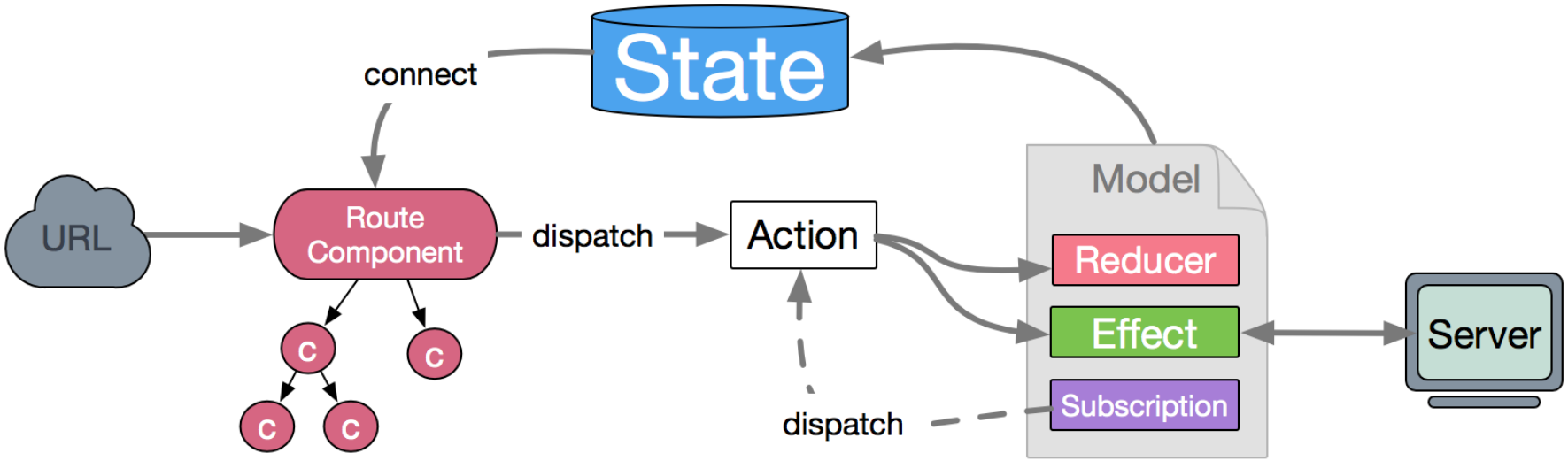
```
{
  namespace: 'count',
  state: 0,
  reducers: {
    add(state) { return state + 1 },
  },
  effects: {
    *addAfter1Second(action, { call, put }) {
      yield call(delay, 1000);
      yield put({ type: 'add' });
    },
  },
}
```

Model，state的单元

dva管理数据

```
{
  namespace: 'count',
  state: 0,
  reducers: {
    add(state) { return state + 1 },
  },
  effects: {
    *addAfter1Second(action, { call, put }) {
      yield call(delay, 1000);
      yield put({ type: 'add' });
    },
  },
}
```

- namespace：当前 Model 的名称。整个应用的 State，由多个小的 Model 的 State 以 namespace 为 key 合成。
- state：该 Model 当前的状态。数据保存在这里，直接决定了视图层的输出。
- reducers：Action 处理器，处理同步动作，用来算出最新的 state（必须返回一个新的state）。
- effects：Action 处理器，处理异步动作（进行网络请求的调用，再调用执行reducers刷新state）。



编辑主页model

dva管理数据

新建主页model，新建main/models/main.js，分为namespace、state、reducer和effect

1. 引入封装好的API函数

```
import {
    getDeviceList,
    queryDeviceProp,
    selectChartDataByTime,
    setDeviceProperty,
    queryAlarmHistoryLogs,
    clearAlarm
} from '../../../axios/index';
import { message } from 'antd';
```

2. 将各组件所需数据按照格式初始化在state

```
//储存model的state状态
state: {
    deviceName: "test_zhinengjiaju",
    deviceSelectList: ["test_zhinengjiaju"],
    alarmCard: {
        state: true,
        recentAlarmTime: null
    },
    onlineCard: {
        state: "在线",
        recentOnlineTime: null
    },
    tempCard: {
        value: 22,
        threshold: 22
    },
    humiCard: {
        value: 43,
        sensor: "DHT11温湿度传感器"
    },
    updateTime: "待初始化",
    chart: {
        dataList: [],
    },
    table: {
        data: []
    }
},
```

3.Reducer为新的state合并到旧的函数

```
reducers: {
    //自动合并新的状态到旧的状态
    update(state, { payload }) {
        let newState = payload;
        return {
            ...state,
            ...newState
        };
    },
},
```

？？？

state

```
{
    devicName : "oldDevice",
    deviceData : "oldDeviceData"
}
```

newState

```
{
    devicName: "newDevice",
}
```

{...state,...newStatus}

```
{
    devicName: "newDevice",
    deviceData : "oldDeviceData"
}
```

[Example]State和newStatus做merge

life.augmented

阿里云

编辑主页model

新建主页model，分为namespace、state、reducer和effect

4. Effect是API接口函数

```
effects: {
    *getDeviceList({ payload }, { call, put }) {
        let res = yield call(getDeviceList);
        yield put({
            type: 'update',
            payload: {
                deviceSelectList: res.data
            },
        });
    },
    //周期性更新数据，封装成newState更新
    *queryDeviceProp({ payload }, { call, put }) {
        let res = yield call(queryDeviceProp, payload);
        yield put({
            type: 'update',
            payload: {
                alarmCard: {
                    state: res.data.alarmStatus,
                    recentAlarmTime: res.data.lastAlarmDate
                },
                onlineCard: {
                    state: res.data.connectStatus,
                    recentOnlineTime: res.data.lastOnlineDate
                },
                tempCard: {
                    value: res.data.currentTemperature,
                    threshold: res.data.tempThreshold
                },
                humiCard: {
                    value: res.data.currentHumidity,
                    sensor: "DHT11温湿度传感器"
                },
                updateTime: res.data.gmtUpdate
            },
        });
    },
    //根据起止时间查询设备温湿度数据
    *selectChartDataByTime({ payload }, { call, put, select }) {
        let deviceName = yield select(state => state.main.deviceName);
        let res = yield call(selectChartDataByTime, { ...payload, deviceName: deviceNam
        yield put({
            type: 'update',
            payload: {
                chart: {
                    dataList: res.data
                },
            }
        });
    },
    //设定报警超限值
    * setAlarmThreshold({ payload }, { call, select }) {
        let deviceName = yield select(state => state.main.deviceName);
        let res = yield call(setDeviceProperty, { ...payload, deviceName: deviceName });
        if (res.code === 0) {
            message.success('提交成功···');
        }
    },
    //根据起止时间查询报警数据
    * selectTableDataByTime({ payload }, { call, put, select }) {
        let deviceName = yield select(state => state.main.deviceName);
        let res = yield call(queryAlarmHistoryLogs, { ...payload, deviceName: deviceName });
        yield put({
            type: 'update',
            payload: {
                table: {
                    data: res.data
                }
            }
        });
    },
    //清除报警
    * clearAlarm({ payload }, { call, select }) {
        console.log("执行了");
        let deviceName = yield select(state => state.main.deviceName);
        let res = yield call(clearAlarm, { deviceName: deviceName });
        if (res.code === 0) {
            message.success('提交成功···');
        }
    },
}
```

Model与组件的connect

dva管理数据

在model中定义了组件所需要的数据和网络请求函数，用connect将组件所需数据和请求函数绑定。

「connect」将model和组件联系在一起，可以让组件访问state中的数据，调用state中reducer方法和effect方法，一个model可以connect到多个组件，多个组件可以共用一整套数据，共享一整套方法，model中数据的改动会实时改变在组件的UI显示上（通过props传入，组件对应的生命周期函数会被调用（componentWillReceiveProps））。

•connect state数据

```
import { connect } from 'dva';

const mapStateToProps = (state) => {
    const cardContent = state.main.humiCard;
    return {
        cardContent,//connect后，组件通过this.props.cardContent取值
    };
};

@connect(mapStateToProps)
export default class HumiStatusCard extends React.Component {
  //·····组件内容
  //·····组件内容
  //·····组件内容
  render(){
    return <div>取到state中的数据：{this.props.cardContent} </div>
  }
}
```

1 2 3

将model中的humiCard映射成组件props属性中的cardContent

•connect reducer和effect方法

```
const mapDispatchToProps = (dispatch) => {
  return {
    onWillMount() {
      dispatch({
        type: `main/getDeviceList`,
      });
    },
    peroidlyUpdate(params) {
      dispatch({
        type: `main/queryDeviceProp`,
        payload: params
      })
    },
    onDeviceSelected(params) {
      dispatch({
        type: 'main/update',
        payload: params
      })
    }
  };
};

@connect(mapStateToProps, mapDispatchToProps)
export default class BasicLayout extends React.Component {

  //组件即将加载时执行，请求设备列表并刷新state中的设备列表
  async componentWillMount() {
    await this.props.onWillMount();
  }
  //组件加载后执行，设定定时器定时刷新设备最新状态
  componentDidMount() {
    const inteval = setInterval(async () => {
      await this.props.peroidlyUpdate({ deviceName: this.props.deviceName });
    }, 2000);
  }
```

1 2 3

connect方法

调用方法

将model中的getDeviceList等API映射成组件props属性中的onWillMount等

dva数据流转项目下载

Model和组件connect后的项目示例代码打包在码云

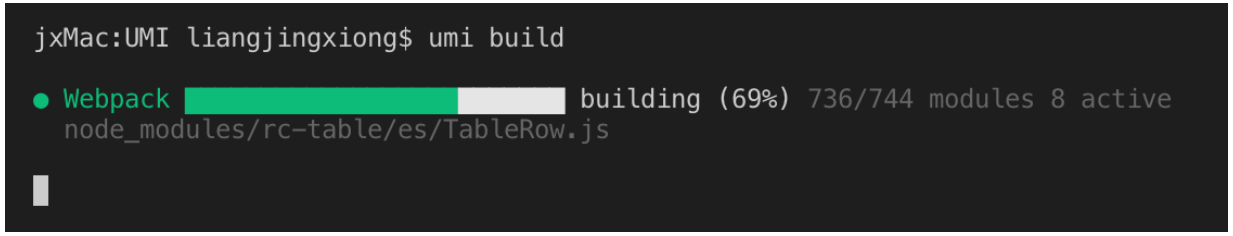
/exercise3下，启动前通过yarn命令安装依赖

第四章.第三节 内容简介

- 前端基础概念和知识
- 认识前端框架
  - 了解React
  - 学习Umi.js
  - 学习Ant Design
  - 了解dva.js
- 初始化并运行第一个前端项目
  - 安装yarn包管理工具
  - 全局安装umi
  - 使用脚手架初始化
  - Umi路由页面添加
- 创建和使用组件
  - Layout组件
  - 主页面创建
  - 组件引入
  - 组件完善
- 使用dva实现数据流转
  - 数据请求
  - Dva管理数据
- 应用调试与部署
  - 项目打包
  - 项目部署

项目打包

前端项目文件多，体系复杂，一般使用打包工具对项目进行打包压缩，最终只需要部署压缩后的文件。umi集成了打包工具webpack，在项目主目录下执行umi build：




```
$ umi build
✔ Webpack
  Compiled successfully in 24.68s

 DONE  Compiled successfully in 24685ms

File sizes after gzip:

  700.16 KB  dist/umi.js
  73.63 KB   dist/umi.css
```

打包构建的输出在dist文件夹下，众多的js文件被压缩打包进umi.js，css被打包压缩进umi.css，图片等前端素材被打包进static文件夹，打包后目录结构如下所示：

```
./dist/
├── index.html
├── static
│   └── yay.44dd3333.jpg
├── umi.css
└── umi.js
```

Nginx服务器部署

执行umi build打包后的前端内容想要实现应用文件的高性能分发，还需要将项目部署到高性能服务器上去，「nginx」是一款流行的免费开源服务器软件，配置简单，性能强大，经常在生产环境中使用。

Linux环境下部署

1. 在命令行执行以下命令安装nginx：

```
sudo apt-get install nginx
```

3. 重启nginx服务：

```
service nginx start
```

访问ip地址，即可访问nginx部署好的项目内容。

2. 修改/etc/nginx/nginx.config配置文件内容

```
#user  nobody;
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       mime.types;
    default_type  application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    #access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

     server {
        listen
        root /home/dist;  //修改成为build文件所在的目录
        index index.html index.htm;

        location / {
            try_files $uri $uri/ /index.html;
        }
    }
}
```

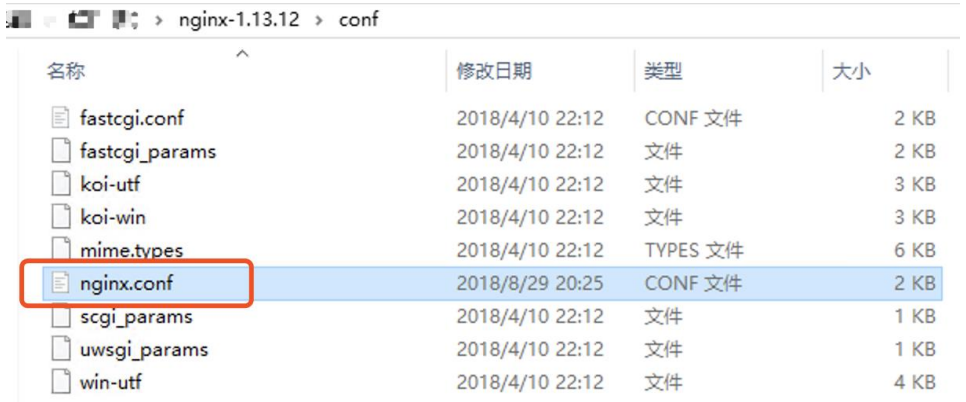
修改为build后文件夹路劲，通过80端口访问

项目部署

执行umi build打包后的前端内容已经可以在本地计算机的浏览器中运行，想要实现应用文件的高性能分发，还需要将项目部署到高性能服务器上去，「nginx」是一款流行的免费开源服务器软件，配置简单，性能强大，经常在生产环境中使用。

Windows环境下部署

windows环境下nginx配置内容与Ubuntu基本一致，目录地址与Ubuntu稍有差异，文件系统地址需要改为“/”，nginx 1.14版本「下载地址」，下载解压后，修改conf文件夹下nginx.config内容即可，点击解压文件夹下.exe文件，即可启动nginx，windows环境下nginx的重启与关闭可以通过任务管理器执行。




```
#user  nobody;
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       mime.types;
    default_type  application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    #access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;
    server {
        listen       80;
    #    listen       somename:80;
    #    server_name  somename  alias  another.alias;
        root C:\Users\Administrator\Desktop\wanwu\dist;
        index index.html index.htm;

        location / {
            try_files $uri $uri/ /index.html;
        }
    }
}
```

修改为build后文件夹路径，80端口访问

总结

本次教程帮助广大物联网硬件开发者实现前端开发入门，前端开发是非常流行并且快速发展的领域，技术更新迭代周期较快，希望本次例程能够成为您探索物联网前端开发领域所迈出的第一步。

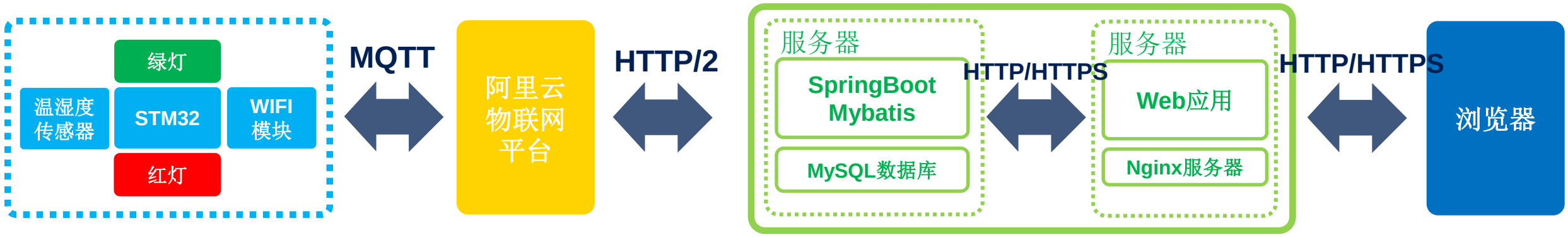
参考资料

「HTML教程」：http://www.runoob.com/html/html-tutorial.html
「CSS教程」：http://www.runoob.com/css/css-tutorial.html
「JavaScript教程」：http://www.runoob.com/js/js-tutorial.html
「React学习教程」：http://www.runoob.com/react/react-tutorial.html
「React中文文档」：https://react.docschina.org/
「UMI官方网站」：https://umijs.org/zh/
「Ant Design官方网站」：https://ant.design/index-cn
「DVA官网网站」：https://dvajs.com/

课程表

课程章节	模块内容	详细目录
（一）课程指南	1. 课程要解决的痛点	课程场景介绍，数据路径端到端
	2. 课程适用于不同资源水平的节点设备	低配版节点设备、高配版设备
	3. 课程所需具备的软、硬件	
（二）阿里云IoT平台介绍	1. 物联网平台简介	物联网平台简介
	2. 物联网平台基础概念讲解	设备相关概念
		平台相关概念
（三）基于STM32的节点设备接入阿里云IoT平台	1. 基于STM32的节点端及开发环境介绍	STM32产品介绍：十四大家族和IoT策略
		STM32生态系统介绍：STM32Cube
		STM32L4R5以及Nucleo-L4R5介绍
		ST sensor板和EMW3080板介绍
	2. 基于Paho MQTT的直连（适用于资源受限设备）	Demo运行起来
		MQTT协议介绍
		Demo介绍
	3. 基于Linkkit C-SDK的MQTT直连（适用于资源丰富设备）	Demo运行起来
		Linkkit C-SDK介绍
		Demo介绍
（四）服务器端的应用开发	1.综合软件架构介绍	软件架构介绍
		知识结构梳理
	2.后端服务开发	认识后端框架
		初始化运行第一个后端项目
		应用系统开发
		应用调试与部署
	3.前端服务开发体验	认识前端框架
		初始化并运行第一个前端项目
		创建和使用组件
		使用dva实现数据流转
		应用调试与部署

课程场景 概览



- 每5秒上报温湿度值，闪烁绿灯
- 温度超【阈值】亮红灯，并在每10秒向用户服务器报警，直到温度恢复【阈值】以下或者收到警报解除消息
- 收到警报解除信息后红灯闪烁
- 温度恢复到【阈值】以下灭红灯
- 湿度值被阿里云IoT转发到用户服务器，进行数据库存储，同时在web端显示近期温湿度数据曲线
- 报警消息被阿里云IoT转发到用户服务器，在web端显示
- 用户通过web端页面解除报警
- 用户通过web端页面设置【阈值】参数

life.augmented | 阿里云

STM32

STM32 Nucleo

STM32 CubeMX

阿里云大学

LONGEVITY 10 YEARS COMMITMENT

10 years of STM32

阿里云

STM32 Cube

STM32公众号

STM中文官网

电堂公众号

阿里云大学