热更新

热更新 `codepush` `pushy`

热更新

热更新可选技术方案如下：
微软codepush：
本地codepush：
国内pushY：
自己实现更新库：

一：背景
CodePush是提供给React Native 和 Cordova 开发者直接部署移动应用更新给用户设备的云服务。CodePush 作为一个中央仓库，开发者可以推送更新到 (JS, HTML, CSS and images)，应用可以从客户端 SDKs 里面查询更新。CodePush 可以让应用有更多的可确定性，也可以让你直接接触用户群。在修复一些小问题和添加新特性的时候，不需要经过二进制打包，可以直接推送代码进行实时更新。

二：原理
把需要更新的文件上传到服务器地址，APP去下载进行安装（直接调用jslocation的jsbundle文件和assets资源文件）。

三：可使用方案对比

	微软codepush	**本地codepush**	**pushY**	**自己搭建实现**
优点	差异化更新，客户端hash检查，友好的命令行操作	访问速度快，简化操作步骤，可定制操作和功能	完善的文档和技术支持，访问速度快，简化操作步骤	代码可控，安全，功能可定制
缺点	访问速度慢，更新不及时(服务器在国外)， 无操作界面	代码更新不及时，升级困难，升级风险不可控	代码不可控，经常出现兼容性问题	技术难点大，原生和后端均需实现很多功能，成本巨大

技术难点	访问速度慢，功能强大，命令方式需自定义	需搭建平台，进行功能和页面管理，一些命令需自定义	兼容性问题严重，无区分线上线下key	后端更新代码版本管理，APP下载解压加载更新.....
参考	codepush	code-push-server	react-native-pushy	无

四：大致流程如下：
1-：配置codepush终端工具：

```
npm install code-push-cli -g
```

2-：进行账号注册(有账户可以使用login进行登录，可以使用github账号进行登录)：

```
code-push register  # 注册账号
```

3-：执行以上命令会返回一个key,在终端进行输入，即可进行登录
4-：在项目根目录下执行如下命令：

```
npm install --save react-native-code-push@latest
```

5-：添加项目到codepush平台，使用如下命令(platform值为iOS或者android)(此时返回两个值Staging和Production)：

```
code-push app add <AppName>-<platform> <platform>  react-native
```

6-：使用如下命令，可查看添加的项目，根据返回的Staging和Production的值在不同平台的不同环境进行配置(release和debug)

```
code-push app list
```

7-：执行 link操作，配置不同的平台代码：
IOS:

```
react-native link
```

会出现如下命令语句：

```
? What is your CodePush deployment key for Android (hit <ENTER> to ignore)
```

接下来，有两种操作方式：
1-：直接使用enter进行下一步：
在xcode里面，打开项目的info.plist，在CodePushDeploymentkey中输入产生的Staging和Production值。

2-：把刚才返回的Staging或者Production值复制进去：
进行回车

Android:
1-：在android/app/build.gradle文件里面检查是否有如下代码，如果没有则加上（默认如果link上的话就会有）：

```
apply from: "../../node_modules/react-native/react.gradle"
apply from: "../../node_modules/react-native-code-
push/android/codepush.gradle"
```

2-：添加配置。当APP启动时我们需要让app向CodePush咨询JS bundle的所在位置，这样CodePush就可以控制版本。更新 MainApplication.Java文件

```
import android.app.Application;
import android.util.Log;

import com.facebook.react.ReactApplication;
import com.facebook.react.ReactInstanceManager;
import com.facebook.react.ReactNativeHost;
import com.facebook.react.ReactPackage;
import com.facebook.react.shell.MainReactPackage;

import java.util.Arrays;
import java.util.List;
import com.microsoft.codepush.react.CodePush;//确定导入codepush,有时候link不上
```

```
public class MainApplication extends Application implements
ReactApplication {

  private final ReactNativeHost mReactNativeHost = new
ReactNativeHost(this) {

    @Override
    protected String getJSBundleFile() {
      return CodePush.getJSBundleFile();
    }

    @Override
    protected boolean getUseDeveloperSupport() {
      return BuildConfig.DEBUG;
    }

    @Override
    protected List getPackages() {
      return Arrays.asList(
          new MainReactPackage(),

          new CodePush("Staging或者Production值", MainApplication.this, Buil
dConfig.DEBUG)
      );
    }
  };

  @Override
  public ReactNativeHost getReactNativeHost() {
      return mReactNativeHost;
  }
}
```

3-：在android/app/build.gradle中有个 android.defaultConfig.versionName属性，我们需要把 应用版本改成 1.0.0（默认是1.0，但是codepush需要三位数）

```
android{
    defaultConfig{
        versionName "1.0.0"
    }
}
```

或者：

```
defaultConfig {
    versionName  rootProject.ext.android.versionName
}
```

这里使用了自定义文件路径，详情可参考http://blog.csdn.net/asddavid/article/details/53322689

8-：在项目中进行编码，生成差异化更新包，使用如下命令上传到不同平台(默认Staging，可以不输入)：
IOS：

```
code-push release-react <AppName>-<platform> <platform>
```

Android:
发布更新分为两种：
-1:只更新项目的JS
-2:更新图片和JS

步骤如下：
一：值更新JS
-1:在项目的根目录下面创建bundles文件夹
-2:打包修改文件，进行命名，放入指定目录下：

```
react-native bundle --platform ios --entry-file index.ios.js --bundle-outpu
t ./bundles/main.jsbundle --dev fa
```

-3:将main.jsbundle二进制文件push到platform环境中:

```
code-push release <AppName> ./bundles <Version>
```

二：更新图片和js
-1:打包命令修改如下，其他正常：

```
react-native bundle --platform ios --entry-file index.ios.js --bundle-outpu
t ./bundles/main.jsbundle --assets
```

9-：App调用更新代码进行更新。
入口文件导入：

```
import codePush from "react-native-code-push";
```

在componentDidMount中调用sync方法，当APP启动时会在后台静默更新：

```
componentDidMount(){
   codePush.sync();
}
```

或者使用代码，当App启动时会立即进行更新：

```
 codePush.sync({installMode:codePush.InstallMode.IMMEDIATE});
```

10-：使用命令查看更新安装情况

```
 code-push deployment history <appName> <deploymentNmae> 查看历史版本(Product
ion 或者 Staging)
```

五：相关命令
1-：登录

```
code-push login
```

2-：注销

```
code-push-loout
```

3-：列出登录的key

```
code-push access-key ls
```

4-：删除某个key

```
code-push access-key rm<accessKey>
```

5-：code-push相关

```
add 在账号里面添加一个新的app
remove 或者 rm 在账号里移除一个app
rename 重命名一个存在app
list 或则 ls 列出账号下面的所有app
transfer 把app的所有权转移到另外一个账号
```

6-：部署Deployment Key
App创建成功后会默认显示两个部署:Production 和Staging 。部署，简单的说就是环境，比如ReactNative的jsbundle，iOS和Android是不可以共用一个的，所以我们需要生成两个jsbundle，而我们可以通过部署这个功能，创建两个部署：AppDemo-iOS 和AppDemo-Android ，并且App中分别使用这两个部署的Key，之后我们上传jsbundle只要分别上传到这两个部署中就可以了。每个部署都有一个对应的Deployment Key ，需要在项目中使用对应的Key。

```
code-push deployment add <appName> <deploymentName>
```

7-：列出所有部署

```
code-push deployment ls <appName>
```

8-：Deployment相关命令

```
code-push deployment rename <appName> 重命名
code-push deployment rm <appName> 删除部署
code-push deployment ls <appName> 列出应用的部署情况
code-push deployment ls <appName> -k 查看部署的key
code-push deployment history <appName> <deploymentNmae> 查看历史版本(Producti
on 或者 Staging)
```

六：参考

1-：可参考谷歌diff算法执行增量更新，目前pushy已经实现

2-：回滚更新

```
code-push rollback <appName> <deploymentName>
--targetRelease, -r 指定回归到哪个标签，默认是回滚到上一个更新 [string] [默认值: null]

示例：
code-push rollback MyApp Production "MyApp"中"Production"部署执行回滚
code-push rollback MyApp Production --targetRelease v4
"MyApp"中"Production"部署执行回滚，回滚到v4这个标签版本
```

3-：促进更新

```
code-push promote <appName> <sourceDeploymentName> <destDeploymentName>

-description, -des 描述 [string] [默认值: null]
-disabled, -x 该促进更新，客户端是否可以获得更新 [boolean] [默认值: null]
-mandatory, -m 是否强制更新 [boolean] [默认值: null]
-rollout, -r 此促进更新推送用户的百分比 [string] [默认值: null]

示例：
code-push promote MyApp Staging Production
"MyApp" 中"Staging" 部署的最新更新发布到"Production" 部署中

code-push promote MyApp Staging Production -des "Production rollout" -r 25
"MyApp" 中"Staging" 部署的最新更新发布到"Production" 部署中，并且只推送25%的用户
```

4-：修改更新

```
code-push patch <appName> <deploymentName>

--label, -l 指定标签版本更新，默认最新版本 [string] [默认值: null]
--description, --des 描述 [string] [默认值: null]
--disabled, -x 该修改更新，客户端是否可以获得更新 [boolean] [默认值: null]
--mandatory, -m 是否强制更新 [boolean] [默认值: null]
--rollout, -r 此更新推送用户的百分比，此值仅可以从先前的值增加。 [string] [默认值: null]

示例：
```

```
code-push patch MyApp Production --des "Updated description" -r 50 修改"MyApp"的"Production"部署中最新更新的描述 ， 并且更新推送范围为50%
code-push patch MyApp Production -l v3 --des "Updated description for v3" 修改"MyApp"的"Production"部署中标签为v3的更新的描述
```

5-：本地codepush环境搭建，可参考：

https://github.com/lisong/code-push-server