React-Native开发规范

`React-Native` `JavaScript`

版本号	制定团队	更新日期	备注
0.0.4	信雅达	2017-08-29	版本更新

2017-08-29
更新记录：
1-：更新页面编写规范
2-：更新编码约定
3-：更新自定义组件

一、编程规约

(一) 命名规约

1. 【强制】 代码中命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束；

   `_name / $Object / name_ / name$ / Object$`

2. 【强制】 代码中命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式； 说明:正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式也要避免采用；

   `反例: DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3`

3. 【强制】类名使用 UpperCamelCase 风格，必须遵从驼峰形式，第一个字母必须大写；

   `LoginPage/MsgPage`

4. 【强制】方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase风格，必须遵从驼峰形式，第一个字母必须小写；

   `localValue / getHttpMessage() / inputUserId`

5. 【强制】常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长；

   `正例: MAX_STOCK_COUNT`

   `反例: MAX_COUNT`

6. 【强制】使用抽象单词命名类名或者变量时，需加以修饰；

   `userMsg 等价于 userMessaage,`

   `userPic 等价于 userPicture`

7. 【强制】中括号是数组类型的一部分，数组定义如下:String[] args;

反例:请勿使用String args[]的方式来定义。

8. 【强制】包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。包名统一使用单数形式，但是类名如果有复数含义，类名可以使用复数形式；
   `正例：应用工具类包名为com.fcs.open.util、类名为UrlUtils`

9. 【强制】文件夹命名统一小写； 组件，或者类名,首字母全部大写，遵守驼峰命名法；

```
img     存放图片
app     APP一些component
artcomponent     一些art组件
```

(二) 常量定义

1. 【强制】不允许出现任何魔法值(即未经定义的常量)直接出现在代码中；
2. 【推荐】不要使用一个常量类维护所有常量，应该按常量功能进行归类，分开维护。
   如:缓存相关的常量放在类:CacheConsts下;
   系统配置相关的常量放在类:ConfigConsts下； 说明:大而全的常量类，非得使用查找功能才能定位到修改的常量，不利于理解和维护；

(三) 格式规约

1. 【强制】大括号的使用约定。如果是大括号内为空，则简洁地写成{}即可，不需要换行;如果 是非空代码块则:
   1) 左大括号前不换行；
   2) 左大括号后换行；
   3) 右大括号前换行；
   4) 右大括号后还有else等代码则不换行 ‘；’ 表示终止右大括号后必须换行。
2. 【强制】 左括号和后一个字符之间不出现空格;同样，右括号和前一个字符之间也不出现空格；

3. 【强制】if/for/while/switch/do 等保留字与左右括号之间都必须加空格；
4. 【强制】任何运算符左右必须加一个空格；
   说明:运算符包括赋值运算符=、逻辑运算符&&、加减乘除符号、三目运行符等；
5. 【强制】缩进采用 4 个空格，禁止使用 tab 字符；
6. 【强制】单行字符数限制不超过120个，超出需要换行，换行时遵循如下原则:
   1) 第二行相对第一行缩进4个空格，从第三行开始，不再继续缩进，参考示例；
   2) 运算符与下文一起换行；
   3) 方法调用的点符号与下文一起换行；
   4) 在多个参数超长，逗号后进行换行；

```
const path = Path()
        .moveTo(0, -radius/2)
        .arc(0, radius, 1)
        .arc(0, -radius, 1)
        .close();
```

7. 【强制】方法参数在定义和传入时，多个参数逗号后边必须加空格；

```
onMsgByCallAndMsg=(msg, title, type)=>{
    this.setState({
        callMsgAndMsg:msg
    })
}
```

8. 【推荐】方法体内的执行语句组、变量的定义语句组、不同的业务逻辑之间或者不同的语义之间插入一个空行。相同业务逻辑和语义之间不需要插入空行。
   说明:没有必要插入多行空格进行隔开。
9. 【推荐】使用webStomr时，导入附件的hoop-settings-1.0.jar文件，可统一格式化。

(四) package.json

1. 【强制】在使用npm或者yarn获取资源时，必须在命令末尾添加--save；
   说明：使用此命令会把使用的第三方相关信息写入到package.json，这样，其他成员在下载或者更新代码后使用npm i，就可以下载最新的npm，若不加 —save ，执行npm i的时候不会下载，其他成员运行项目后在运行可能会报错，此时需要分析查看报错信息进行重

新的npm install XX；

2. 【推荐】使用git或者svn进行代码版本管理时，尽量不上传node_module文件；
   说明：使用package.json进行包管理，下载或更新代码后，只需要执行npm i；当有修改npm包，建议进行版本管理，上传到私有的github仓库。

3. 【强制】使用第三方或拉取新仓库时，第一步使用npm i或者npm install；
   说明：检查版本是否存在冲突

4. 【推荐】在使用npm或者yarn获取资源时，推荐不在命令后添加 -g；
   说明，此命令可以让此资源包在根目录进行获取，不利于资源管理；

5. 【强制】当升级或降级react-native版本时，必须进行代码备份；
   说明：升级失败或者涉及到原生代码时，可以进行代码回滚

6. 【强制】每个项目必须配置一个readMe文件，内容包括测试，正式环境等相关配置文件以及注意事项；

7. 【推荐】安装npm包是，推荐~来标记版本号；
   说明：~和^的作用和区别：~会匹配最近的小版本依赖包，比如~1.2.3会匹配所有1.2.x版本，但是不包括1.3.0
   ^会匹配最新的大版本依赖包，比如^1.2.3会匹配所有1.x.x的包，包括1.3.0，但是不包括2.0.0。那么该如何选择呢？当然你可以指定特定的版本号，直接写1.2.3，前面什么前缀都没有，这样固然没问题，但是如果依赖包发布新版本修复了一些小bug，那么需要手动修改package.json文件；~和^则可以解决这个问题。但是需要注意^版本更新可能比较大，会造成项目代码错误，旧版本可能和新版本存在部分代码不兼容。所以推荐使用~来标记版本号，这样可以保证项目不会出现大的问题，也能保证包中的小bug可以得到修复。

(五) 控制语句

1. 【强制】在一个 switch 块内，每个case要么通过 break/return 等来终止，要么注释说明程序将继续执行到哪一个 case 为止;在一个 switch 块内，都必须包含一个default 语句并且 放在最后，即使它什么代码也没有。

2. 【强制】在 if/else/for/while/do 语句中必须使用大括号，即使只有一行代码，避免使用下面的形式:

```
if (condition) statements;
```

3. 【推荐】推荐尽量少用 else， if-else 的方式可以改写成:

```
if(condition){
                ...
return obj; }
```

// 接着写 else 的业务逻辑代码;
说明:如果非得使用

```
if()
...
else if(

)...else...
```

方式表达逻辑，【强制】请勿超过3层，
超过请使用状态设计模式。
正例:逻辑上超过 3 层的 if-else 代码可以使用卫语句，或者状态模式来实现。

4.【推荐】使用三目运算，替换if else结构，精简代码

```
let   account=5;
if(account>10){
        console.log("true");
}else {
        console.log("false");
}

let msg=account>10?"true":"false";
```

5.【推荐】除常用方法(如 getXxx/isXxx)等外，不要在条件判断中执行其它复杂的语句，将复杂逻辑判断的结果赋值给一个有意义的布尔变量名，以提高可读性。
说明:很多 if 语句内的逻辑相当复杂，阅读者需要分析条件表达式的最终结果，才能明确什么样的条件执行什么样的语句，那么，如果阅读者分析逻辑表达式错误呢?

```
        //伪代码如下
boolean existed = (file.open(fileName, "w") != null)&& (...) || (...);
```

```
        if (existed) {
            ...
         }
```

(六) 注释规约

1. 【强制】类、类属性、类方法的注释必须使用 Javadoc 规范，使用/**内容*/格式，不得使用 //xxx 方式；
   说明:在 IDE 编辑窗口中，Javadoc 方式会提示相关注释，生成 Javadoc 可以正确输出相应注释;在 IDE 中，工程调用方法时，不进入方法即可悬浮提示方法、参数、返回值的意义，提高阅读效率。
2. 【强制】所有的类都必须添加创建者信息，以及类的说明；
3. 【强制】方法内部单行注释，在被注释语句上方另起一行，使用//注释；
   方法内部多行注释使用/* */注释，注意与代码对齐。
4. 【强制】所有的常量类型字段必须要有注释，说明每个值的用途；
5. 【参考】注释掉的代码尽量要配合说明，而不是简单的注释掉。
   说明:代码被注释掉有两种可能性:
   1)后续会恢复此段代码逻辑。
   2)永久不用。前者如果没有备注信息，难以知晓注释动机。后者建议直接删掉(代码仓库保存了历史代码)。
6. 【参考】对于注释的要求:
   第一、能够准确反应设计思想和代码逻辑;
   第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同 天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路;注释也是给继任者看的，使其能够快速接替自己的工作。
7. 【参考】好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的一个极端:过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。
8. 【参考】特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描，经常清理此类标记。
   1) 待办事宜(TODO):(标记人，标记时间，[预计处理时间]) 表示需要实现，但目前还未实现的功能。

2) 错误，不能工作:(标记人，标记时间，[预计处理时间])
在注释中用 FIXME标记某代码是错误的，而且不能工作，需要及时纠正的情况。

(七) 日志管理

1. 【推荐】 代码中过多使用console.log()会消耗性能，推荐去除不必要的日志输入代码；
2. 【强制】 在入口文件添加以下代码；
   说明：可以在发布时屏蔽掉所有的console.*调用。React Native中有一个全局变量**DEV**用于指示当前运行环境是否是开发环境。我们可以据此在正式环境中替换掉系统原先的console实现。

```
if (!__DEV__) {
  global.console = {
    info: () => {},
    log: () => {},
    warn: () => {},
    error: () => {},
  };
}
```

   这样在打包发布时，所有的控制台语句就会被自动替换为空函数，而在调试时它们仍然会被正常调用。

(八) 目录结构规范

```
    以下目录结构示例中只展示js与静态资源，不包含原生代码：
├── index.ios.js
├── index.android.js
└── js
    ├── component           可复用的组件（非完整页面）
    ├── page                完整页面
    ├── config              配置项（常量、接口地址、路由、多语言化等预置数据）
    ├── util                工具类（非UI组件）
    ├── style               全局样式
    └── image               图片
在component和page目录中，可能还有一些内聚度较高的模块再建目录
```

```
page/component
├── HomeView.component.js
├── HomeView.style.js
└── MovieView
    ├── MovieList.component.js
    ├── MovieList.style.js
    ├── MovieCell.component.js
    ├── MovieCell.style.js
    ├── MovieView.component.js
    └── MovieView.style.js
```

二、页面编写规范

(一) state,props

1. 【强制】 代码中初始化state因在constructor(props)函数中，而且尽量对每个变量进行注释；
2. 【强制】 代码中使用setState时，因注意异步可能导致的问题，尽量使用回调函数；

```
this.setState({
       //todo
    },()=>{
       //执行setState后执行此函数
    })
```

3. 【强制】 代码中使用props时，需进行propTypes检测和defaultProps默认值初始化；

```
static propTypes = {
        color: PropTypes.string,
        dotRadius: PropTypes.number,
        size: PropTypes.number
    };

static defaultProps = {
        color: '#1e90ff',
        dotRadius: 10,
        size: 40
    };
```

4.【强制】 代码中用于页面展示处理UI的组件，命名以Page结尾，自定义组件命名中必须包含Component；
例子：

```
LoginPage            登录页
BtuuonComponent      按钮组件
```

5.【强制】代码中创建数组或对象使用以下方式；

```
const user={
        name:'time',
        sex:'男',
        age:25,
    };

const itemArray=['0','1','2',3,{name:'25',age:'男'}];
```

6.【强制】代码中函数绑定this，强制使用箭头函数；
注：除组件原有方法，其他自定义函数命名时，需使用箭头函数；

```
//系统组件生命周期方法
constructor(props){
    super(props);
};
//自定义方法
goMainPage=()=>{

};
```

7.【推荐】代码中一些网络数据初始化，配置信息，推荐在此生命周期进行初始化；

```
componentWillMount
```

8.【强制】代码中使用定时器或者DeviceEventEmitter，必须在组件卸载进行销毁或者清除；

```
componentDidMount() {
    //注意addListener的key和emit的key保持一致
    this.msgListener = DeviceEventEmitter.addListener('Msg',(listenerMsg) =
```

```
> {
        this.setState({
            listenerMsg:listenerMsg,
        })
    });
}

goMainPage=()=>{
    this.timer = setTimeout(
        () => { console.log('把一个定时器的引用挂在this上'); },
        500
    );
};


componentWillUnmount() {
    //此生命周期内，去掉监听和定时器
    this.msgListener&&this.msgListener.remove();
    // 如果存在this.timer，则使用clearTimeout清空。
    // 如果你使用多个timer，那么用多个变量，或者用个数组来保存引用，然后逐个clear
    this.timer && clearTimeout(this.timer);
}
```

9.【强制】使用本地图片资源时，需设置宽高并进行适当适配；

```
 imgHeight=screenHeight, imgWidth= screenWidth
```

10.【强制】在React-Native版本小于0.46.0使用本地图片资源时，当不指定特殊尺寸图片时，需引入不同尺寸XX.png,XX2@.png,XX3@.png图片，并在代码引用中，使用如下方式：

```
<Image style={{flex: 1, height: screenHeight, width: screenWidth}}
      source={require('../XX.png')}>
```

说明：当使用XX.png时，程序运行过程中会根据不同屏幕尺寸获取不同资源；当使用如下方式：

```
<Image style={{flex: 1, height: screenHeight, width: screenWidth}}
           source={require('../XX2@.png')}>
```

时，程序运行过程中不会根据不同屏幕尺寸获取不同资源。

注意：此方式适用于React-Native0.46.0版本之前。

9.【强制】在React-Native版本大于0.46.0使用本地资源，图片命名不能出现‘@’符号：
说明：不同大小图片需要原生不同的尺寸文件夹，系统自动进行不同适配。

(二) 样式

1. 【强制】 当组件使用样式属性达到三个或者三个以上时，必须使用StyleSheet来创建样式属性并进行引用；

```
const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
        backgroundColor: '#F5FCFF',
        marginTop:10,
    },
});
```

2. 【推荐】 当使用单一属性，或者全局样式属性时，推荐使用公共样式类；

```
//StyleCommon.js
module.exports={
    topColor:{
        backgroundColor: '#3A3D42',
    },
    mainView:{
        backgroundColor: '#12141B',
    },
}
```

3. 【推荐】 当使用多个state或者props值时，推荐使用以下方式；

```
 const {size, dotRadius, color} = this.props;
 const { maxNumber,minNumber,}=this.state;
```

说明使用此方式，代码结构清晰简洁，便于维护；

(三) var,let,const

1. 【强制】对所有变量，对象的引用，使用const,不要使用var;
2. 【推荐】如果一定需要引用可变动的变量，对象，建议使用let代替var;

(四) 代码间隔

1. 【强制】使用ES6编写代码，定义方法时，每个方法结尾使用 ‘;’ 进行分隔；

(五) 其他

1. 【强制】对组件引用，变量引用，需遵从以下方式;

```
import React, {Component} from 'react';
import{
          View,
          Text,
          TouchableHighlight,
          Image,
          StyleSheet,
          InteractionManager,
} from 'react-native';
//from react,react-native优先；
//from npm库其次；
import { connect } from 'react-redux';

//from 项目内组件其次；
import LoadingAndTime from '../component/LoadingAndTime';
import { performLoginAction } from '../action/LoginAction'
import {encode} from '../common/Base64';

//变量初始化，常量初始化 最后；
let screenWidth = Dimensions.get('window').width;
let screenHeight = Dimensions.get('window').height;
let typeCode = Platform.OS == 'android' ? 'android-phone' : 'ios-phone'
let selectColor=Platform.OS=='android'?null:'white'
```

2. 【推荐】对组件引用，变量初始化等，在整个页面或组件内未使用，因去除相关代码;
3. 【推荐】某些全局变量请不要使用global，需新建文件进行导出引用；

   `NetUtil.get(global.url + “”)`
4. 【推荐】render()函数代码过长时，请适当进行拆分，拆分为 ‘ ‘页面内组件 ‘ ‘，提高

可读性。render()函数代码行请勿超过八十行，超过之后，请自行进行拆分；

三、编码约定

(一)入口文件

1.【推荐】统一入口文件为App.js;
说明：在index.android.js和index.ios.js文件中，统一入口文件为App.js,且保持所在目录和index.android.js和index.ios.js同级。

2.【强制】开发中，不要使用任何后端的开发模式来构建APP结构，如使用MVC,MVP,MVVM等开发模式，React-Native推荐组件化，颗粒化，以上设计模式严重违背。若使用Redux,Mobx等数据流第三方，可依据第三方结构编写构建App。

3.【推荐】某些输入框的值，放入到state中，并且设置defaultValue，不要使用全局变量进行引用,参照以下方式：

```
 constructor(props) {
    super(props);
    this.state = {
        editSalesPrice:'',  //修改后的商品售价
        editPurchasePrice:'', //修改后的商品进价
    };
}

render(){
  return(
  <View style={styles.viewPadding}>
                        <TextInput
                             style={styles.rowInput}
                             placeholder="请输入调整后的价格"
                             onChangeText={(text)=>{
                                 this.setState({
                                     editSalesPrice:text,
                                 })
                             }}
                             defaultValue={this.state.editSalesPrice}
```

```
                            placeholderTextColor='#B0B7C2'
                            underlineColorAndroid = 'transparent'
                            autoCapitalize={'none'}
                            autoCorrect={false}
                            clearButtonMode={'while-editing'}
                            keyboardType='numeric'
                        />
                    </View>
                    );
}
```

4.【强制】移除定时器，监听请按照如下代码进编写；

```
componentWillUnmount() {
    //此生命周期内，去掉监听和定时器
    this.msgListener && this.msgListener.remove();
    // 如果存在this.timer，则使用clearTimeout清空。
    this.timer && clearTimeout(this.timer);
}
```

(二) 模版文件

1.【推荐】根据附件，配置代码编写模版，推荐使用第二种配置方式，可配置多种模版。

(三) ListView,FaltList

1.【强制】使用ListView或者FaltList的renderRow时，需对renderRow里面的组件需进行抽取，使用一个单独组件进行包裹，类似于页面子组件方式引入；
请勿使用如下方式：

```
renderRow(news) {
   return (
        <TouchableOpacity onPress={() => {
            this.onPress(news)
        }} style={styles.container}>
            <Image source={require('../imgs/icon_data.png')}/>
            <View style={styles.row_main}>
                <Text style={styles.row_title}>{news.belOutlet}</Text>
```

```
                <Text style={styles.row_bottom}>所属支行:{news.belBranch}</Te
xt>
            </View>
            {this._renderNewFlag(news)}
        </TouchableOpacity>
     );
}
```

推荐使用如下方式：

```
import GoodInCell from './GoodInCell';
renderRow(news) {
    return (
        <GoodInCell  news={news}  />
    );
}
```

说明：使用此方式，可增加代码的可读性和理解性。更符合组件化的开发思路。

四、自定义组件

(一) 自定义组件

1. 【强制】组件命名中必须包含Component;

说明：

```
ButtonComponent.js
LabelComponent.js
```

1. 【强制】组件中定义的state和props必须都要有注释，依次说明每个值的含义;
2. 【强制】在每个类的头部注释中，必须使用/**/说明此组件的基础使用方式以及特殊使用方法；

(二) 属性判断

1. 【强制】代码中使用props时，需进行propTypes检测和defaultProps默认值初始化；

```
static propTypes = {
```

```
        color: PropTypes.string,
        dotRadius: PropTypes.number,
        size: PropTypes.number
    };

static defaultProps = {
        color: '#1e90ff',
        dotRadius: 10,
        size: 40
  };
```

2.【强制】代码中使用props方法时，具体参照以下方式进行调用；

```
    export default class PostCallMsgToPar extends  Component {
        render() {
            return (
            <View style={styles.container}>
                <TouchableOpacity onPress={this.postMsgByCallBack}>
                <Text>使用Callback修改父状态，无返回值</Text>
                </TouchableOpacity>
            </View>
            );
        }

    postMsgByCallBack=()=>{
            if(this.props.onChangeMsg){
                this.props.onChangeMsg();
            }
        }
    }
```

(三) 性能优化

1. 【强制】无状态组件需使用PureComponent而不是Component；

说明：无状态组件是指内部没有使用state的组件，但是可以使用props来进行某些属性控制;

```
export default class LinkButton extends PureComponent {
    static  defaultProps= {
        msgName:'请输入此事件函数名！'
    };

    static  propTypes={
        msgName:PropTypes.string.isRequired,
```

```
        onPressCall:PropTypes.func,
    };


    render() {
        return (
            <View style={styles.container}>
                <TouchableOpacity onPress={this.onPressCall} >
                    <View>
                        <Text>{this.props.msgName}</Text>
                    </View>
                </TouchableOpacity>
            </View>
        );
    }

    onPressCall=()=>{
        if(this.props.onPressCall){
            this.props.onPressCall();
        }
    }
}
```

2.【推荐】使用InteractionManager.runAfterInteractions，在动画或者某些特定场景中利用InteractionManager来选择性的渲染新场景所需的最小限度的内容；
使用场景类似于：

```
    class ExpensiveScene extends React.Component {
      constructor(props, context) {
        super(props, context);
        this.state = {renderPlaceholderOnly: true};
      }

      componentDidMount() {
        InteractionManager.runAfterInteractions(() => {
          this.setState({renderPlaceholderOnly: false});
        });
      }

      render() {
        if (this.state.renderPlaceholderOnly) {
          return this.renderPlaceholderView();
        }
```

```
        return (
          <View>
            <Text>Your full view goes here</Text>
          </View>
        );
      }


      renderPlaceholderView() {
        return (
          <View>
            <Text>Loading...</Text>
          </View>
        );
      }
    };
```

说明：更多使用于Navigator的页面跳转
3.【推荐】使用新版本组件替换旧办法组件；
例如：FlatList替换ListView，React Navigation替换Navigator等

4.【推荐】在使用Touchable系列组件时，进行setState或者大量调帧操作，请使用如下方式：

```
handleOnPress() {
          this.requestAnimationFrame(() => {
             //todo
          });
     }
```

五、安全规约

1. 【强制】用户敏感数据禁止直接展示，必须对展示数据脱敏; 说明:查看个人手机号码会显

示成:158****9119，隐藏中间 4 位，防止隐私泄露

2. 【强制】请求传入任何参数必须做有效性验证;避免过度请求服务，造成服务器压力，或者双向校验；
   如：验证手机号长度，是否是手机号等；

六、其他

1. 【推荐】开发工具使用WebStorm,安装ESLint插件进行代码检测，代码中不要出现使用ESLint检查出的错误;
   说明：变量命名规范，使用var或者const错误
2. 【推荐】在WebStorm中导入附件的hoop-settings.jar文件，进行代码格式化，提交的任何代码，都需要进行格式化。快捷键是option+command+L。

对本文档有任何意见或者建议，欢迎在github提问。
GitHub:
https://github.com/sunyardTime/React-Native-CodeStyle