

# 多端统一开发实践

蔡珉星 @ 趣店

# 目录

- 背景介绍
- 开源方案
- 实现原理
- 实践分享

# 多端挑战

➤ 小程序无法在其他端上运行

若多端独立开发，可能面临：

- 多套技术栈
- 多个代码仓库
- 维护成本高
- 未来更多的入口需求

# 多端统一开发意义

- ✓ 统一技术栈，减少学习成本
- ✓ 优化工作流，提升开发体验
- ✓ 降低开发、维护成本

立足小程序

能否做到 “Write Once, Run Everywhere” ？

# 小程序约束

代码构成:

- JSON 配置文件
- WXML 模板文件
- WXSS 样式文件
- JS 脚本逻辑文件

# 小程序开发框架

Vue-like

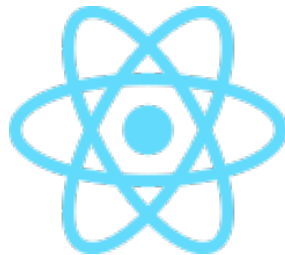


mpvue



wepy

React-like



taro

# Vue-like、React-like 代码，如何在小程序中运行？

## React-like 代码

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

export default class extends Component {
  getUser = () => {
    // ....
  }

  render() {
    return (
      <View className="container">
        <Text className="user-name">{userName}</Text>
        <Button onClick={this.getUser}>获取用户信息</Button>
      </View>
    )
  }
}
```

## 小程序代码

```
// .js
Page({
  data: {
    userName: '',
  },
  getUser: function() {
    // ...
  }
})

// .wxml
<view class="container">
  <text class="user-name">{{userName}}</text>
  <button bindtap="getUser">获取用户信息</button>
</view>
```



# 基本原理



编译时处理（转译成小程序语法）

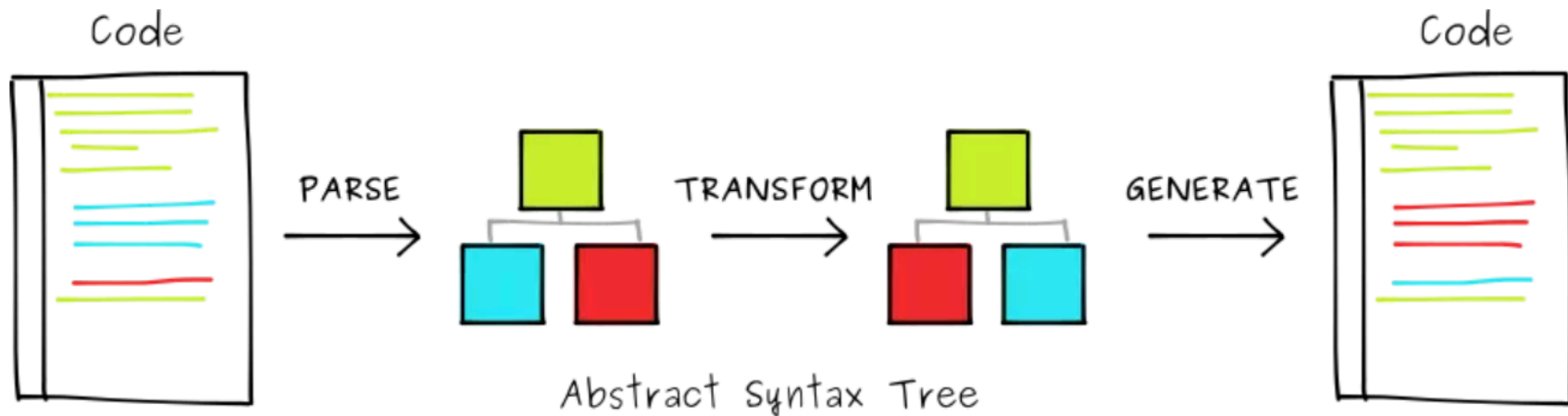


运行时适配（管理生命周期、数据，处理事件等）

# 编译时处理

## ➤ 编译原理

源代码->词法/语法/语义分析->抽象语法树->转换->目标代码



# 抽象语法树 AST

- JS 的 AST 规范: ESTree Spec([github.com/estree/estree](https://github.com/estree/estree))
- 描述语法结构, 如 Functions、Statements、Declarations .....

## AST Descriptor Syntax

```
extend interface Program {  
  sourceType: "script" | "module";  
  body: [ Statement | ModuleDeclaration ];  
}
```

# AST 示例

## ● ● ● JSX代码

```
<View className="user">
  <Text>{name}</Text>
</View>
```

借助 [astexplorer.net](https://astexplorer.net) 查看 AST

```
- Program {
  type: "Program"
- body: [
  - ExpressionStatement {
    type: "ExpressionStatement"
  - expression: JSXElement {
    type: "JSXElement"
    + openingElement: JSXOpeningElement {type, attributes, name,
    + closingElement: JSXClosingElement {type, name}
  - children: [
    - JSXElement = $node {
      type: "JSXElement"
      + openingElement: JSXOpeningElement {type, attributes,
        selfClosing}
      + closingElement: JSXClosingElement {type, name}
      + children: [1 element]
    }
  ]
}
}
]
sourceType: "module"
}
```

# AST 相关学习内容

**THE  
SUPER  
TINY  
COMPILER.**

[github.com/jamiebuilds/the-super-tiny-compiler](https://github.com/jamiebuilds/the-super-tiny-compiler)

[www.youtube.com/watch?v=Tar4WgAfMr4](https://www.youtube.com/watch?v=Tar4WgAfMr4)

# AST 相关工具



babel-core	解析代码获取 AST
babel-types	AST 节点定义、生成节点
babel-traverse	递归遍历操作 AST
babel-generator	AST 生成源码

# 小程序框架多端支持情况

框架	H5	Native	支付宝小程序、快应用等
wepy	支持	No	支持支付宝小程序
mpvue	支持	No	No
taro	支持	支持 RN	支持百度/支付宝小程序

(以是否具备基本的编译、运行能力为准)



**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。

# Taro 原理

package.json

```
"scripts": {  
  "build:weapp": "taro build --type weapp",  
  "build:h5": "taro build --type h5",  
  "build:rn": "taro build --type rn",  
  "dev:weapp": "npm run build:weapp -- --watch",  
  "dev:h5": "npm run build:h5 -- --watch",  
  "dev:rn": "npm run build:rn -- --watch"  
}
```

通过 Taro CLI 管理、构建项目



# Taro 原理

## build weapp

```
async function build ({ watch, adapter }) {  
  // ...  
  buildProjectConfig()           // 项目配置文件 project.config.json  
  appConfig = await buildEntry() // 编译项目入口页  
  await buildPages()             // 编译 page 页面  
  // ...  
}
```

# Taro 原理

## 小程序页面编译过程

```
async function buildSinglePage (page) {  
  // 读取代码文件  
  const pageJsContent = fs.readFileSync(pageJs).toString()  
  // 进行转换  
  const transformResult = wxTransformer({  
    code: pageJsContent,  
    // ...  
  })  
  // 进一步处理, 如编译依赖的组件文件  
  const res = parseAst(transformResult, ...)  
  let resCode = await compileScriptFile(res.code)  
  // ... 最终输出编译后的文件, 包含 .js, .wxml, .wxss, .json  
  fs.writeFileSync(outputPageJSONPath, ...)  
  fs.writeFileSync(outputPageWXMLPath, transformResult.template)  
  await compileDepStyles(outputPageWXSSPath, res.styleFiles, false)  
  copyFilesFromSrcToOutput(res.jsonFiles)  
  // ...  
}
```

Babel 构建 AST, 进行 traverse 操作

# Taro 原理

➤ 多端能力：提供对应端的runtime、组件、redux、router等实现

```
└─ packages
  ├── taro
  ├── taro-alipay
  ├── taro-components
  ├── taro-components-rn
  ├── taro-h5
  ├── taro-redux
  ├── taro-redux-h5
  ├── taro-redux-rn
  ├── taro-rn
  ├── taro-rn-runner
  ├── taro-router
  ├── taro-router-rn
  └── taro-weapp
```

- runtime

- taro-weapp
- taro-h5
- taro-alipay

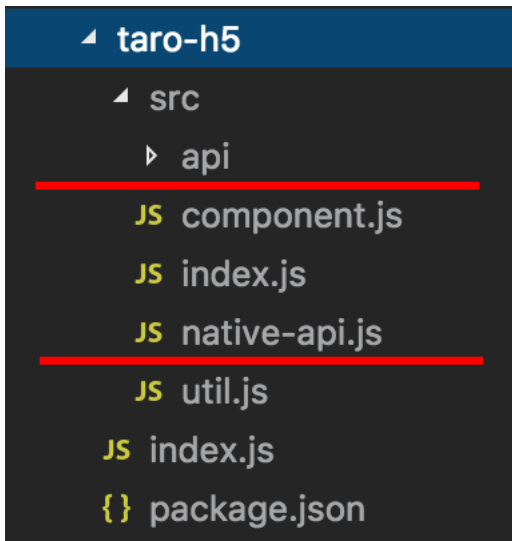
- 组件

- taro-components
- taro-components-rn

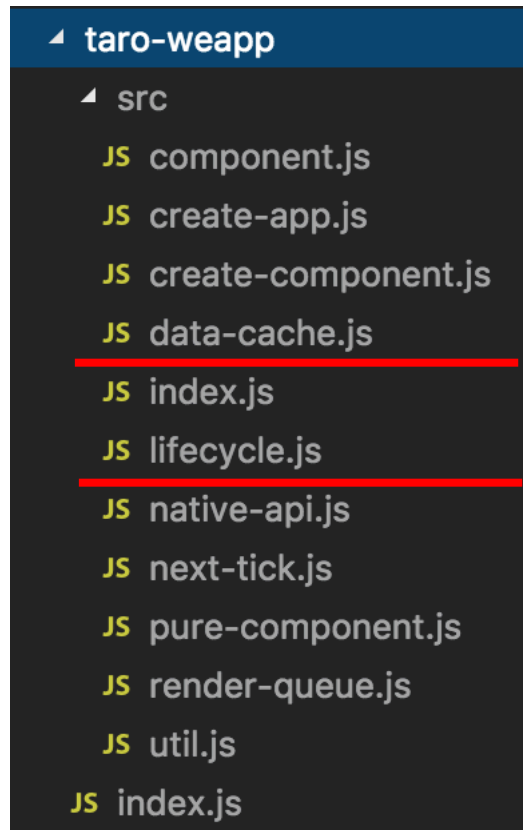
- redux

- taro-redux
- taro-redux-h5
- taro-redux-rn

# Taro 原理 - Runtime



H5 主要提供端能力API



小程序端需要处理生命周期等

# Taro 原理 - Components

## Slider 组件 - H5

```
class Slider extends Nerv.Component {  
  // ...  
  render() {  
    return  
      <div className={cls}>  
        <div className='weui-slider'>  
          <div className='weui-slider__inner' style={innerStyles}>  
            // ...  
          </div>  
        </div>  
      </div>  
  }  
}
```

小程序端调用的是原生标签，而 H5 端则是基于 Dom 实现

# Taro 原理 - Components

## Slider 组件 - React Native

```
import * as React from 'react'
import {
  View,
  Text,
  Slider,
  StyleSheet,
} from 'react-native'
import styles from './styles'

// ...
```

React Native 端则是基于 RN 原生组件实现

# Taro 原理

- 提供相应端的 Runtime 实现，具备适配 API 的能力
- 提供统一的基础组件，编译时替换为相应端的组件实现

Taro 实际使用如何，是否有坑？



# 坑在哪

React-like, 不完全等同 React

## Taro 新人常犯的错误

```
export class extends Nerv.Component {  
  render () {  
    return (  
      <View>  
        <Comp {...props} /> // 不支持展开操作符  
        {this.renderList()} // JSX 暂时只能写在 render 中  
      </View>  
    )  
  }  
}
```

小程序中受限于 WXML 语法

# 坑在哪

## 端能力差异不可避免：某些功能在相应端上没有支持

- 例如小程序的录音功能在其他端上没有

### Taro.startRecord(OBJECT)

使用方式同 `wx.startRecord`，支持 `Promise` 化使用。

### Taro.stopRecord()

主动调用停止录音。

#### API 支持度

API	微信小程序	H5	ReactNative
Taro.startRecord	✓		
Taro.stopRecord	✓		

# 坑在哪

## 样式支持、写法上存在差异

### 小程序 vs Web:

- 第三方 UI 组件库缺少自定义主题功能
- 自定义组件对样式支持不够好
  - 不支持 ID 选择器、属性选择器、标签名选择器
  - 全局样式需基础库版本 2.2.3 以上，目前还有 10%+ 用户不满足
  - 外部样式类 `externalClasses` 通常只支持组件最外层元素

# 坑在哪

## 样式支持、写法上存在差异

### React Native vs Web:

- 实现了 CSS 子集，支持有限
- 不支持伪类、不支持设置 background-image、!important 等
- 部分样式默认值不同，如 flex 布局的 flex-direction
- .....

# 坑在哪

## React Native 代码

```
import React, { Component } from 'react'
import { StyleSheet, View, Text } from 'react-native'
```

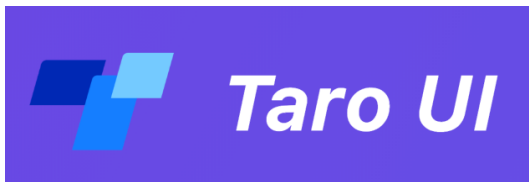
```
const styles = StyleSheet.create({
  red: {
    color: 'red',
  }
})
```

```
export default class extends Component {
  render() {
    return (
      <View>
        <Text style={styles.red}>just red</Text>
      </View>
    )
  }
}
```

样式只支持对象传入  
无 className

# 坑在哪


还没有真正支持多端的 UI 组件库能在 Taro 上使用



- 基于 Taro 框架开发的多端 UI 组件
- 暂不支持 React Native

# 坑在哪

- React Native 的 view 不支持 click 事件，需要用 Touchable 组件



```
<TouchableOpacity onPress={this.handlePress}>  
  <View>{...}</View>  
</TouchableOpacity>
```

Taro 的基础组件还没有考虑这个情况，

而 Taro UI 的 Button 组件的点击事件加在外层 View 上，自然没法用

# 坑在哪

- React Native 不支持 text-overflow, 而是提供原生支持 (Text 组件传入 numberOfLines={num} 属性)

Taro 目前没有暴露原生 RN Text 组件的 numberOfLines 属性  
因此目前通过 Text 基础组件没法实现多端统一的文本截断



# H5情况较为乐观

- 主要差异1: 小程序的页面、组件样式都是独立的, H5会受同名样式影响
- 主要差异2: 小程序的组件多了一层标签

```
▼ <div class="q-tag--list">  
  ▼ <div class="q-tag"> == $0  
    <span class="taro-text">语文</span>  
  </div>  
  ▶ <div class="q-tag q-tag_checked">...</div>  
  ▶ <div class="q-tag q-tag_plain">...</div>  
  ...  
</div>  
,  
.q-tag--list .q-tag {  
  display: inline-block;  
  vertical-align: middle;  
  margin-left: 0.42667rem;  
  margin-bottom: 0.42667rem;  
}
```

H5 的标签、层级

```
▼ <view class="q-tag--list">  
  ▼ <tag is="components/qui/tag/index">  
    #shadow-root  
    ▼ <view bindtap="func__mCjQX" class="q-tag">  
      <text>...</text>  
    </view>  
  </tag>  
  ...  
</view>  
,  
.q-tag--list tag {  
  display: inline-block;  
  vertical-align: middle;  
  margin-left: 20rpx;  
  margin-bottom: 20rpx;  
}
```

小程序的标签、层级 (多了一层)

# 多端要求较高

- 对不同端的具体差异有所了解
- 样式实现较为苛刻，需兼顾多端、有所取舍
- 端能力差异可能需要自己填坑

# 多端填坑方式

```
class default class extends Component {  
  handleClick() {  
    if (process.env.TARO_ENV === 'weapp') {  
      // ...  
    } else if (process.env.TARO_ENV === 'rn') {  
      // ...  
    }  
  }  
}  
  
render() {  
  return (  
    <View>  
      {process.env.TARO_ENV === 'weapp' && <WeNativeComp />}  
      {process.env.TARO_ENV === 'rn' && <RnNativeComp />}  
    </View>  
  )  
}
```

Taro 支持环境判断  
编译时只保留相应端代码

Taro 支持引入相应端的原生代码

# 多端实践经验

- 对多端差异做好封装
- 采用 BEM 命名方式管理样式
- 采用全局样式维护基础组件 @taro/components 的样式
- 基于 @taro/components 自行封装组件库
- .....

# 回顾

- 多端统一开发意义
- 开源框架介绍
- Vue-like、React-like 代码如何在小程序上运行
- Taro 的多端原理
- Taro 的多端局限及实践经验

# 总结

## ➤ Taro 值得关注、尝试

- ✓ 优秀的小程序开发框架
- ✓ 目前对多端编译支持最好
- ✓ 保持较快的迭代节奏
- ✓ 提供友好的技术支持

Thanks

Q&A