React+开发小程序

1. 课前准备

1. 官网 https://taro.aotu.io/
2. UI组件 https://taro-ui.aotu.io/#/
3. 脚手架
    1. `npm install -g @tarojs/cli`安装

2. 回顾

1. 云开发
2. 获取用户信息

3. 云函数
4. 云存储
5. 云数据库的增删改查

3. 课堂主题

1. 学习taro框架
2. 学习taro-ui跨端的UI框架

4. 课堂目标

1. 使用React开发自己的小程序

5. 知识点

为什么要有跨端的框架

代码组织

每个页面都是.js .wxss .wxml .json四个部分构成

开发方式

不够现代化，虽然现在能用npm了，但是对ES6语法的支持，以及sass等css预处理支持的不是很好

端太多了

1. 微信小程序
2. 网页H5
3. 头条小程序
4. 百度小程序
5. 支付宝小程序
6. 快应用
7. 原生APP
8. ....敬请期待

每一个都独立开发，前端要吐血了💔

跨端

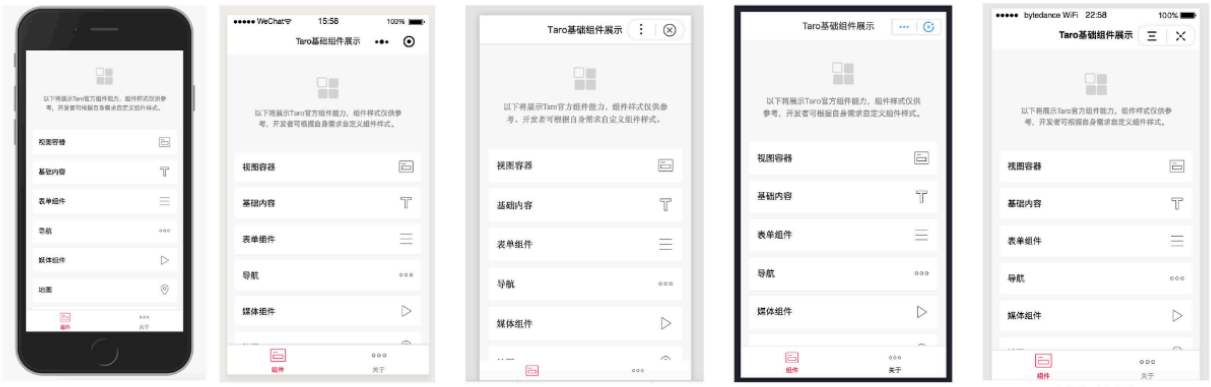
其实由于微信原生小程序开发自己玩了一套自己的语法，所以早早就有用vuejs来开发小程序的框架，比如webpy和mpvue，但是基本都是单纯的开发微信小程序，现在基本对多端支持足够好的，就是taro和uni-app了，分别是使用React和Vue的语法来开发小程序生态

> https://taro.aotu.io/ Taro官网

2. Taro简介

> https://taro.aotu.io/
>
> **Taro** 是一套遵循 React 语法规范的 **多端开发** 解决方案。现如今市面上端的形态多种多样，Web、React-Native、微信小程序等各种端大行其道，当业务要求同时在不同的端都要求有所表现的时候，针对不同的端去编写多套代码的成本显然非常高，这时候只编写一套代码就能够适配到多端的能力就显得极为需要。
>
> 使用 **Taro**，我们可以只书写一套代码，再通过 **Taro** 的编译工具，将源代码分别编译出可以在不同端（微信/百度/支付宝/字节跳动小程序、H5、React-Native 等）运行的代码。

- 符合React语法规范
- 多端开发



3. Taro初体验

安装

```
npm install -g @tarojs/cli

npm install -g babel-cli
npm install -g babel-preset-es2015
```

新建一个工程

```
taro init taro-03
```

- 启动h5

```
# npm script
npm run dev:h5
npm run build:h5
```

- 微信小程序端

```
# npm script
npm run dev:weapp
npm run build:weapp
```

- 浏览器端演示

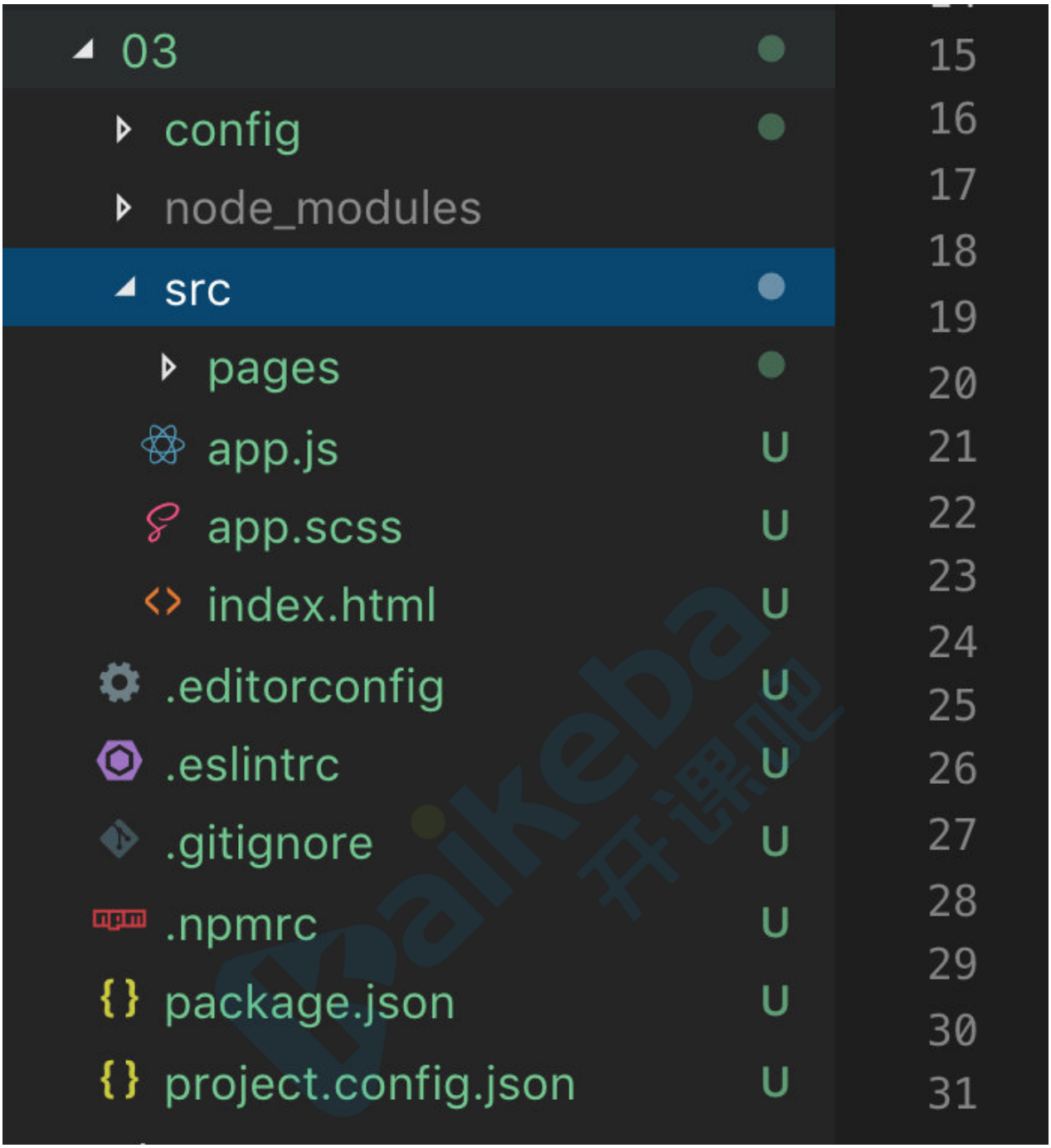
```
# npm script
npm run dev:h5
npm run build:h5
```

- 百度小程序端

```
npm run dev:swan
```

```
➜  mini-program-lesson git:(master) ✗ taro init 03
👽 Taro v1.2.13

Taro即将创建一个新项目！
Need help? Go and open issue: https://github.com/NervJS/taro/issues/new

? 请输入项目介绍！ 第三次课
? 是否需要使用 TypeScript ？ No
? 请选择 CSS 预处理器（Sass/Less/Stylus） Sass
? 请选择模板 默认模板
```

目录结构



核心代码都在src里，

执行`npm run dev:h5`会打开浏览器，预览H5效果

执行npm run dev:weapp 打开微信开发者工具预览dist 查看微信小程序效果

其他

1. 百度小程序 npm run dev:swan
2. 头条小程序： npm run dev:tt
3. 支付宝 npm run dev:alipay
4. React-native npm run dev:rn

开发简单的todolist

打开app.js 集合了app.js和app.json的功能

```
import Taro, { Component } from '@tarojs/taro'
import Index from './pages/index'

import './app.scss'

// 如果需要在 h5 环境中开启 React Devtools
// 取消以下注释:
// if (process.env.NODE_ENV !== 'production' && process.env.TARO_ENV === 'h5')
{
//   require('nerv-devtools')
// }

class App extends Component {

  config = {
    pages: [
      'pages/index/index'
    ],
    window: {
      backgroundTextStyle: 'light',
      navigationBarBackgroundColor: '#fff',
      navigationBarTitleText: 'WeChat',
      navigationBarTextStyle: 'black'
    }
  }

  componentDidMount () { }

  componentDidShow () {}

  componentDidHide () {}

  componentDidCatchError () {}

  // 在 App 类中的 render() 函数没有实际作用
  // 请勿修改此函数
  render () {
    return (
```

```
      <Index />
    )
  }
}

Taro.render(<App />, document.getElementById('app'))
```

config就是 app.json的配置

生命周期也和小程序一一对应

1. componentDidMount onLoad
2. componentDidShow onShow
3. componentDidHide onHide

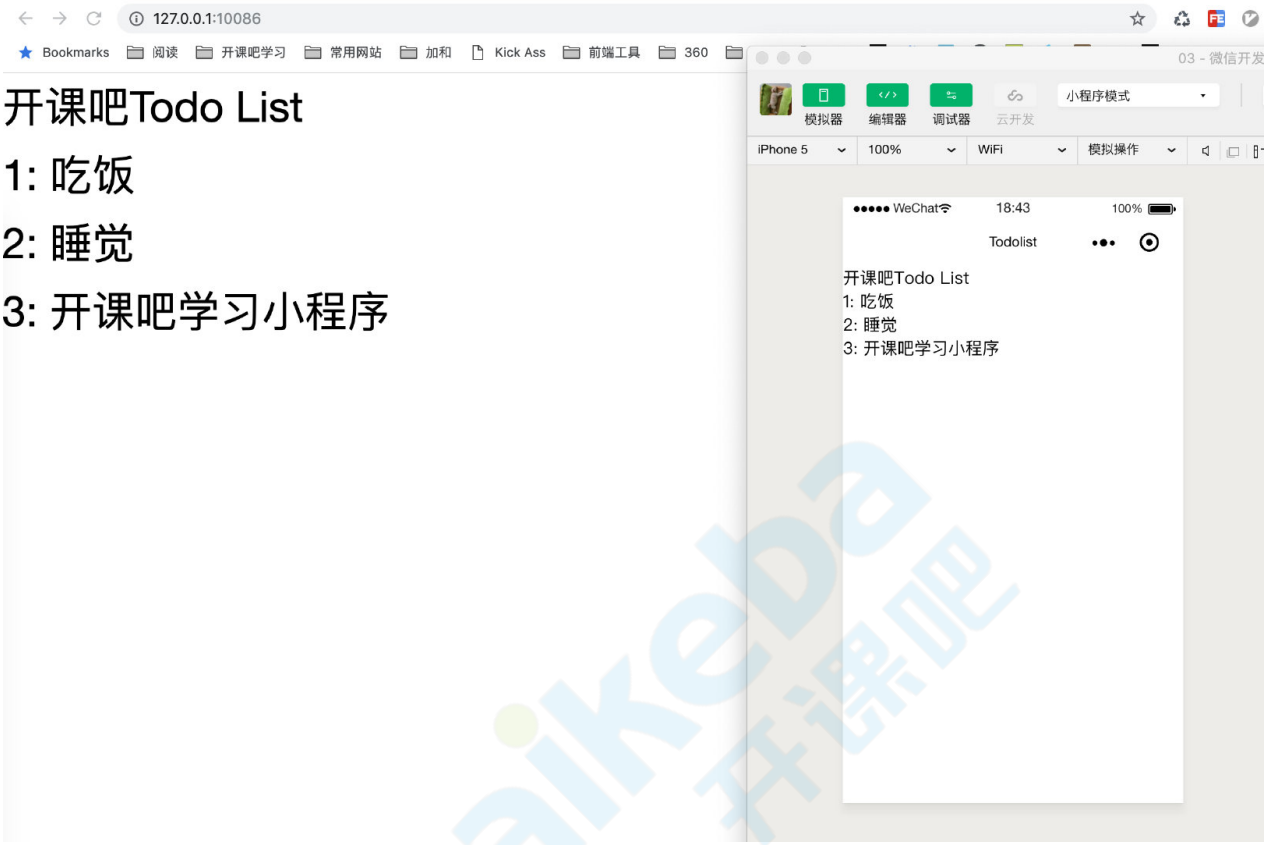
使用state

> https://nervjs.github.io/taro/docs/components-desc.html 组件库

修改pages/index/index.js

```
export default class Index extends Component {

  config = {
    navigationBarTitleText: 'Todolist'
  }
  constructor(props){
    super(props)
    this.state={
      todos:['吃饭','睡觉','开课吧学习小程序'],
      val:''
    }
  }


  render () {
    return (

      <View className='index'>
        <Text>开课吧Todo List</Text>
        {this.state.todos.map((item,i)=>{
          return <View>
              <Text>{i+1}: {item}</Text>
            </View>
        })}
      </View>
    )
```

```
  }
}
```



添加用户输入

```
 // *******Add***************
import { View, Text, Button, Input } from '@tarojs/components'
 // *******Add***************
export default class Index extends Component {

  config = {
    navigationBarTitleText: 'Todolist'
  }
  constructor(props){
    super(props)
    this.state={
      todos:['吃饭','睡觉','开课吧学习小程序'],
      val:''
    }
  }
```

```
  handleInput = (e)=>{
    this.setState({
      val:e.target.value
    })
  }
  handleClick = ()=>{
    this.setState({
      todos:[...this.state.todos,this.state.val],
      val:''
    })
  }
  render () {
    return (
      <View className='index'>
        <Text>开课吧Todo List</Text>
        // *******Add**************
        <Input value={this.state.val} onChange={this.handleInput}></Input>
        <Button onClick={this.handleClick}> 添加</Button>
         // *******Add**************
        {this.state.todos.map((item,i)=>{
          return <View>
              <Text>{i+1}: {item}</Text>
            </View>
        })}
      </View>
    )
  }
}
```

使用UI组件

```
npm install taro-ui --save
npm install nervjs --save
```

> 官网 https://taro-ui.aotu.io/#/docs/introduction
>
> 主题选择器 https://nervjs.github.io/taro-ui-theme-preview/

拷贝custom-theme.scss到app.scss上面

```scss
/* Custom Theme */
$color-brand: #e93b3d;
$color-brand-light: #ef6c6e;
$color-brand-dark: #ba2f31;

@import "~taro-ui/dist/style/index.scss";
// 引入组件样式，仅需引入一次即可
```

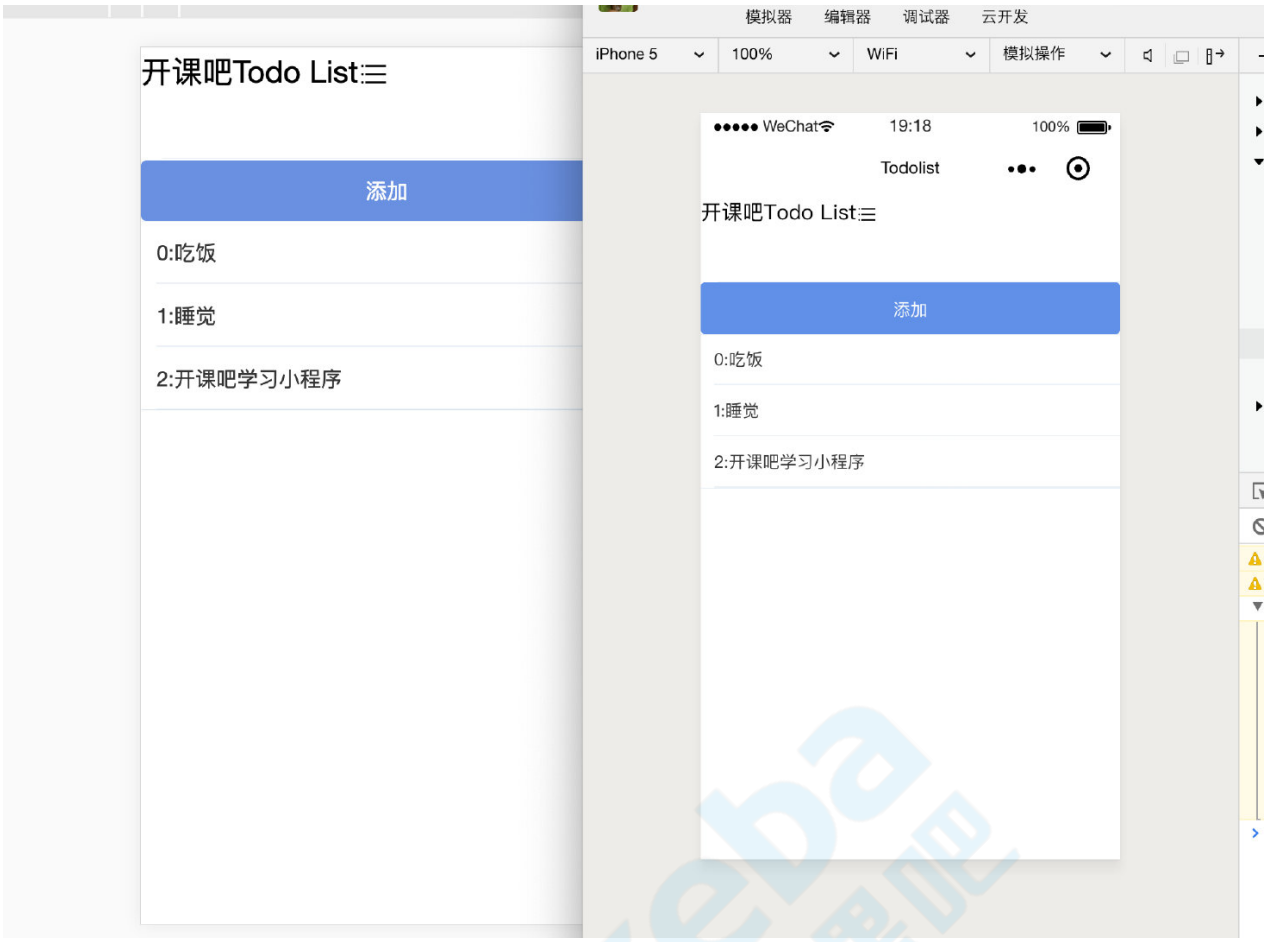
// H5兼容性测试

由于引用 `node_modules` 的模块，默认不会编译，所以需要额外给 H5 配置 `esnextModules`，在 taro 项目的 `config/index.js` 中新增如下配置项：

```
// config/index.js
h5: {
  esnextModules: ['taro-ui']
}
```

```
import Taro, { Component } from '@tarojs/taro'
import { View, Text,Input } from '@tarojs/components'

import { AtButton,AtInput,AtList, AtListItem } from 'taro-ui'
import './index.scss'

export default class Index extends Component {

  config = {
    navigationBarTitleText: 'Todolist'
  }
  constructor(props){
    super(props)
    this.state={
      todos:['吃饭','睡觉','开课吧学习小程序'],
      val:''
    }
  }
//****** update
  handleInput = (val)=>{
    this.setState({val})
  }
  //****** update
```

```
  handleClick = ()=>{
    this.setState({
      todos:[...this.state.todos,this.state.val],
      val:''
    })
  }
  render () {
    return (
      <View className='index'>
        <Text>开课吧Todo List</Text>
        <View className='at-icon at-icon-bullet-list'></View>
        <AtInput value={this.state.val} onChange={this.handleInput}></AtInput>

        <AtButton type='primary' onClick={this.handleClick}>123</AtButton>
        {/* <Input value={this.state.val} onInput={this.handleInput}></Input>
*/}
        <AtList>
        {this.state.todos.map((item,i)=>{
          return <AtListItem title={i+':'+item}>

          </AtListItem>

        })}

        </AtList>

      </View>
    )
  }
}
```



使用mobx

基本上React数据管理就是redux和mobx的天下，redux我们已经很熟悉了，那么mobx呢

mobx主打的是类似vue的响应式

`npm install --save mobx@4.8.0 @tarojs/mobx @tarojs/mobx-h5 @tarojs/mobx-rn`

src下新建`store/counter.js`

```
// src/store/todo.js
import { observable } from 'mobx'

const todoStore = observable({
  todos: [],

  addTodo(item) {
    this.todos.push(item)
  }

})

export default todoStore
```

app.js入口

添加store和provider

```
// app.js
import { Provider } from '@tarojs/mobx'
import todoStore from './store/todo'
const store = {
  todoStore
}


...

  render () {
    return (
      <Provider store={store}>
        <Index />
      </Provider>
    )
  }
```

Index.js

```
import Taro, { Component } from '@tarojs/taro'
import { View, Text,Input } from '@tarojs/components'

import { AtButton,AtInput,AtList, AtListItem } from 'taro-ui'
import './index.scss'

////**********Adding***************
```

```
import { observer, inject } from '@tarojs/mobx' // Adding
@inject('todoStore')  // Adding
@observer   // Adding
////**********Adding***************
export default class Index extends Component {

  config = {
    navigationBarTitleText: 'Todolist'
  }
  constructor(props){
    super(props)
    this.state={
      val:''
    }
  }
  handleInput = (val)=>{
    this.setState({val})
  }
  handleClick = ()=>{
    ////**********Adding***************
    this.props.todoStore.addTodo(this.state.val) // Adding
    this.setState({
      val:''
    })
    ////**********Adding***************
  }
  render () {
    ////**********Adding***************
    const {todoStore} = this.props
    ////**********Adding***************
    return (
      <View className='index'>
        <Text>开课吧Todo List</Text>
        <View className='at-icon at-icon-bullet-list'></View>

        <AtInput value={this.state.val} onChange={this.handleInput}></AtInput>
        <AtButton type='primary' onClick={this.handleClick}>添加</AtButton>
         ////**********Adding***************
        <AtList>
        {todoStore.todos.map((item,i)=>{
          return <AtListItem title={i+':'+item}>
          </AtListItem>
        })}
          ////**********Adding***************
        </AtList>
      </View>
    )
  }
}
```

删除功能

```
// src/store/todo.js
 ////**********Adding***************
import Taro from '@tarojs/taro'
////**********Adding***************
import { observable } from 'mobx'

const todoStore = observable({
  todos: ['吃饭','睡觉','开课吧学习'],

  addTodo(item) {
    this.todos.push(item)
  },

  ////**********Adding***************
  removeTodo(i){
    Taro.showLoading({
      title: 'loading'
    })
    setTimeout(()=>{
      this.todos.splice(i,1)
      Taro.hideLoading()
    },1000)
  }
  ////**********Adding***************
})

export default todoStore
```

```
  {todoStore.todos.map((item,i)=>{
    return <AtListItem
              title={i+':'+item}
              isSwitch
              onSwitchChange={()=>this.handleChange(i)}
              >
    </AtListItem>
  })}

  handleChange=(i)=>{
    this.props.todoStore.removeTodo(i)
  }
```

6. Taro原理解析

- 为什么选择React
  - React 是一个非常流行的框架，也有广大的受众，使用它也能降低小程序开发的学习成本；
  - React 采用 JSX 作为自身模板，JSX 相比字符串模板来说更自由，更自然，更具表现力，不需要依赖字符串模板的各种语法糖，也能完成复杂的处理
  - 小程序的数据驱动模板更新的思想与实现机制，与 React 类似；
  - React 本身有跨端的实现方案 - React Native，并且非常成熟，社区活跃，对于 Taro 来说有更多的多端开发可能性。
- 抹平多端差异
- 组件不同

小程序组件

组件	作用
view	容器组件
text	文字组件
image	图片组件
icon	图标组件
swiper	轮播组件
button	按钮组件
...	...

Web 组件

组件	作用
div	块级元素标签
span	行内元素标签
a	超链接标签
table	表格标签
ul	无序列表
p	段落标签
...	...

- API不同

小程序 API

API	作用
wx.request	网络请求
wx.uploadFile	上传文件
wx.downloadFile	下载文件
wx.login	微信登录
wx.getStorage	获取本地存储
wx.getStorage	设置本地存储

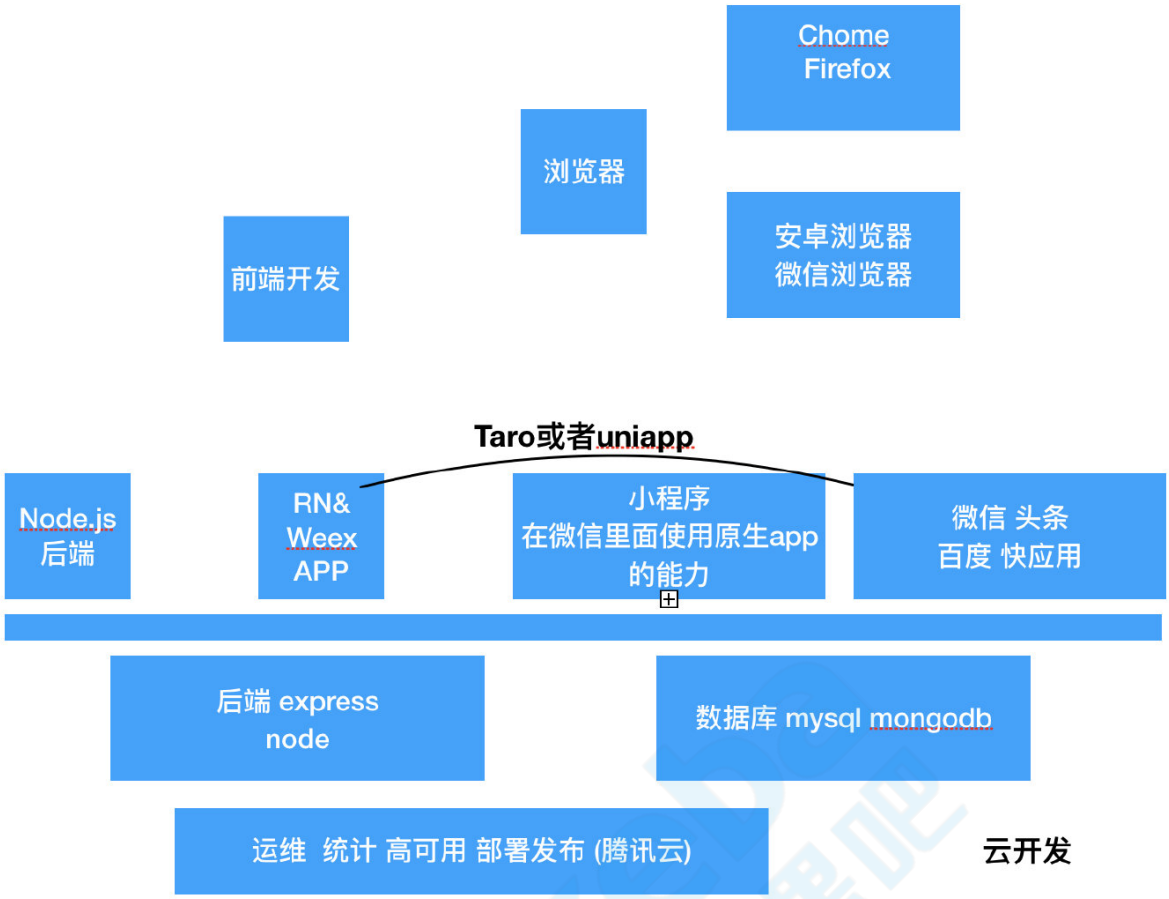
Web API

组件	作用
window.fetch	网络请求
window.localStorage	本地存储相关
window.open	打开新页面
window.scrollTo	页面滚动
window.navigator	浏览器相关信息
window.location	文档当前位置信息

- Taro框架组成

6. 扩展点

1. 扩展
    1. 云开发
2. 和别的章节联系
    1. 多端 云开发VS ajax
3. 知识体系
    1. 小程序多端



7. 总结

1. 回顾知识点
2. 提示学习方法
3. 提示本次课重点和必会知识

React+开发小程序

- 1. 课前准备
- 2. 回顾
- 3. 课堂主题
- 4. 课堂目标
- 5. 知识点
  - 为什么要有跨端的框架
    - 代码组织
    - 开发方式
    - 端太多了
  - 跨端
  - 2. Taro简介
  - 3. Taro初体验
  - 目录结构
  - 开发简单的todolist
  - 使用state
  - 添加用户输入
  - 使用UI组件

8. 作业 && 答疑

9. 下节课预告

全栈小程序实战