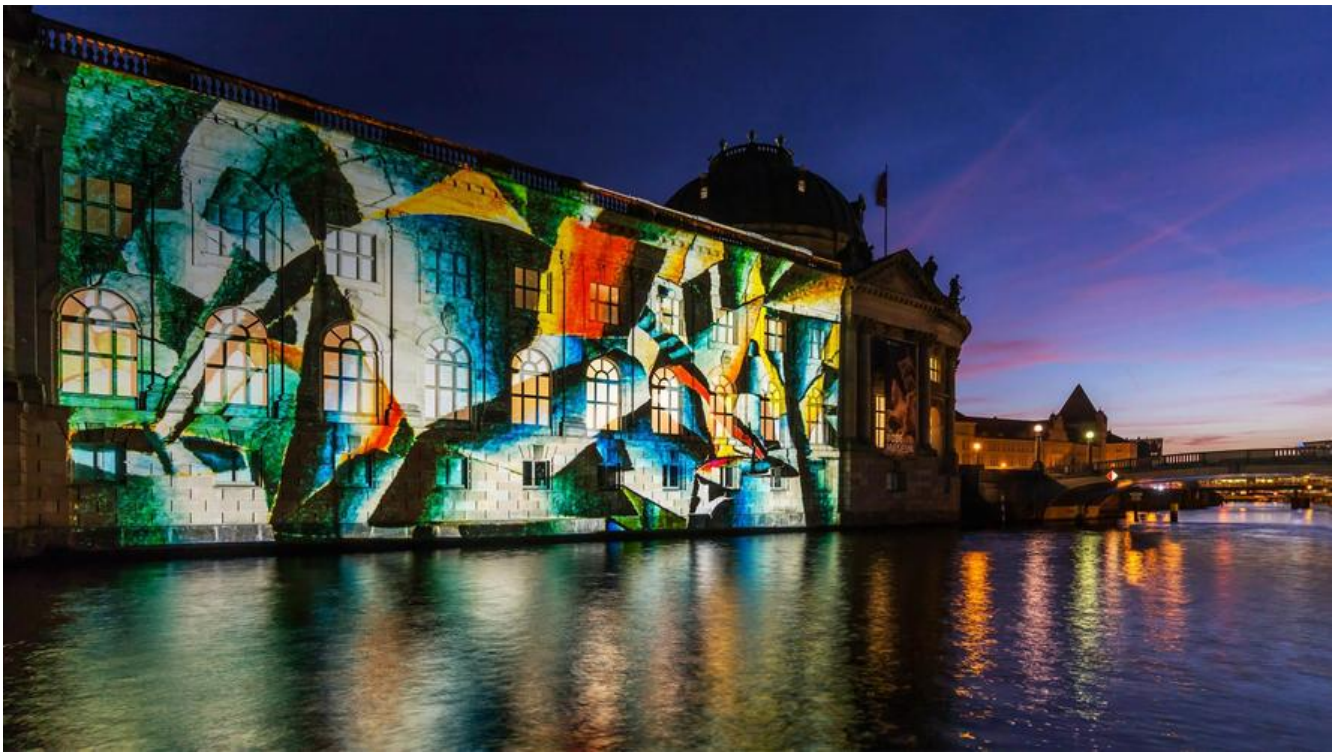
native 源码修改、优化

作者：HuixiaZhang
原文链接：https://ld246.com/article/1540985139413
来源网站：链滴



iOS

1. 本地图片资源加载闪退问题

环境： react-native 0.55.4

友盟上报错误如下：

7 libsystem_malloc.dylib 0x00000001824e0384 + 0

8 CoreFoundation 0x0000000182924ec8 + 44

9 libicucore.A.dylib 0x0000000182667ee8 utext_close + 68

10 libicucore.A.dylib 0x000000018271b3fc _ZN3icu12RegexMatcherD2Ev + 132

11 libicucore.A.dylib 0x000000018271b43c _ZN3icu12RegexMatcherD0Ev + 12

12 libicucore.A.dylib 0x00000001827690ac uplrules_select + 216

13 libicucore.A.dylib 0x0000000182769554 uregex_close + 36

14 Foundation 0x0000000183259540 + 64

15 Hebao!-[RCTImageLoader canHandleRequest:] [RCTImageLoader.m : 794]

16 Hebao!-[RCTNetworking handlerForRequest:] [RCTNetworking.mm : 209]

17 Hebao!-[RCTNetworking networkTaskWithRequest:completionBlock:] [RCTNetworking.mm 0]

18 Hebao!-[RCTImageLoader _loadURLRequest:progressBlock:completionBlock:] [RCTImageLoader.m : 0]

19 Hebao!__118-[RCTImageLoader _loadImageOrDataWithURLRequest:size:scale:resizeMode:rogressBlock:partialLoadBlock:completionBlock:]_block_invoke.181 [RCTImageLoader.m : 404]

根据报错内容和符号解析定位到代码位置

/workspace/Libraries/Image/RCTImageLoader line 802

```
NSString *query = requestURL.query;
if (query != nil && [videoRegex firstMatchInString:query
options:0
range:NSMakeRange(0, query.length)]) {
return  NO;
}
```

此处可能出现多线程问题，修改如下

```
static dispatch_once_t onceToken;
dispatch_once(&onceToken, ^{
NSError *error = nil;
videoRegex = [NSRegularExpression  regularExpressionWithPattern:@"(?:&|^)ext=MOV(?:&|$
"
options:NSRegularExpressionCaseInsensitive
error:&error];
if (error) {
RCTLogError(@"%@", error);
}
});
```

2. Reload 方法桥接

虽然react-native 提供 Live Reload, 修改代码会实时打包部署，从而页面也会随着一起更新，但是这功能开启会导致一定的卡顿，特别如果你是用WebStorm开发项目。

目前摇一摇是唯一手动触发更新的操作，但是很明显效率不高，摇起来很费劲，其实我们完全可以将reload方法桥接出来提供给js这边调用，你可以在任何地方调用此方法更新代码。

新建一个DevHelper类

RNBridge.h

```
#import <Foundation/Foundation.h>
#import <React/RCTBridgeModule.h>
#import <React/RCTEventEmitter.h>
@interface DevHelper : RCTEventEmitter<RCTBridgeModule>
@end
```

`DevHelper.m`

```
#import "DevHelper.h"
@implementation  DevHelper
```

```
RCT_EXPORT_MODULE();
- (dispatch_queue_t)methodQueue{
  return dispatch_get_main_queue();
}
RCT_EXPORT_METHOD(debugReload){
  [(RCTRootView *)[UIApplication sharedApplication].keyWindow.rootViewController.view bri
ge] reload];
}
```

JS端调用NativeModules.DevHelper.debugReload()

Android

1. compile react-native from source

环境： react-native 0.55.4

fresco 1.3.0 在png进行resize时有性能问题，经google后，发现需要改动 fresco 源码，因此将 react native 改为 arr 包进行引入

文件位置：

node_modules/react-native/android/com/facebook/react/react-native/0.55.4/react-native-0.5.4-sources.jar!/com/facebook/react/modules/fresco/FrescoModule.java

找到以下代码修改为：

```
public static ImagePipelineConfig.Builder getDefaultConfigBuilder(ReactContext context) {
HashSet requestListeners = new HashSet<>();
requestListeners.add(new SystraceRequestListener());
OkHttpClient client = OkHttpClientProvider.createClient();
// make sure to forward cookies for any requests via the okHttpClient
// so that image requests to endpoints that use cookies still work
CookieJarContainer container = (CookieJarContainer) client.cookieJar();
ForwardingCookieHandler handler = new ForwardingCookieHandler(context);
container.setCookieJar(new JavaNetCookieJar(handler));
return OkHttpImagePipelineConfigFactory
.newBuilder(context.getApplicationContext(), client)
.setNetworkFetcher(new ReactOkHttpNetworkFetcher(client))
.setDownsampleEnabled(true)
.setRequestListeners(requestListeners);
}
```

```
#### 2. Reload 方法桥接
新建一个`DevHelper`类
>```
public class DevModules extends ReactContextBaseJavaModule {
```

```
    private static final String MODULES_NAME_Dev = "DevHelper" ;
    public DevModules(ReactApplicationContext reactContext) {
        super(reactContext);
    }
    @Override
    public String getName() {
        return MODULES_NAME_Dev;
    }
    @ReactMethod
    public void reload() {
        UIManagerModule uiManager = getReactApplicationContext().getNativeModule(UIMan
gerModule.class);
        uiManager.addUIBlock(new UIBlock() {
            @Override
            public void execute(NativeViewHierarchyManager nativeViewHierarchyManager) {
                getReactNativeHost().getReactInstanceManager()
                        .getDevSupportManager().handleReloadJS();
            }
        });
    }
    protected ReactNativeHost getReactNativeHost() {
        return  MainApplication.getMyApplication().getReactNativeHost();
    }
}
```

将其加到AppReactPackage

```
public class AppReactPackage implements ReactPackage {
@Override
public List createNativeModules(ReactApplicationContext reactContext) {
List modules = new ArrayList<>() ;
modules.add(new DevModules(reactContext)) ;
return modules ;
}
@Override
public List createViewManagers(ReactApplicationContext reactContext) {
return Collections.emptyList();
}
}
```

修改`MainApplication`
>```
```
public class MainApplication extends Application implements ReactApplication {
  private static MainApplication instance;
  public static MainApplication getMyApplication() {
    return instance;
  }
```

```
  private final ReactNativeHost mReactNativeHost = new ReactNativeHost(this) {
    @Override
    public boolean getUseDeveloperSupport() {
      return BuildConfig.DEBUG;
    }
    @Override
    protected List getPackages() {
      return Arrays.asList(
          new MainReactPackage(), new LottiePackage(), new AppReactPackage()
      );
    }
    @Override
    protected String getJSMainModuleName() {
      return "index";
    }
  };
  @Override
  public ReactNativeHost getReactNativeHost() {
    return mReactNativeHost;
  }
  @Override
  public void onCreate() {
    super.onCreate();
    instance = this;
    SoLoader.init(this, /* native exopackage */ false);
  }
}
```