React Native 개요(2)

React Native의 장점

- Javascript와 HTML을 이용하여 웹뷰를 통해 렌더링하는 Cordova, Ionic과는 차이가 있음
  - 대상 플랫폼의 표준 렌더링 API를 사용한다.
  - 개발자가 작성한 마크업을 플랫폼에 따라 실제의 Native Component로 전환
- React는 Main UI 쓰레드와 분리되어 실행됨.
  - 상대적으로 좋은 성능을 냄
- React와 동일한 생명주기
  - React를 이미 다뤄본 개발자라면 빠르게 Native App에 개발에 적용할 수 있음

React Native 개요(3)

React Native의 장점(이어서)

- React에서의 개발도구, 디버깅 도구를 그대로 사용할 수 있음.
  - Xcode나 Android Studio,Eclipse 사용을 강제하지 않는다.
- 크로스 플랫폼 지원
  - React에 대한 지식만 있다면 크로스 플랫폼 대응이 어느정도 가능하다.
  - 작성한 코드들도 상당수 공유할 수 있다.(100%는 아님)

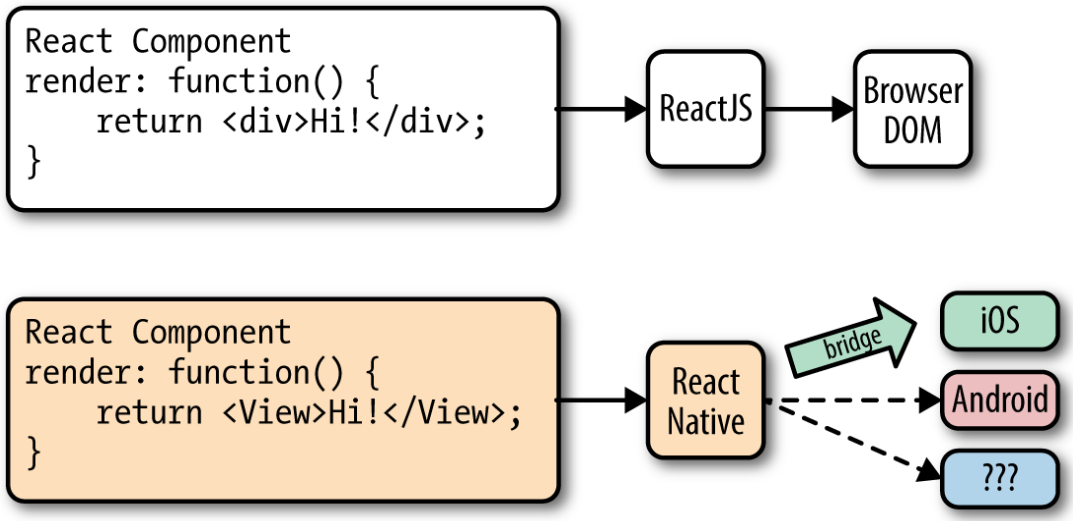
React Native의 단점

- 성숙도의 문제
  - 발표된지 얼마되지 않아 지원되지 않는 기능들도 많고 버그도 존재한다.
- 디버깅이 까다로울 수 있음
  - 자바스크립트 코드 영역과 Native 영역 사이의 비동기 통신!!
  - 디버깅은 어려울 수 있다.

React Native 작동 방식(1)

Virtual DOM

- 성능상의 장점 + 추상화
- Browser DOM으로 렌더링하지 않음
  - iOS : Objective C API --> iOS 컴포넌트 렌더링
  - Android : Java API --> Android 컴포넌트 렌더링
  - Bridge가 각 플랫폼별 인터페이스를 제공하기 때문에 가능한 일이다.



React Native 작동 방식(2)

Rendering Lifecycle

- React Native에서의 Lifecycle은 웹브라우저에서의 Rendering과 동일하지만 절차는 조금 다름
  - React Native는 Bridge에 의존하기 때문
  - JS에서 실행되는 호출을 대상 플랫폼에 포함되어 있는 API, UI 컴포넌트로 연결함.
  - React Native는 Main Thread에서 실행되지 않기 때문에 비동기적으로 실행할 수 있음

React Native UI Component

- --> <View />
- <img /> --> <Image />
- , --> <ListView />
- 사용하기를 원하는 컴포넌트를 명시적으로 import 해야 함
  - import { ListView, DatePickerAndroid } from 'react-native';
- OS 별로 서로 다른 컴포넌트를 사용하는 경우가 많다.
  - 예) DatePickerIOS, DatePickerAndroid
  - 크로스플랫폼 지원을 위해 어떻게 구조화하는지가 중요함

React Native 작동 방식(3)

- JSX
  - React와 동일하게 JSX 사용하여 뷰를 생성함.
  - JSX와 한 파일안에 JS 코드를 작성함.
  - JSX는 하는 일에 따라 구분하여 작성
    - React Native에서 더 중요하게 여겨짐
    - 과거에는 기술에 따른 구분(CSS, HTML, JS)
- Native Component Styling
  - React와 동일하게 CSS 사용
    - props와 state를 기반으로 동적인 Css 클래스 지정가능
  - React Native에서는 브리지가 CSS 구현체 제공
    - React와 동일한 방법으로 CSS 스타일 지정.
  - 인라인 스타일만 지원함.

```
var style = { backgroundColor: 'yellow' };

var tv = (
<View style={style}>
…
</View>);
```

React Native 작동 방식(4)

대상 플랫폼별 API 접근 기능

- 플랫폼별 API
  - 위치 서비스, 카메라 등의 하드웨어 접근 기능
  - React Native는 공통적인 API는 대부분 지원
  - 모든 것을 지원하지는 않는다.
- 이 기능은 특정 플랫폼에서만 작동할 수 있다.
- 모든 것을 제공하지 않으므로 커뮤니티 수준에서 만들어진 API가 존재하는지를 확인해볼 필요가 있다.

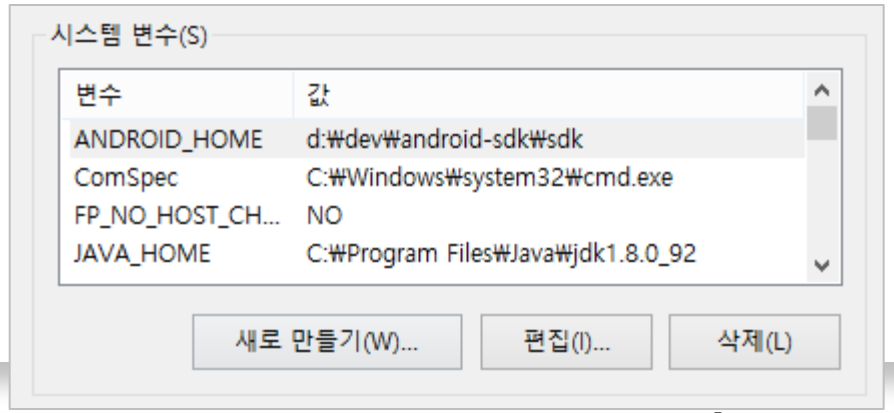
환경 설정(1)

- 현재 지원하는 환경
  - iOS : MacOSX
    - 이 과정에서는 직접 실습하지 않음
  - Android : Win, Linux, MacOSX
  - 이 과정에서는 Windows + Android 중심으로 테스트한다.
  - React Native의 전반적인 작동 방식을 알아보자는 것이 목적임.
- MacOSX 기반 환경 설정
  - Homebrew를 사용하여 설정
    - brew install node
    - brew install watchman
    - brew install flow
  - Xcode 설치: iOS 앱스토어에서 설치
  - React Native 설치
    - npm install -g react-native-cli

환경 설정(2)

Android 기반 환경 설정

- JDK 1.8 이상 설치
  - JAVA_HOME 환경 변수 지정
- Android Studio 설치
- Android SDK 설치
  - SDK는 Android Studio를 설치할 때 같이 설치해도 되고, 별도로 설치해도 됨.
  - 설치후 Android 버전별로 다운로드 받아야 할 것들이 많기 때문에 과정 중에서는 미리 제공함.
  - ANDROID_HOME 환경 변수 지정 : Android SDK 디렉토리를 지정함



환경 설정(3)

- Android SDK Manager 실행
  - 적절한 요소가 설치되어 있는지 확인
    - Android SDK Build-tool version X.X.X
    - Android X.X(API XX)
    - Android Support Repository
  - Emulator 관련 도구
    - Intel x86 Atom System Image(for Andriod X.X.X - API XX)
    - Intel x86 Emulator Accelerator(HAXM installer)
  - 모든 버전을 설치할 필요는 없음.

Components(1)

- **React Native의 컴포넌트와 React의 컴포넌트는 서로 공유할 수 없음**
  - React Native는 HTML 요소를 이용하여 렌더링할 수 없음
  - 렌더링하는 UI가 없다면 공유가능
- **텍스트 컴포넌트**
  - Text : 텍스트 표시 기능

```
<Text>Hello World!!</Text>
```

  - Style 지정 가능 : JS 객체를 {} 안에 표현
    - HTML CSS Style과는 차이가 있음 (CSS : { font-size:20pt; font-style:"verdana" })

```
<Text style={{fontSize:20, fontStyle:'verdana' }}>
   Hello World!!
</Text>
```

    - 외부에 별도의 Style 변수를 지정해 할당하는 것도 가능함.

Components(2)

Image Component

- 이미지 컴포넌트

```
<Image source={require('./my-icon.png')} />
```

  - 안드로이드, iOS 모두 지원하므로 my-icon.ios.png, my-icon.android.png 을 작성하고 있는 Component와 같은 폴더에 포함시킨다.

- 조건에 따라 다른 이미지 보기

```
var icon = this.props.active ? require('./my-icon-active.png') : require('./my-icon-inactive.png');

<Image source={icon} />
```

- 원격서버의 이미지

```
<Image source={{uri: 'https://facebook.github.io/react/img/logo_og.png'}}
    style={{width: 400, height: 400}} />
```

- 백그라운드 이미지

```
return ( <Image source={...}> <Text>Inside</Text> </Image> );
```

Components(3)

ListView Component

- 수직 스크롤이 가능한 목록형 UI 제공
- 두가지 속성
  - dataSource : 리스트(배열)의 데이터 제공
  - renderRow
    - dataSource의 리스트를 받아 아이템마다 실행할 함수를 등록
    - 지정된 함수를 ListView에 보여줄 Item를 생성함.
- rowChanged 함수를 반드시 작성해야 함.
  - 보여줄 데이터 목록이 변경되면 dataSource를 변경한다.

```
class ListViewBasics extends Component {
   // Initialize the hardcoded data
   constructor(props) {
    super(props);
    const ds =
        new  ListView.DataSource({
            rowHasChanged: (r1, r2) => r1 !== r2});
    this.state = {
      dataSource:
         ds.cloneWithRows([ 'John', 'Joel', 'James',
'Jimmy', 'Jackson', 'Jillian', 'Julie', 'Devin'])
    };
   }
   render() {
    return (
    <View style={{paddingTop: 22}}>
     <ListView
       dataSource={this.state.dataSource}
       renderRow={
         (rowData) => <Text>{rowData}</Text>
       }
     />
    </View>
   );
  }
}
```

Components(4)

Navigator

- 여러 화면 간에 Navigation 기능을 제공하는 컴포넌트
- initialRoute : 초기에 보여줄 화면에 대한 정보
- renderScene : 라우트 정보에 의해 보여줄 화면을 렌더링함.

```
render() {
   return (
      <Navigator
         initialRoute={{ title: 'My Initial Scene', index: 0 }}
         renderScene={(route, navigator) => {
            return <MyScene title={route.title} />
         }} />
   );
}
```

Components(5)

- TouchableHighlight
  - 사용자의 터치에 반응하는 UI 작성시 사용
  - onPressIn
  - onPressOut
  - onLongPress

```
render() {
    return (
        <TouchableHighlight onPress={this._onPressButton}>
            <Image style={styles.button} source={require('./favorite.png')} />
        </TouchableHighlight>
    );
},
```

  - 단순한 터치가 아닌 멀티 터치등의 복잡한 작업은 GestureResponder를 이용해 작성함.
    - 이번 과정에서는 자세히 다루지 않음

Components(6)

- TextInput
  - 문자열 입력을 받기 위한 컴포넌트
  - 속성
    - autoFocus, autoCapitalize, defaultValue, editable
    - keyboardType, maxLength, multiline, placeholder, secureTextEntry(****)
  - 이벤트
    - onChangeText, onEndEditing, onSelectionChanged, onSubmitEditing

```
render() {
  return (
    <TextInput
      style={{height: 40, borderColor: 'gray', borderWidth: 1}}
      onChangeText={(text) => this.setState({text})}
      value={this.state.text}
    />
  );
}
```

Components(7)

- **이밖에도 많은 UI 컴포넌트가 있다.**
  - 내려받아 사용가능한 UI 컴포넌트도 있음
  - react native material UI 디자인 : 다양한 컴포넌트와 UI 템플릿 제공
    - https://github.com/react-native-material-design/react-native-material-design

Style(1)

- React는 CSS 스타일을 준용함.
  - 하지만 작성 형식은 inline style!!
    - HTML => inputText { font-size : 20pt; color: aqua }
    - React Native => var inputStyle = { fontSize:20, color:'aqua' }
      - JS 객체 형식으로 지정
    - <Text style={inputStyle}>Hello</Text>
  - StyleSheet.create() 메서드
    - 선택적 사용이나 권장함.
    - 스타일의 작성시 문법적 실수를 방지.
    - propTypes로 prop 검증
    - props이나 state이나 저장하고 디자인 정보가 변경될 때 불변성을 제공함.

Style(2)

Style 내보내기 & 사용하기

```
//--styles.js
const styles = StyleSheet.create({
    test1 : {  color: 'blue', fontWeight: 'bold', fontSize: 30 },
    test2 : { color: 'red' },
}); red
export default styles;
```

```
//--TextSample.js
import styles from './styles';
……
class  TextSample extends Component {
  render() {
    return (
      <View>
        <Text style={styles.test1}>Test1</Text>
        <Text style={styles.test2}>Test2</Text>
        <Text style={[styles.test1, styles.test2]}>test1-test2</Text>
        <Text style={[styles.test2, styles.test1]}>test2-test1</Text>
      </View>
    );
  }
}
```

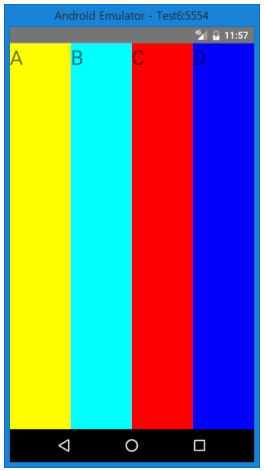
Style(3)

Flex Layout

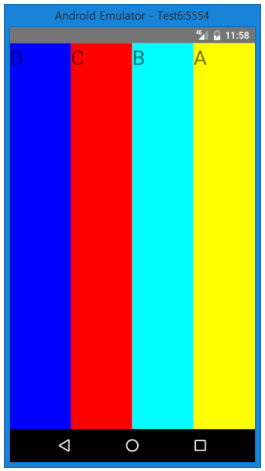
```
constr Styles = {
    container1 : { flex:1 , flexDirection:'row' },
    container2 : { flex:1 , flexDirection:'row-reverse' },
    container3 : { flex:1 , flexDirection:'column' },
    container4 : { flex:1 , flexDirection:'column-reverse' }
}
```

```
<View style={Styles.container1}>
    <View style={{flex:1, backgroundColor:'yellow'}}>
        <Text style={{fontSize:30}}>A</Text></View>
    <View style={{flex:1, backgroundColor:'aqua'}}>
        <Text style={{fontSize:30}}>B</Text></View>
    <View style={{flex:1, backgroundColor:'red'}}>
        <Text style={{fontSize:30}}>C</Text></View>
    <View style={{flex:1, backgroundColor:'blue'}}>
        <Text style={{fontSize:30}}>D</Text></View>
</View>
```

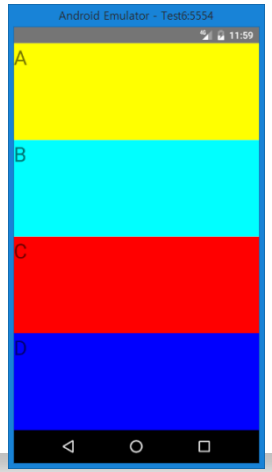
container1



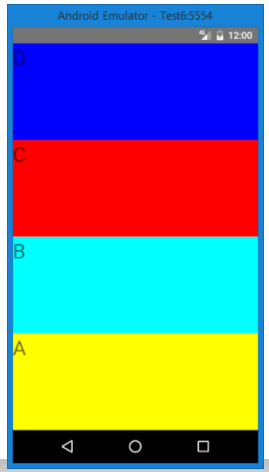
container2



container3



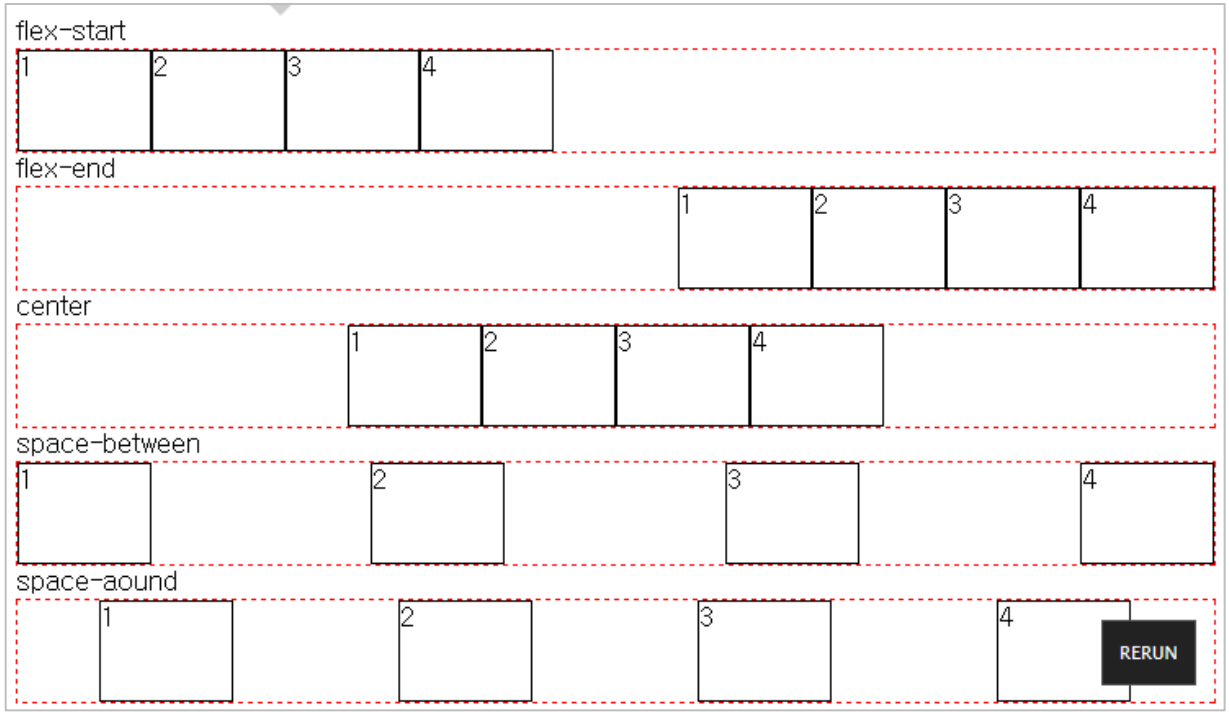
container4



Style(4)

justifyContent

- 컨테이너 안에서 가로로 정렬하는 방법 제공
- [flex-start, flex-end, center, space-between, space-around]



참조 : http://uxuiz.cafe24.com/wp/?p=1050

Platform API(1)

Geolocation API

- 사용하기 위해서 permission 부여
  - iOS : info.plist 파일에 NSLocationWhenInUseUsageDescription Key가 포함되어야 함.(react-native init 를 실행해서 프로젝트를 생성하면 자동으로 포함)
  - Android : AndroidManifest.xml 파일에 다음 내용이 추가되어야 함.
    - <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
- 지원 API
  - static getCurrentPosition(successCallback[, errorCallback][, options])
    - 현재 위치 정보를 획득하는 함수
  - static watchPosition(successCallback[, errorCallback][, options])
    - 위치가 변경될때마다 successCallback을 호출하는 함수
  - static clearWatch(watchID)
    - watchPosition() 처리를 종료하는 함수

Platform API(2)

CameraRoll API

- 로컬 카메라 롤, 갤러리에 액세스를 허용하는 API
- API
  - promise getPhotos(params)
    - params 옵션으로 카메라 롤에서 이미지 가져옴
    - 리턴값은 promise객체 --> 비동기 처리

```
var fetchParams: Object = {
      first: this.props.batchSize,
      groupTypes: this.props.groupTypes,
      assetType: this.props.assetType,
};

CameraRoll.getPhotos(fetchParams)
.then((data) => this._appendAssets(data), (e) => logError(e));
```

  - saveToCameraRoll(tag[, type])
    - 사진이나 동영상 클립을 카메라 롤/갤러리에 저장
    - tag : 파일의 경로 [안드로이드 예 : "file:///sdcard/abc/test.png"]
    - 리턴 값은 promise 객체

Platform API(3)

AsyncStorage

- 로컬 저장소 기능. 브라우저의 localStorage API와 유사함
- API
  - getItem(key[, callback])
  - setItem(key, value[, callback])
  - removeItem(key[, callback])
  - mergeItem(key, value[, callback])
  - clear([callback])

```
AsyncStorage.setItem('A-001', JSON.stringify(obj) );
```

Platform API(4)

BackAndroid(안드로이드 전용)

- 안드로이드에서 back 버튼을 누르는 것을 탐지하여 프로그래밍 처리를 가능하게 하는 API
- API
  - exitApp() : App 종료
  - addEventListener(eventName, handler)
  - removeEventListener(eventName, handler)

```
BackAndroid.addEventListener('hardwareBackPress', function() {
  if (!this.onMainScreen()) {
      this.goBack();
      return true;
  }
  return false;
});
```

Platform API(5)

Clipboard

- 클립보드에 쓰기/읽기를 제공함.
  - getString()
  - setString(value)

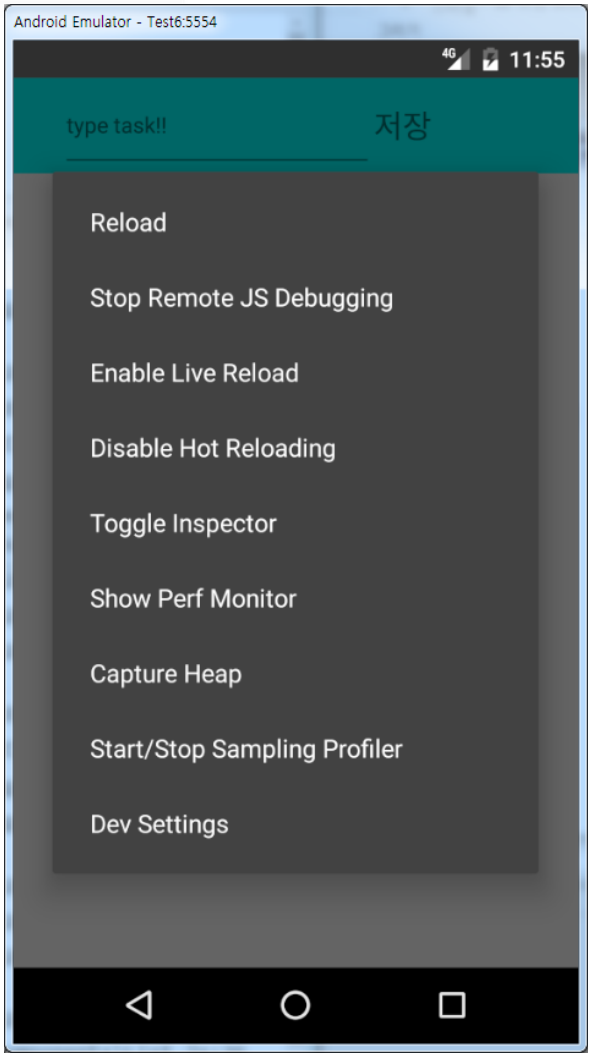
Alert

- 메시지박스 기능 제공
  - Browser의 alert(), confirm() 과 차이가 있음. 버튼을 직접 정의할 수 있음.

```
Alert.alert( 'Title', 'Message', [
      {text: 'Cancel', onPress: () => console.log('Cancel!!')},
      {text: 'OK', onPress: () => console.log('OK!!')}
])
```

디버깅(1)

- **개발자 옵션 활성화**
  - iOS 시뮬레이터 : Command + Control + Z
  - Android 에뮬레이터 : Control + M
- **원격 디버깅**
  - 개발자 옵션 활성화 시킨 후에 나타난 메뉴에서 'Debug in Chrome' 또는 'Remote JS Debugging'을 선택함.
  - 개발 코드에서 console.log()로 출력한 코드가 크롬 브라우저의 개발자 도구에 나타남.
    - http://localhost:8081/debugger-ui
  - Android의 경우 "react-native log-android" 명령어로 로그뷰를 얻을 수 있음



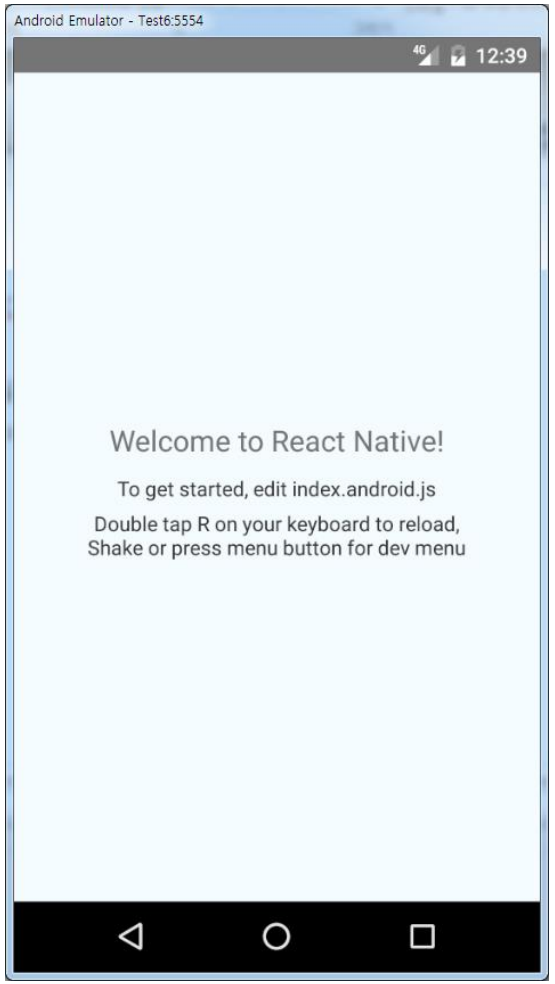
디버깅(2)

Red Screen!

- 에러 발생시에 자주 보게 되는 화면
- 자세히 들여다보면 간단한 문제인 경우가 많다.
  - 변수명 누락
  - 라이브러리 참조 오류(주로 경로 지정 문제)
  - 잘못된 this의 참조
  - Syntax Error
  - Style 구문 문법 오류. 잘못된 스타일 속성 지정
    - fontWeight(O)
    - font-weight(X)

간단한 계산기(1)

- 최종적으로 간단한 계산기 작성
  - 개발 환경상의 문제로 안드로이드로 중심으로 작성함.
- 프로젝트 생성
  - 프로젝트 초기화
    - react-native init ReactCalc : 약간의 시간 필요!
    - cd ReactCalc
  - 생성된 코드 그대로 실행
    - react-native run-android 또는 react-native run-ios



간단한 계산기(2)

HelloWorld 기능

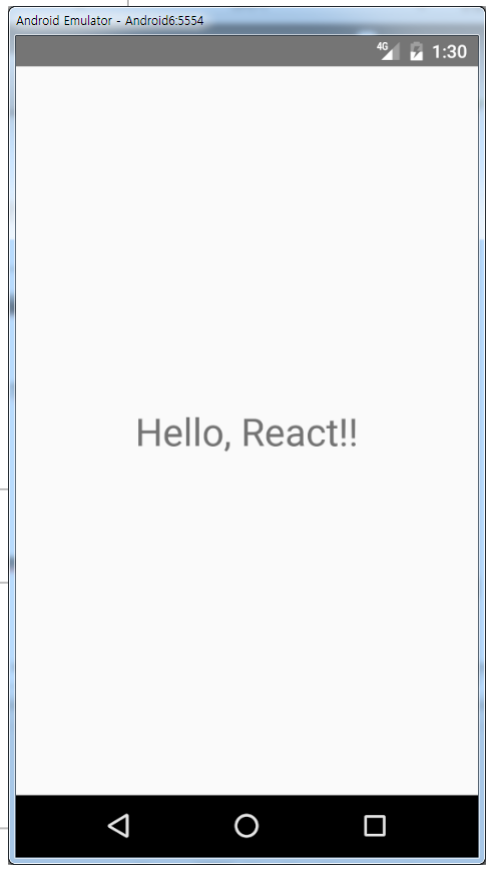
- index.android.js 파일을 재작성
  - HelloWorld를 출력하도록 변경
  - 여러 개의 파일로 분리
- 스타일 작성 : app/Style.js

```
const Style = {
    rootContainer : {
        flex:1,
        alignItems:'center',
        justifyContent:'center'
    },
    helloText : {
        fontSize:30
    }
};
export default Style;
```

간단한 계산기(3)

- ReactCalc 클래스 : app/ReactCalc.js

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';
import Style from './Style';
class ReactCalc extends Component {
    render() {
        return (
          <View style={Style.rootContainer}>
            <Text style={Style.helloText}>Hello, React!!</Text>
          </View>
        )
    }
}
export default ReactCalc;
```



- index.android.js

```
import React, { Component } from 'react';
import { AppRegistry } from 'react-native';
import ReactCalc from './app/ReactCalc';

AppRegistry.registerComponent('ReactCalc', () => ReactCalc);
```

간단한 계산기(4)

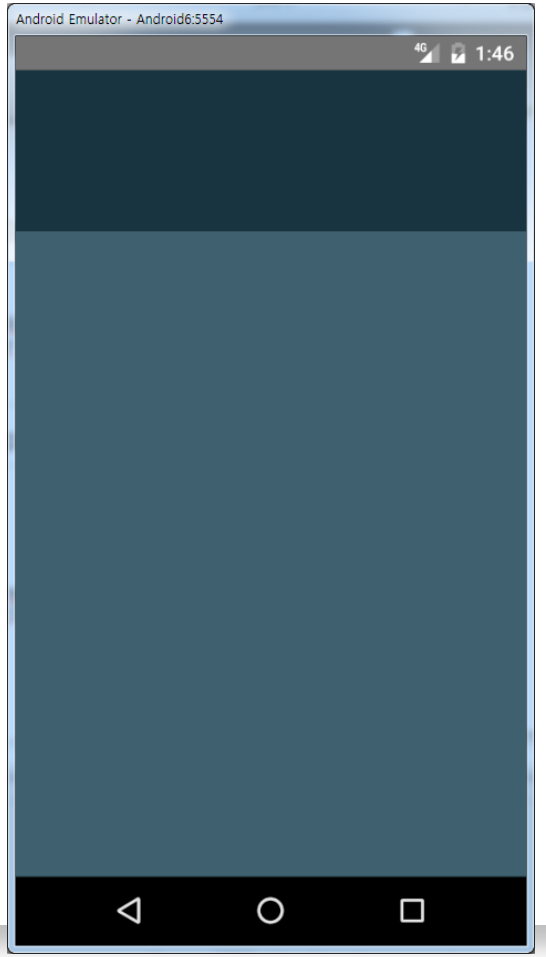
계산기 레이아웃 작성

- Style 파일 전체 수정 : app/Style.js

```
const Style = StyleSheet.create({
    rootContainer: {  flex: 1  },
    displayContainer: { flex: 2, backgroundColor: '#193441' },
    inputContainer: { flex: 8, backgroundColor: '#3E606F' }
});
export default Style;
```

- ReactCalc 클래스 변경 : app/ReactCalc.js

```
……
class ReactCalc extends Component {
    render() {
        return (
            <View style={Style.rootContainer}>
                <View style={Style.displayContainer}></View>
                <View style={Style.inputContainer}></View>
            </View>
        )
    }
}
……
```



간단한 계산기(5)

계산기 버튼 추가

- InputButton 컴포넌트 생성 : app/InputButton.js

```
import React, { Component } from 'react';
import { View, Text } from 'react-native';
import Style from './Style';

class InputButton extends Component {
    render() {
        return (
            <View style={Style.inputButton}>
                <Text style={Style.inputButtonText}>{this.props.value}</Text>
            </View>
        )
    }
}
export default InputButton;
```

간단한 계산기(6)

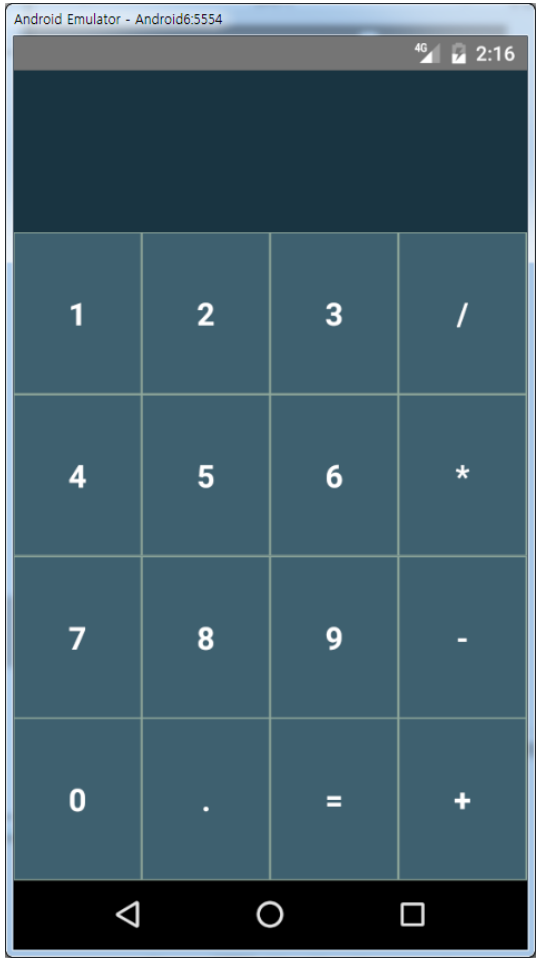
- ReactCalc 클래스에서 InputButton 렌더링

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';
import Style from './Style';
import InputButton from './InputButton';

//버튼 리스트
const inputButtons = [
    [1, 2, 3, '/'],
    [4, 5, 6, '*'],
    [7, 8, 9, '-'],
    [0, '.', '=', '+']
];

class ReactCalc extends Component {
    renderInputButtonList() {
      let views = [];
      for (var r = 0; r < inputButtons.length; r ++) {
        let row = inputButtons[r];
        let inputRow = [
          for (var i = 0; i < row.length; i ++) {
            let input = row[i];
            inputRow.push(
            <InputButton value={input} key={r + "-" + i} />
            );
          }
```

```
          views.push(<View style={Style.inputRow}
            key={"row-" + r}>{inputRow}</View>);
        }
        return views;
    }
    render() {
        return (
            <View style={Style.rootContainer}>
                <View style={Style.displayContainer}></View>
                <View style={Style.inputContainer}>
                    {this.renderInputButtonList()}
                </View>
            </View>
        )
    }
}

export default ReactCalc;
```

간단한 계산기(7)

- 버튼 디자인 적용 결과

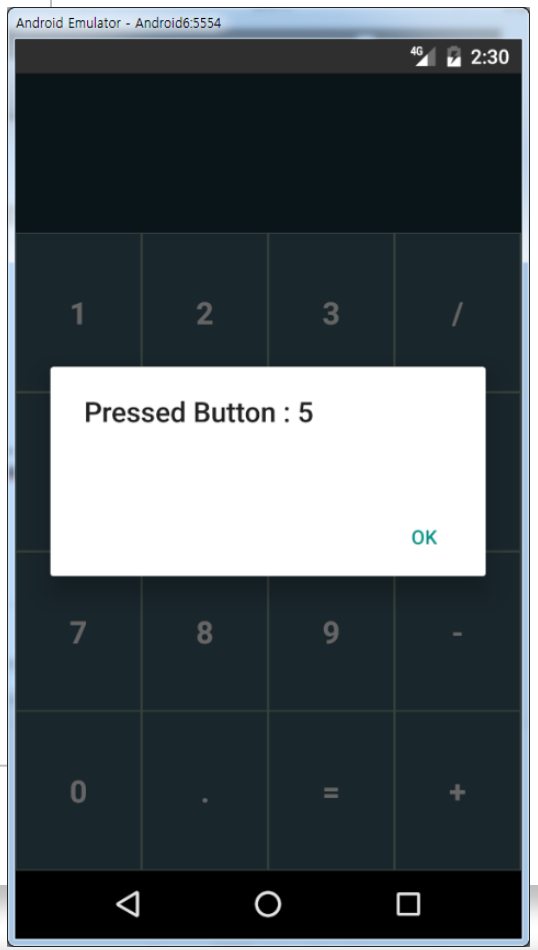


간단한 계산기(8)

터치 이벤트 적용

- InputButton에 터치시 사용할 메서드를 속성으로 전달

```
import { Text, View, Alert } from 'react-native';
……

handleNumButtonPress(input) {
    Alert.alert('Pressed Button : ' + input)
}
renderInputButtonList() {
    ……
    for (var i = 0; i < row.length; i ++) {
        inputRow.push(
            <InputButton value={input} key={r + "-" + i}
              onPress={this.handleNumButtonPress.bind(this, input)}
            />
        );
    }
    ……
    return views;
}
```



간단한 계산기(9)

- state 사용
  - ReactCalc 변경

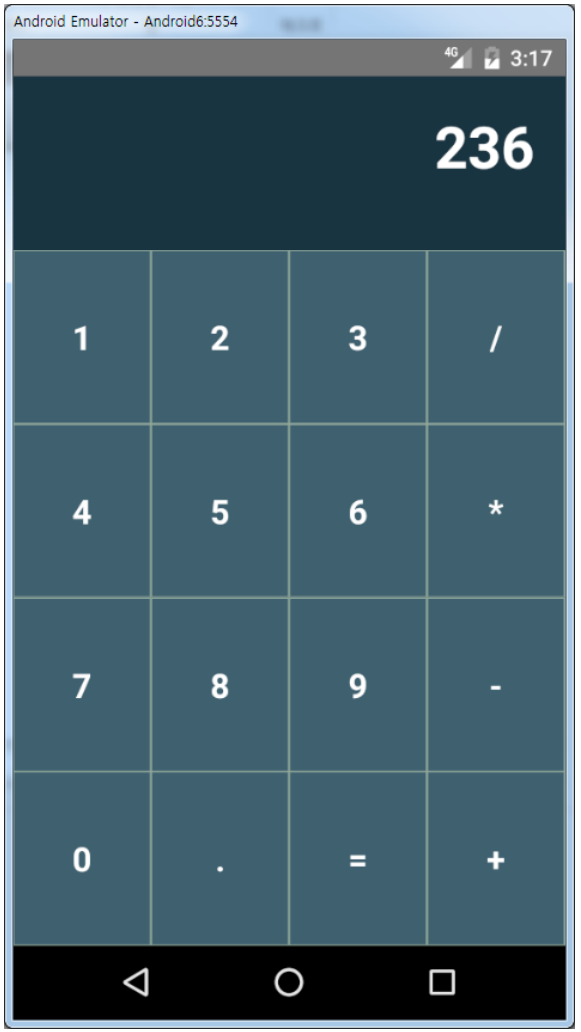
```
class ReactCalc extends Component {
    constructor(props) {
        super(props);
        this.state = {
            inputValue: 0
        }
    }
    handleNumButtonPress(input) {
        switch (typeof input) {
            case 'number':
                return this.handleNumberInput(input)
        }
    }
    handleNumberInput(num) {
        let inputValue = (this.state.inputValue * 10) + num;
        this.setState({
            inputValue: inputValue
        })
    }
    ……
}
```

```
render() {
    return (
        <View style={Style.rootContainer}>
            <View style={Style.displayContainer}>
                <Text style={Style.displayText}>
                        {this.state.inputValue}</Text>
            </View>
            <View style={Style.inputContainer}>
                {this.renderInputButtonList()}
            </View>
        </View>
    )
}
```

간단한 계산기(10)

state 적용후 실행 결과

- 숫자 버튼을 눌렀을 때 state의 inputValue에 값을 저장하고 저장된 state는 렌더링을 통해 화면에 반영됨



간단한 계산기(11)

계산 기능 추가

- 계산 버튼을 눌렀을 때의 처리를 위해 state 정보를 추가한다.
  - 직전의 입력 값 : previousInputValue
  - 현재 버튼의 입력값 : InputValue
  - 앞에서 누른 계산 버튼 : selectedSymbol
- 버튼을 렌더링하는 도중 버튼이 이미 앞에서 누른 계산 버튼과 일치하면 시각적으로 식별할 수 있도록 버튼의 스타일을 추가한다.

```
import { StyleSheet } from 'react-native';
const Style = StyleSheet.create({
    ......
    inputButtonHighlighted: {
        backgroundColor: '#193441'
    }
});
export default Style;
```

간단한 계산기(12)

- InputButton 클래스 수정

```
render() {
    return (
        <TouchableHighlight
            style={[Style.inputButton, this.props.highlight ? Style.inputButtonHighlighted : null]}
                    underlayColor="#193441"
                    onPress={this.props.onPress}>
            <Text style={Style.inputButtonText}>{this.props.value}</Text>
        </TouchableHighlight>
    )
}
```

- ReactCalc 클래스의 생성자 수정

```
constructor(props) {
    super(props);
    this.state = {
        previousInputValue: 0,
        inputValue: 0,
        selectedSymbol: null
    }
}
```

간단한 계산기(13)

- ReactCalc 클래스에 메서드 추가/변경

```
handleNumButtonPress(input) {
    switch (typeof input) {
        case 'number':
            return this.handleNumberInput(input);
        case 'string':
            return this.handleStringInput(input);
    }
}

handleStringInput(str) {
    switch (str) {
        case '/':
        case '*':
        case '+':
        case '-':
            this.setState({
                selectedSymbol: str,
                previousInputValue: this.state.inputValue,
                inputValue: 0
            });
            break;
```

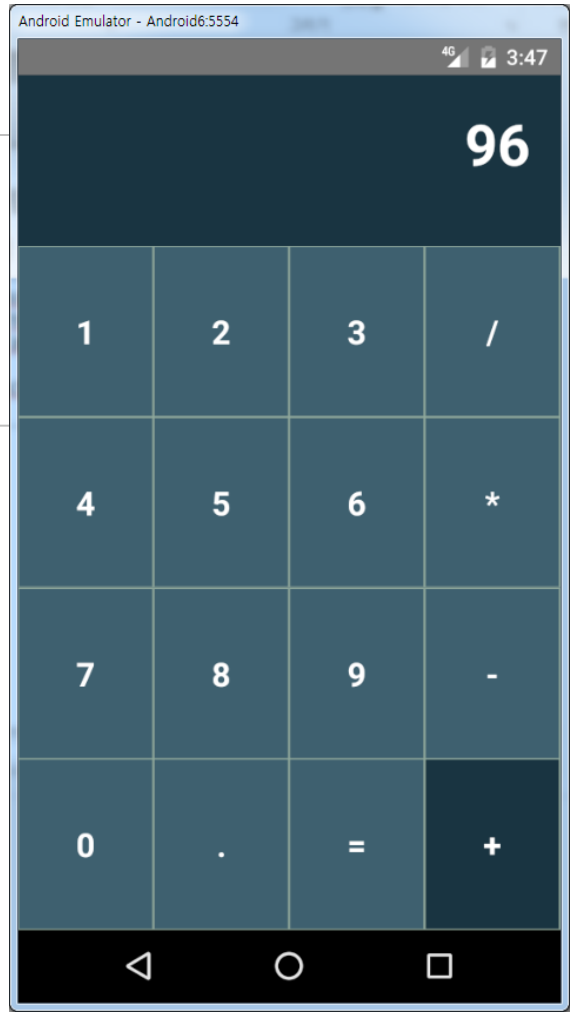
```
        case '=':
            let symbol = this.state.selectedSymbol;
            let inputValue = this.state.inputValue;
            let previousInputValue =
                    this.state.previousInputValue;

            if (!symbol) {
                return;
            }

            this.setState({
                previousInputValue: 0,
                inputValue: eval(previousInputValue +
                    symbol + inputValue),
                selectedSymbol: null
            });
            break;
    }
}
```

간단한 계산기(14)

Back 버튼을 눌러 계산기 앱 종료하기

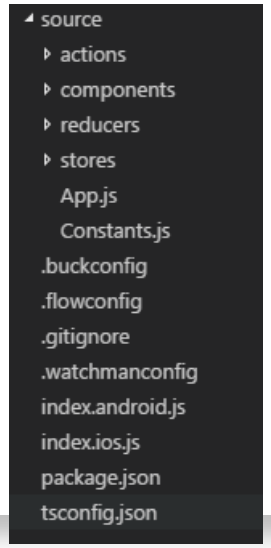
- index.android.js 파일 수정

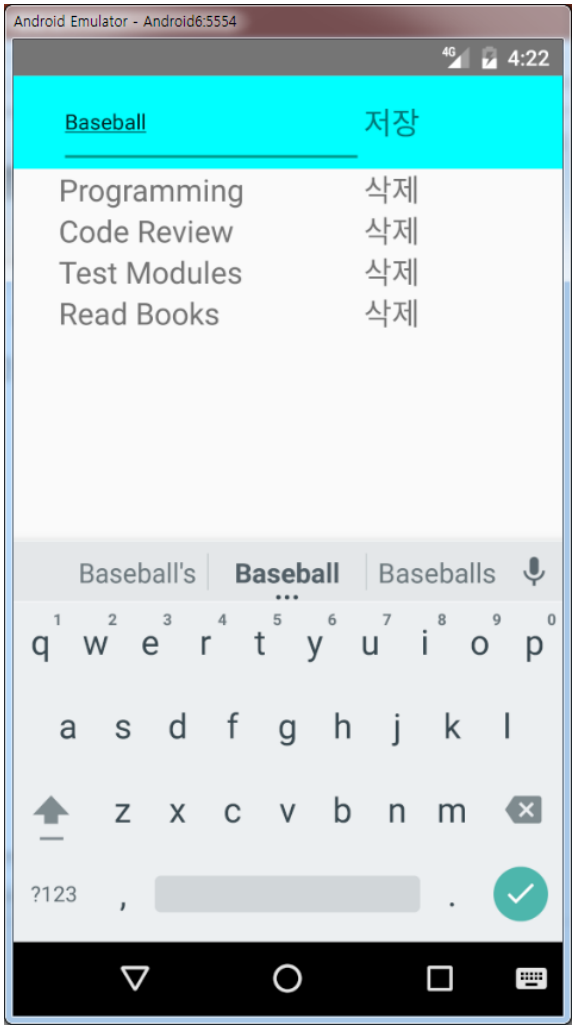
```
import React, { Component } from 'react';
import { AppRegistry, BackAndroid } from 'react-native';
import ReactCalc from './app/ReactCalc';

AppRegistry.registerComponent('ReactCalc', () => ReactCalc);
BackAndroid.exitApp();
```



TodoList App(1)

- React Native + Redux 를 적용한 TodoList App
  - 디자인 스타일에 신경쓰지 않고 Redux를 적용하는 측면에 집중
- 프로젝트 초기화
  - react-native init MyTodoApp
  - npm install --save react-redux redux
  - npm install --save react-addons-update
  - 디렉토리 구조





TodoList App(2)

- 상수파일 작성 : source/Constants.js

```
export default {
    ADD_ITEM : 'add item',
    REMOVE_ITEM : 'remove item'
};
```

- 리듀서 작성 : source/reducers/TodoReducer.js

```
import constants from '../Constants';
import update from 'react-addons-update';

const initialState = {
    list : []
}
const todoReducer = (state = initialState, action) => {
    console.log("### red : ");
    console.log(action);
    switch (action.type) {
        case constants.ADD_ITEM :
        let ret1 = {
            list : update(state.list,
                { $push:[action.data] })
        };
        return ret1;
```

```
 case constants.REMOVE_ITEM:
        let ret2 = {
            list : update(state.list,
                { $splice: [[action.data, 1 ]] })
        };
        return ret2;
    default:
        return state;
    }
};
export default todoReducer;
```

TodoList App(3)

- 스토어 작성 : source/stores/TodoStore.js

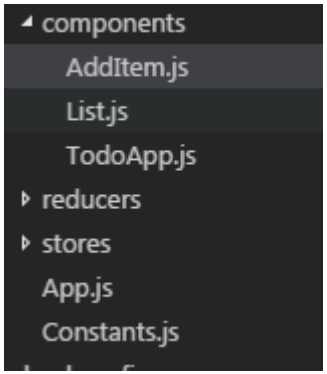
```
import { createStore } from 'redux';
import todoReducer from '../reducers/TodoReducer';

const todoStore = createStore(todoReducer);
export default todoStore;
```

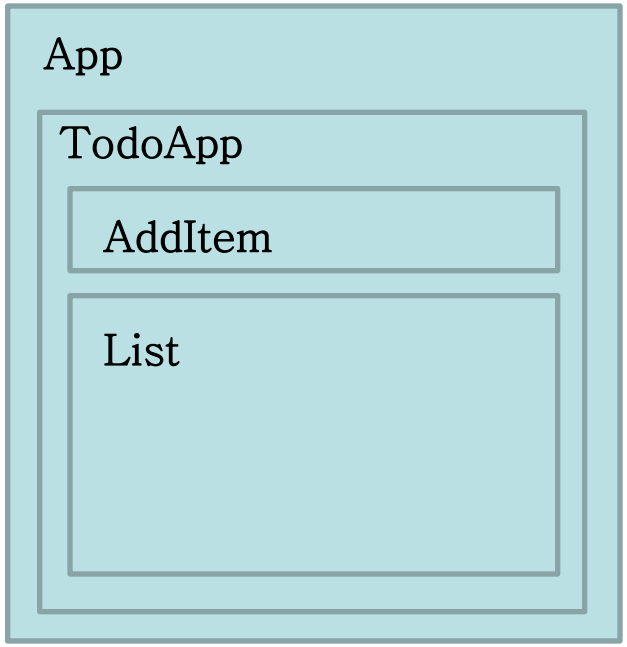
- 액션 생성자 작성 : source/actions/TodoActionCreators.js

```
import constants from '../Constants';
const todoActionCreators = {
    addItem(item) {
        return {    type: constants.ADD_ITEM,  data : item   };
    },
    removeItem(index) {
        return {    type: constants.REMOVE_ITEM, data : index   };
    }
}
export default todoActionCreators;
```

Component 작성

- 먼저 컴포넌트 구조 정의





index.android.js 수정

```
import { AppRegistry, BackAndroid } from 'react-native';
import App from './source/App';

AppRegistry.registerComponent('MyTodoApp', () => App);
BackAndroid.exitApp();
```

TodoList App(4)

- source/App.js 작성

```
import React, { Component, PropTypes } from 'react';
import todoStore from './stores/TodoStore';
import constants from './Constants';
import todoActionCreators from './actions/TodoActionCreators';
import TodoApp from './components/TodoApp';
import { connect, Provider } from 'react-redux';
import { AppRegistry } from 'react-native';

const mapStateToProps = (state) => {
    return {
        list: state.list
    }
}

const mapDispatchToProps = (dispatch) => {
    return {
        onAddItem: (item) => dispatch(todoActionCreators.addItem(item)),
        onRemoveItem: (index) => dispatch(todoActionCreators.removeItem(index))
    }
}
```

TodoList App(5)

- App.js (이어서)

```
const TodoAppContainer = connect(mapStateToProps, mapDispatchToProps)(TodoApp);

const App = () => {
    return (
        <Provider store={todoStore}>
            <TodoAppContainer />
        </Provider>
    );
};

export default App;
```

  - 애플리케이션의 state와 dispatch 호출을 자식컴포넌트로 속성을 맵핑한 컨테이너 컴포넌트를 자동으로 생성함. 이를 위해 connect() 함수를 사용한다.
  - 앞서 생성한 컨테이너 컴포넌트를 감싸는 Provider 컴포넌트를 사용함. 이를 통해 자식 컴포넌트들이 스토어에 연결될 수 있도록 한다.
  - 리턴된 컴포넌트를 App으로 export 한다.

TodoList App(6)

- source/components/TodoApp.js

```
import React, { Component } from 'react';
import AddItem from './AddItem';
import List from './List';
import { View } from 'react-native';

class TodoApp extends Component {
    render() {
        console.log("##TodoApp Render : ");
        console.log(this.props.onRemoveItem);
        return (
            <View style={{ flex:1 }}>
                <View style={{ height:70 }}>
                    <AddItem add={this.props.onAddItem}/>
                </View>
                <View>
                    <List items={this.props.list} remove={this.props.onRemoveItem} />
                </View>
            </View>
        );
    }
}
export default TodoApp;
```

TodoList App(7)

- source/components/List.js

```
import React, { Component } from 'react';
import { ListView, View, Text, TouchableHighlight } from 'react-native';

const ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});

class List extends Component {
    constructor(props) {
        super(props);
        this.state = {
            dataSource: ds.cloneWithRows(this.props.items)
        };
    }
    updateDataSource(items) {
        this.setState({
            dataSource: ds.cloneWithRows(items)
        });
    }
    componentWillReceiveProps(newProps) {
        this.updateDataSource(newProps.items);
    }
```

TodoList App(8)

- source/components/List.js(이어서)

```
  renderRow(item, sectionId, rowId) {
    console.log("### renderRow : ");
    console.log(this.props.remove);
    return (
    <View style={{flex:1, flexDirection: 'row', alignItems: 'center', justifyContent: 'center' }}>
        <View style={{ width:200 }}>
            <Text style={{ fontSize:20 }}>{item}</Text>
        </View>
        <TouchableHighlight style={{ width:100 }}
            onPress={this.props.remove.bind(this, rowId)}
            activeOpacity={75 / 100}>
            <Text style={{ fontSize:20 }}>삭제</Text>
        </TouchableHighlight>
    </View>
    );
  }
```

TodoList App(9)

- source/components/List.js(이어서)

```
 render() {
        console.log("### render : ");
        return (
            <View>
                <ListView
                    dataSource={this.state.dataSource}
                    renderRow={this.renderRow.bind(this)}
                    enableEmptySections={true}
                    automaticallyAdjustContentInsets={false}
                />
            </View>
        );
    }
}

export default List;
```

TodoList App(10)

- source/components/AddItem.js

```
import React, { Component } from 'react';
import { View, TextInput, Text, TouchableHighlight } from 'react-native';
import DismissKeyboard from 'dismissKeyboard';

class AddItem extends Component {
    constructor(props) {
        super(props);
        this.state = {
            text: ''
        };
        this.save = this.save.bind(this);
        this.reset = this.reset.bind(this);
        this.handleTextChange = this.handleTextChange.bind(this);
    }

    handleTextChange(text) {
        this.setState({text});
    }

    reset() {
        this.setState({text: ''});
    }
```

TodoList App(11)

- source/components/AddItem.js

```
    save() {
        if (this.state.text.trim() != "") {
            this.props.add(this.state.text);
            this.reset();
            DismissKeyboard();
        }
    }
    render(){
        return (
        <View style={{backgroundColor:'aqua', flex: 1, flexDirection: 'row',
            justifyContent: 'center', alignItems: 'center'}}>
            <TextInput style={{width:200}}
                ref="item"  placeholder={ 'type task!!' } value={ this.state.text || '' }
                onChangeText={ this.handleTextChange }
                onSubmitEditing={ this.reset }
            />
            <TouchableHighlight style={{width:100}}
                onPress={ this.save.bind(this) }
                activeOpacity={1}>
                    <Text style={{fontSize:20}}>저장</Text>
            </TouchableHighlight>
        </View>
        )
    }
}
export default AddItem;
```

TodoList App(12)

- 작성이 완료되었다면 실행
  - react-native run-android
- DismissKeyboard API
  - 입력후 저장버튼을 누르면 안드로이드 내장 키보드가 사라지도록 하기 위해 사용
- update 함수
  - reducer에서 TodoList를 갱신하기 위해 사용함
  - 불변성 객체 상태로 업데이트하기 위한 방법 제공

배포(1)

Android APK

- 아이콘 이미지 변경
  - 해상도별로 여러개 준비
  - android/app/src/main/AndroidManifest.xml에 지정
  - 파일의 경로는 android/app/src/main/res 경로를 기점으로 상대 경로로 표현
  - 아이콘 이미지 만들어주는 도구
    - https://romannurik.github.io/AndroidAssetStudio/index.html

배포(2)

배포 단계

- 서명용 키 생성
  - keytool -genkey -v -keystore mykey.keystore -alias mykey -keyalg RSA -keysize 2048 -validity 10000
  - 만들어진 키스토어 파일을 android/app/ 디렉토리 아래로 복사
- gradle 변수 설정
  - android/app/gradle.properties 파일 추가

```
MYAPP_RELEASE_STORE_FILE=mykey.keystore
MYAPP_RELEASE_KEY_ALIAS=mykey
MYAPP_RELEASE_STORE_PASSWORD=****
MYAPP_RELEASE_KEY_PASSWORD=****
```

배포(3)

- android/app/build.gradle 설정에 서명 정보 추가

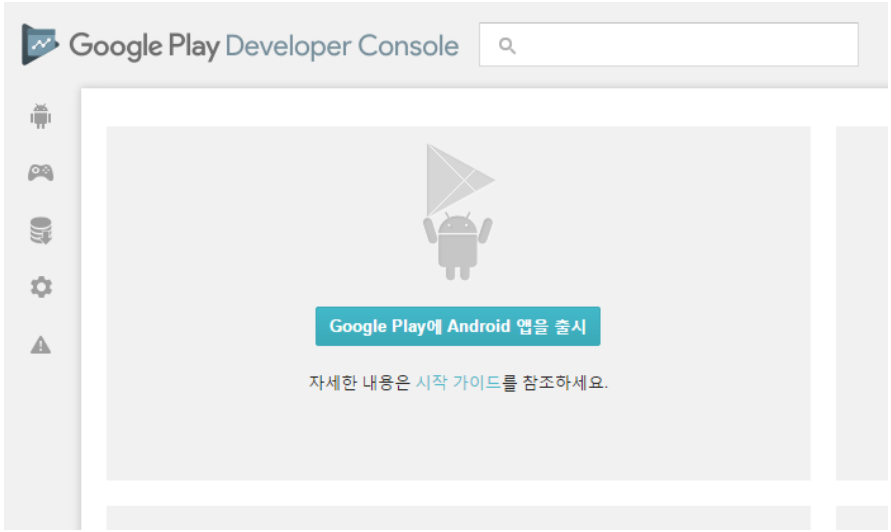
```
android {
    ……
    signingConfigs {
        release {
            storeFile "MYAPP_RELEASE_STORE_FILE"
            storePassword "MYAPP_RELEASE_STORE_PASSWORD"
            keyAlias "MYAPP_RELEASE_KEY_ALIAS"
            keyPassword "MYAPP_RELEASE_KEY_PASSWORD"
        }
    }
    buildTypes {
        release {
            ……
            signingConfig signingConfigs.release
        }
    }
    ……
}
```

- apk 생성
  - cd android
  - ./gradlew assembleRelease

배포(4)

Playstore에 올리기

- http://developer.android.com에 접속
  - 화면 오른쪽의 Developer Console로 이동
  - 개발자 등록을 위해 $25 결제 필요



  - 위 그림의 버튼을 클릭하여 빌드한 release 버전의 apk 업로드하고 나머지 스토어 등록정보를 입력함.

배포(5)

iOS 앱 배포

- 배포를 위한 xcode 프로젝트 설정
- 개발자 등록
- 기기 등록
- 애플리케이션 올리기
- 애플리케이션 심사를 위한 등록

- 자세한 내용은 생략

create-react-native-app 활용(1)

- create-react-native-app?
  - react native app을 더욱 쉽게 개발할 수 있도록 하는 새로운 도구
  - facebook 과 expo간의 협업 제품
  - xcode또는 android studio를 설치할 필요가 없음
- 준비사항
  - npm install -g create-react-native-app
    - 미리 python2와 node-gyp을 설치되어 있어야 함.
    - 별도로 설치해도 되지만 자동화된 도구 이용 가능
      - npm install -g windows-build-tools(관리자 권한으로 실행)
  - expo App 설치(Android, iOS)
    - Javascript 기반의 ios, android앱 개발과 배포를 도와주는 모바일 앱 개발 도구
    - 직접 폰에서 설치(에뮬레이터는 사용 불가)
  - 개발용 컴퓨터와 스마트폰은 동일한 무선 AP에 연결되어 있어야 함.

create-react-native-app 활용(2)

expo 활용 웹기반 개발도구

- https://sketch.expo.io
  - 웹기반의 화면에서 react-native app을 디자인하고 QR Code 생성
  - expo App으로 QRCode 스캔하여 기능 테스트
- create-react-native-app는 expo 기반으로 만들어져 있음.