微信小程序开发技术分享

目前编写微信小程序的技术有如下：微信开发者工具自带的一套开发技术，基于腾讯的WePY框架，美团的Mpvue，京东的Taro，这四个开发小程序技术目前是应用最为广泛的。今天最主要介绍的是来自京东凹凸团队的Taro框架。

什么是Taro?

多端统一开发框架，支持用 React 的开发方式编写一次代码，生成能运行在微信小程序、H5、React Native 等的应用。

为什么要用Taro?

当今市面上产品端的形态多种多样，有传统的Web端、有可以运行在移动端App端(React Native,Weex,Ionic等)、还有微信小程序，支付宝小程序等各种五花八门端，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，而且不易于维护，这时候我们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（微信小程序、H5、App 端等）运行的代码（当然来自美团的Mpvue框架也可以）。有效地提升了开发效率和开发成本且易于维护。

Taro与其它框架进行对比

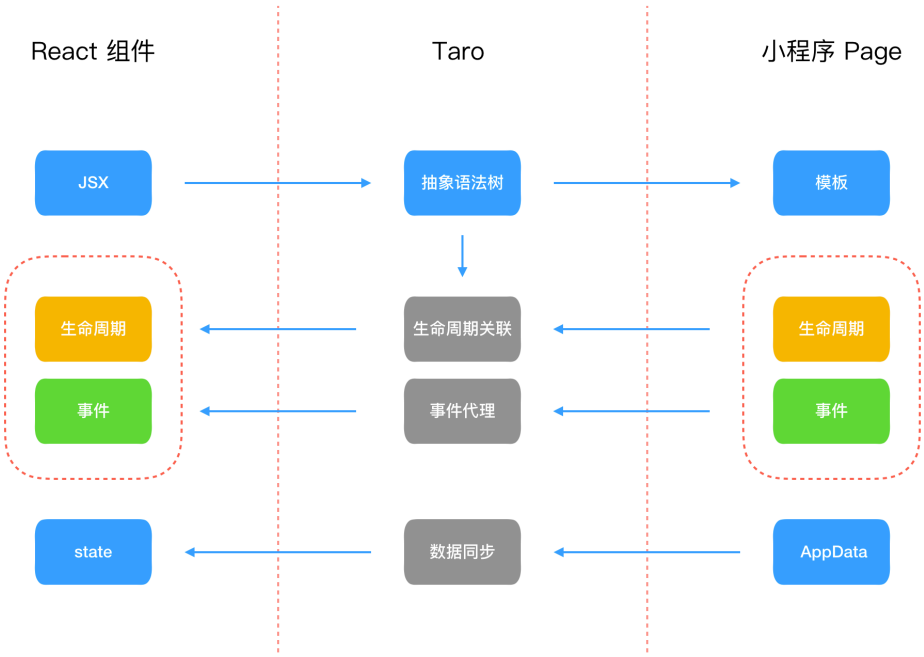
对比分析	微信小程序	mpuve	wepy	Taro
语法规范	小程序规范	Vue.js规范	类Vue.js规范	React规范

模板系统	字符串模板	字符串模板	字符串模板	JSX
类型系统	不支持	业务代码	业务代码	业务代码 + JSX
组件规范	小程序组件	HTML标签+小程序组件	小程序组件	小程序组件
样式规范	wxss	css,less.sass,styus	css,less.sass,styus	css,less.sass,styus
组件化	小程序组件化	Vue组件化	自定义组件化	React组件化
多端复用	复用，不存在的	H5,APP(Weex)等	没看到有介绍	H5,APP(ReactNative)等
自动构建	无	webpack	内部构建系统	webpack+内部构建系统
开发要求	全新学习	熟悉Vue语法即可	熟悉Vue+wepy语法	熟悉React语法
数据流管理	无	Vuex	Redux	Redux+React-thunk

> 从整体可行性，多端代码复用性上来说目前感觉Mpvue和Taro 更胜一筹。

Taro框架设计思想 具体细节请看官网

> 图片来源于网络



Taro特性

- React 语法风格
- 快速开发微信小程序
  - 支持使用 npm/yarn 安装管理第三方依赖
  - 支持使用 ES7/ES8 甚至更新的ES规范，一切都可自行配置
  - 支持使用 CSS 预编译器，例如 Sass 等
  - 支持使用 Redux进行状态管理
  - 小程序API优化，异步API Promise化等等
- 支持多端开发转化

开发配置

- 环境配置
  - sudo npm/yarn global add @tarojs/cli 也可以npx (npm5.2+)
  - npm init lavectorApp && cd lavctorApp && npm run dev:weapp/h5
  - 用微信开发者工具添加项目打开dist目录即可
- 页面设计尺寸配置
  - Taro默认是以iPhone6设计图作为默认尺寸 750 可以在config/index.js 中的修改

    - Taro项目中所有的px 最终转化小程序是以rpx作为单位，转为H5是以rem为单位
- 编辑器
    - 建议VSCode, Taro会有代码提示，支持ES6/7/8语法，ESlint等功能（这个随意，开心就好） *更多配置参见官网....

如何引用fontAwesome字体文件

比起图片，iconfont 更加容易控制其字体颜色，大小，减少HTTP请求等。 我已经做好了一个这个的文件（可自行百度，google）,点击查看 查看

如何使用：

```
import Taro, { Component } from '@tarojs/taro';
import { View, Icon, Image } from '@tarojs/components';
import 'your/path/fontawesome.css'
//引用图片
import img from 'your/path/image.png'
export default class Icon extends Taro.Component {
  ......
  render(){
      <View className='icon_box'>
        <Icon className='fa fa-home'/>
        <Image src={img} />
      </View>
  }
}
```

页面配置

```
//是不是跟小程序入口配置文件差不多?
config = {
// pages 配置 Taro中首页一定放在第一个（注重顺序），在Mpvue中首页一定要加^符号（不注重顺序）
    pages: [
      'pages/book/book',
      'pages/detail/detail',
      'pages/user/user',
      'pages/index/index'
    ],
    window: {
      backgroundTextStyle: 'light',
```

```
    navigationBarBackgroundColor: '#f55002',
    navigationBarTitleText: '首页',
    navigationBarTextStyle: 'light',
    enablePullDownRefresh: false,
    onReachBottomDistance: 30,
    backgroundColor: '#f55002'
  },
  tabBar: {
    backgroundColor: '#eee',
    selectedColor: '#f55002',
    color: '#aaa',
    borderStyle: '#f55002',
    list: [
      {
        pagePath: 'pages/index/index',
        iconPath: './static/images/home_default.png',
        selectedIconPath: './static/images/home_selected.png',
        text: '首页'
      },
      {
        pagePath: 'pages/book/book',
        iconPath: './static/images/book_default.png',
        selectedIconPath: './static/images/book_selected.png',
        text: '书架'
      },
      {
        pagePath: 'pages/user/user',
        text: '我的',
        iconPath: './static/images/user_default.png',
        selectedIconPath: './static/images/user_selected.png'
      }
    ]
  },
  networkTimeout: {
    request: 6000,
    downloadFile: 10000
  },
  debug: true
};
```

消息机制

> Taro 提供了 Taro.Events 来实现消息机制，使用时需要实例化它，看完文档感觉跟vue的global event bus很像 可

> ## 以用作非父子组件间的通讯，应用场景，看业务需求~

```
import Taro, {Events} from '@tarojs/taro'
const events = new Events()
// 监听一个事件，接受参数
events.on('eventName', (arg) => {
  // doSth
})
// 监听同个事件，同时绑定多个handler
events.on('eventName', handler1)
events.on('eventName', handler2)
events.on('eventName', handler3)
// 触发一个事件，传参
events.trigger('eventName', arg)
// 触发事件，传入多个参数
events.trigger('eventName', arg1, arg2, ...)
// 取消监听一个事件
events.off('eventName')
// 取消监听一个事件某个handler
events.off('eventName', handler1)
// 取消监听所有事件
events.off()
```

同时 Taro 还提供了一个全局消息中心 Taro.eventCenter 以供使用，它是 Taro.Events 的实例

```
import Taro from '@tarojs/taro'

Taro.eventCenter.on
Taro.eventCenter.trigger
Taro.eventCenter.off
```

组件和页面的定义

```
import Taro, { Component } from '@tarojs/taro';
import { View, Text, Icon, Image,Button } from '@tarojs/components';
//.........
//import Proptypes from 'prop-types'
// 可以引入prop-types 进行类型检查
export default class App extends Component {
 constructor(props) {
```

```
  super(props);
  this.state = {
    test: []
  };
}
//static propTypes = {
//    lavectorName:PropTypes.object.isRequired,
//      .......
//}
  //只有页面有的
  config = {
  <!--这里面的配置会覆盖app.js 里面window里的数据 跟小程序的app.json配置文件类似-->
  backgroundTextStyle: 'light',
  navigationBarBackgroundColor: '#f55002',
  navigationBarTitleText: '首页',
  navigationBarTextStyle: 'light',
  enablePullDownRefresh: false,
  onReachBottomDistance: 30,
  backgroundColor: '#f55002'
  }
  componentDidShow(){}
  componentDidHide(){}
  ......
  // 页面与组件都有的
  componentDidMount(){}
  componentWillReceiveProps(nextProps) {}
  componentWillUnmount() {}
  componentCatchError() {}
  ......
  //Mpvue生命周期函数多的你数不过来 大概有23个 可以查看 http://mpvue.com/mpvue/
  //自定义方法
  function toOtherPage () {
      // 跳转路由 这种方法不能跳转Tab页面, 如要跳转tab页面用Taro.switchTab({})
      Taro.navigateTo({
          url: 'https://lavector.com?timestmp=323233232323'
      })
      // 在目的页面获取路由参数通过 this.$router.params.timestmp

  }
  render(){
  <!--DOM 只能写在render函数中-->
      const test = `test`
      return (
      <!--不支持style={{...}} CSS in JS, 不支持{...props}这样写-->
      <View className='lavectorApp' style='background:#fff'>
```

```
      <Text>hello Lavector<Text>
      <Text>{test}</Text>
      // Taro中所有的事件都要以On 开头
      <Button onClick={this.toOtherPage.bind(this,'params')}>To Other
Page</Button>
      </View>
      );
  }
}
```

配合使用Redux

> 在 Taro 中可以自由地使用 React 生态中非常流行的数据流管理工具 Redux 来解决复杂项目的数据管理问题。而为了更方便地使用 Redux，Taro 提供了与 react-redux API 几乎一致的包 @tarojs/redux 来让开发人员获得更加良好的开发体验。首先要安装 redux,@tarojs/redux和@tarojs/redux-h5以及一些需要用到的redux中间件

npm install --save redux @tarojs/redux @tarojs/redux-h5 redux-thunk redux-logger

```
// src/store/index.js
import { createStore, applyMiddleware } from 'redux'
import thunkMiddleware from 'redux-thunk'
import { createLogger } from 'redux-logger'
import rootReducer from '../reducers'

const middlewares = [
  thunkMiddleware,
  createLogger()
]

export default function configStore () {
  const store = createStore(rootReducer, applyMiddleware(...middlewares))
  return store
}
```

入口文件app.js

```
// src/app.js
import Taro, { Component } from '@tarojs/taro'
import { Provider } from '@tarojs/redux'
```

```
import configStore from './store'
import Index from './pages/index'

import './app.scss'

const store = configStore()

class App extends Component {
  config = {
    pages: [
      'pages/index/index'
    ],
    window: {
      navigationBarTitleText: 'Test'
    }
  }

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

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

> 然后就可以开始使用了。如 redux 推荐的那样，可以增加constants 目录，用来放置所有的 action type 常量,actions 目录，用来放置所有的 actions,reducers 目录，用来放置所有的 reducers(具体可以查看taro生成的Vuex项目中的对应文件)

补充 JSX语法支持程度

- 不能在自定义组件中写 children（mpvue中的插槽功能slot也不能用，wepy应该支持类似功能）
- 不能在包含 JSX 元素的 map 循环中使用 if 表达式
- 不能使用 Array#map 之外的方法操作 JSX 数组
- 不能在 JSX 参数中使用匿名函数
- 暂不支持在 render() 之外的方法定义 JSX
- 不允许在 JSX 参数(props)中传入 JSX 元素
- 不能在 JSX 参数中使用对象展开符

- 不支持无状态组件

结束语

无论是打算学习Taro还是Mpvue，都应该先学习原生的微信小程序，然后在学习React/Vue进而再学习Taro和Vue,如果你精通React和Vue，感觉上手Mpvue和Taro很快，但是一开始用的不是很习惯。限制了Vue/React许多飘逸的写法与操作。

> 以上是本人粗鄙的见解，欢迎指正！（看到我手里的菜刀了吗,想好了再说！）

> Tips:更多用法请参考官网京东Taro

可能你需要学习的：

【Mpvue】(http://mpvue.com/)

【Mpvue资源合集】(https://github.com/mpvue/awesome-mpvue)

【Taro】(https://nervjs.github.io/taro/)

【Vuejs】(https://cn.vuejs.org/)

【ReactJS】(https://cn.vuejs.org/)

【Redux】(http://www.redux.org.cn/)

【Vuex】(http://www.redux.org.cn/)