JReap4.0 移动应用开发框架

使用手册

文件控制	■受控□不受控				
文档编号				版本号	1.0
分册名称				第 1 册/共 1 册	
总页数		正文		附录	
编制		审批		生效日期	

湖南大唐先一科技有限公司

1. 概述

1.1. 版本

JReap4.0 移动应用开发框架。

1.2. 发布时间

2017 年 1 月 3 日。

1.3. 阅读对象

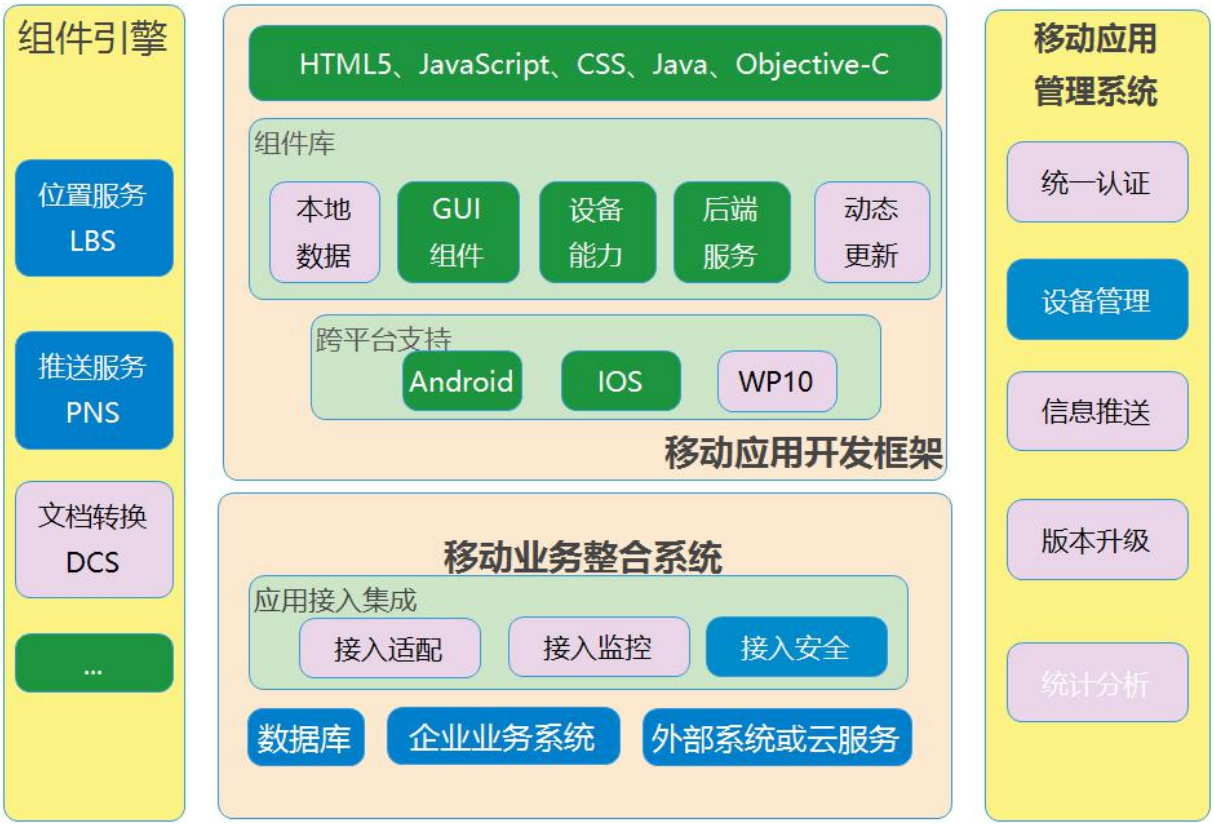
架构师、分析设计工程师和软件开发工程师。

1.4. 背景

在移动互联网的大背景下，H5、APP 大行其道，客户对移动端 APP 需求愈加强烈，结合公司实际需要，亟需开发一个移动开发框架来满足客户单位移动端的快捷开发。

1.5. 功能特性

主要内容如下图所示：



如上图所示，移动应用开发框架主要包含以下功能：

1） 开发 android、iOS 版本原生应用基础程序包

2） 基于 webview+cordova+angular+ionic 方式提供混合应用开发基础框架提供 javascript 与设备交互接口：GPS 定位及数据上报、二维码扫描、拍照、图片上传、水印处理、消息推送、录音、本地数据库访问

3） 提供原生应用开发 API 接口：http 异步请求、图片处理、本地 sqllite 存储、缓存框架、本地后台服务、定时服务、通讯录组件、地图展现

4） 提供微应用管理功能；实现了上传、分发、下载、更新、安装及卸载

5） 提供设备及安全管理功能；实现了服务日志、登录日志、设备管理

2. 开发及运行环境

2.1. 开发环境

开发工具：Xcode、EcLipse、JReap Studio

开发语言：Objective-C、Java、HTML5；

Jdk：jdk 1.7；

界面框架： ionic+angular、android，iOS 原生界面。

2.2. 运行环境

Android：最佳适配的 android 版本为 4.1+；

iOS：最佳适配的 iOS 版本为 iOS8+；

支持的浏览器版本：IE9(+),chrome 推荐在 chrome 中使用；

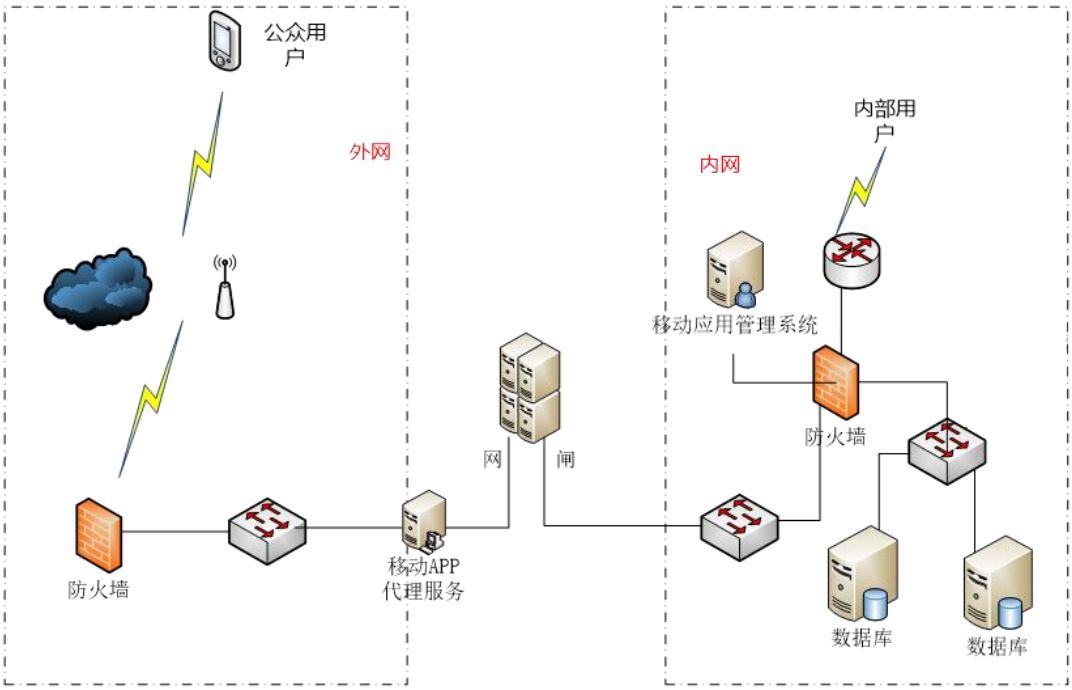
Web 服务器：Tomcat，WebLogic，WebSphere；

应用服务器：Tomcat，WebLogic，WebSphere；

数据库：Oracle、sqlserver、db2、mysql 等主流关系型数据库。

3. 总体结构

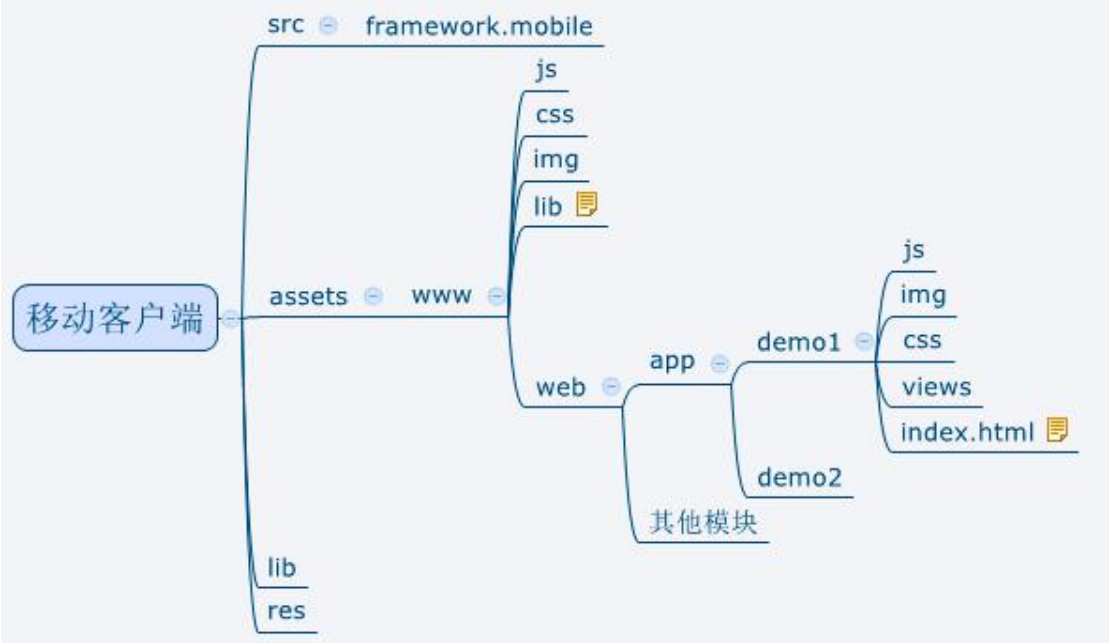
3.1. 网络拓扑结构



3.2. 目录结构

3.2.1. 移动客户端

移动客户端目录结构如下图所示：



说明：

Android

- src 是源码目录，平台提供的源码包在 framework.mobile 包目录中。
- lib 是 jar 包目录，引用的 jar 包放在此目录
- res 目录是资源文件（布局文件，图片等）
- www 是 html 相关文件目录：

  ① js 存放通用 js 文件

  ② css 存放通用 css 文件

  ③ img 存放图片文件

  ④ lib 存放第三方库，如：ionic，angular，cordova 等

  ⑤ web 存放具体功能网页，其中微应用默认放在 app 目录下

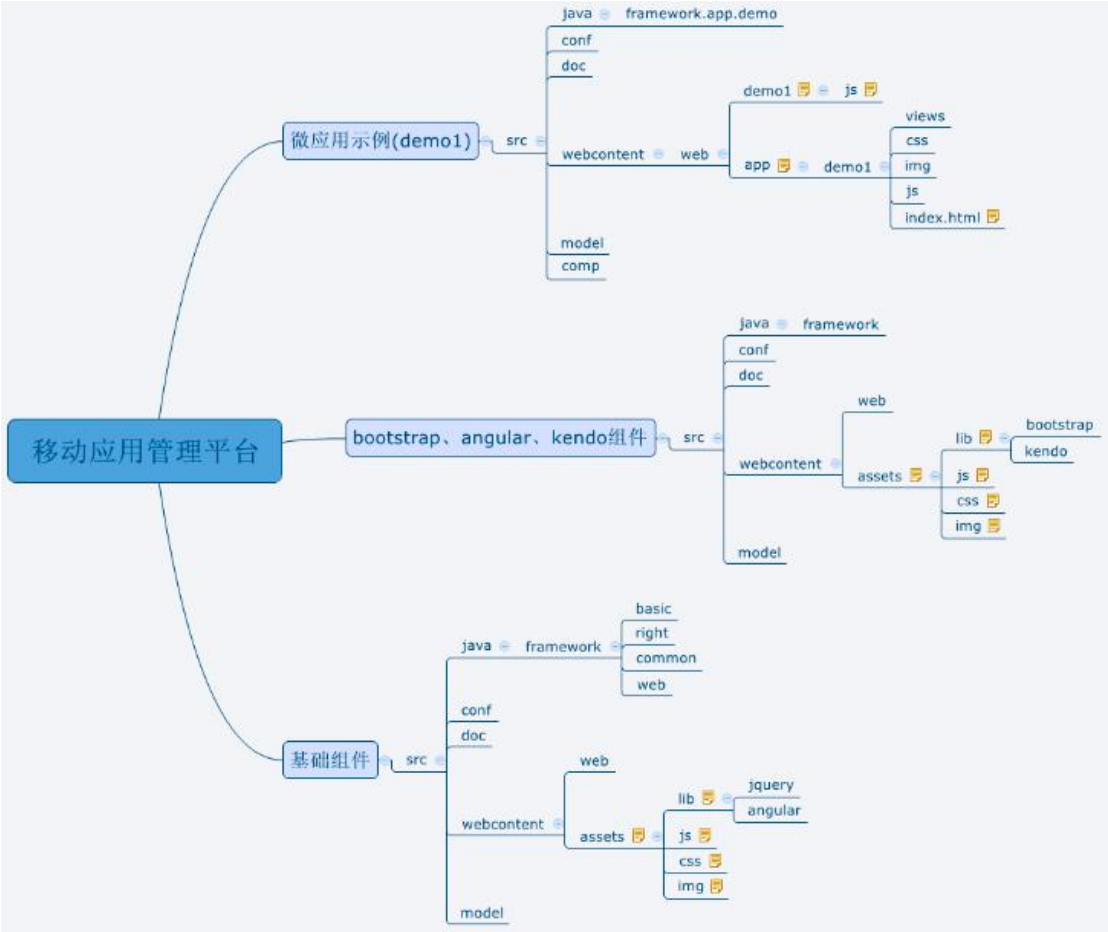
  ⑥ demo1 是微应用目录结构示例，js 存放逻辑代码，img 存放图片，css 存放模块样式文件，views 存放页面片段，index.html 是默认功能主页

iOS

iOS 端 www 目录和 android 是一致的，源码等其他目录遵循 iOS 开发约定

3.2.2. 移动应用管理平台

移动应用管理平台目录结构如下图所示：

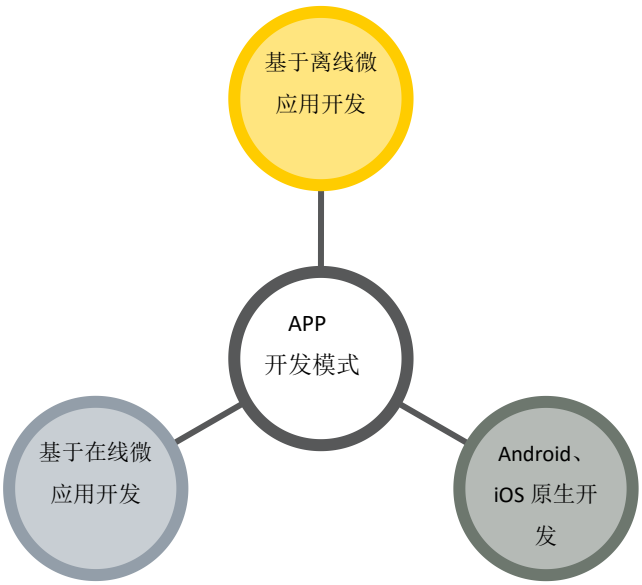


说明：

- 微应用示例遵循组件开发目录
- java 是源码目录
- conf 是配置文件目录（spring 配置文件，sql 脚本）
- doc 是文档目录
- webcontent 是前端目录，微应用统一放在 web/app 目录下
- model 是模型目录
- comp 是组件描述文件

4. 标准开发过程

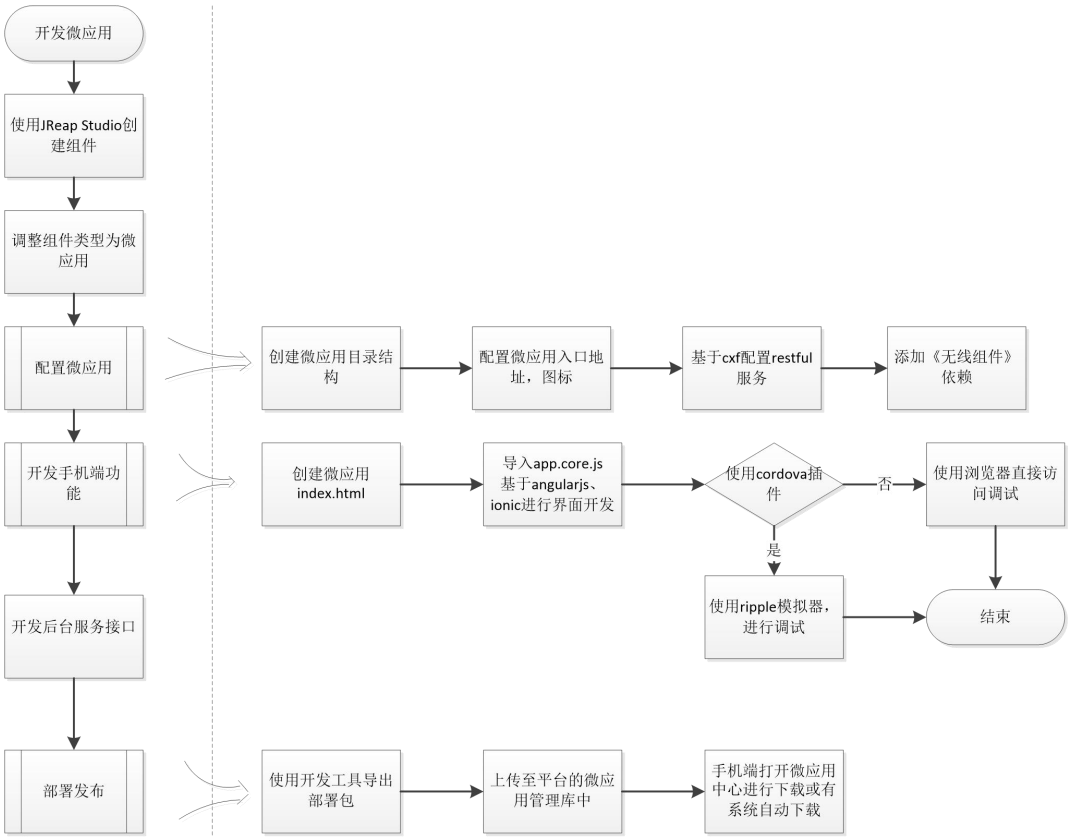
本平台提供多种技术手段，方便具体业务开发部门进行 APP 功能的实现。



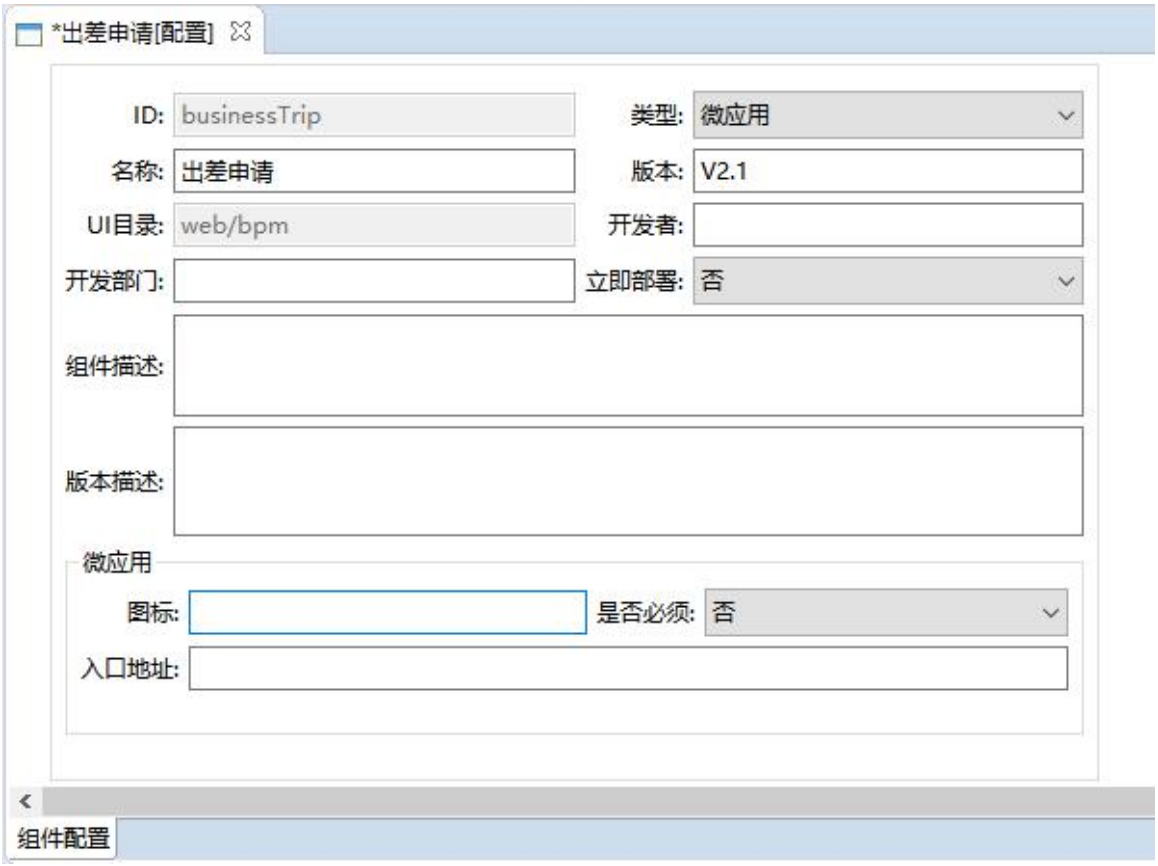
基于在线微应用开发：在线方式即 mobile web 应用开发，平台的 app 替代了浏览器，提供了应用访问的统一入口。
基于离线微应用开发：基于平台提供的 javascript 及 cordova 插件库，及相关约束进行开发，本文档将重点阐述如何使用此方式开发。
基于 Android、iOS 原生开发：基于平台提供的 Java、Objective-C 库进行开发。

4.1. 微应用开发

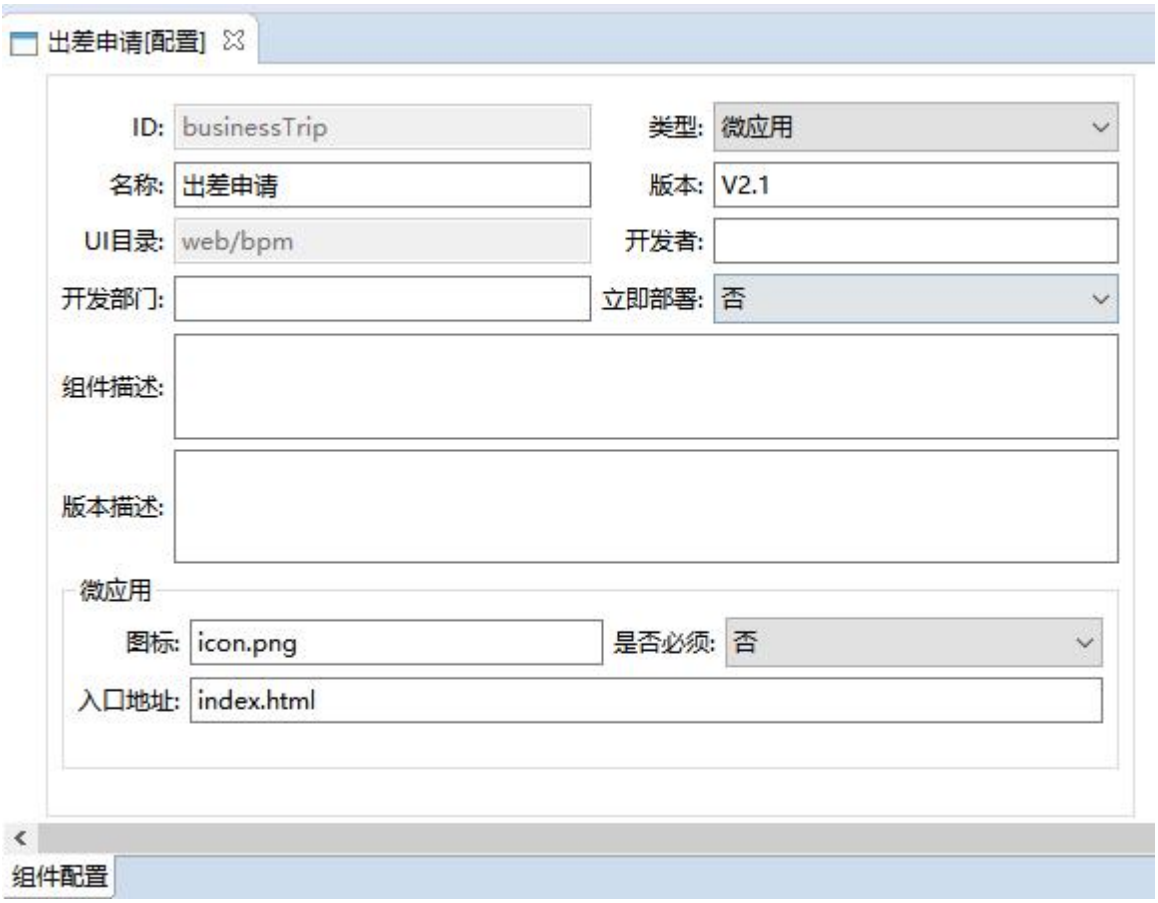
离线和在线应用开发流程基本一致，其开发流程如下：


开发微应用
使用JReap Studio创建组件
调整组件类型为微应用
配置微应用
创建微应用目录结构
配置微应用入口地址，图标
基于cxf配置restful服务
添加《无线组件》依赖
开发手机端功能
创建微应用index.html
导入app.core.js基于angularjs、ionic进行界面开发
使用cordova插件
否
使用浏览器直接访问调试
是
使用ripple模拟器，进行调试
结束
开发后台服务接口
部署发布
使用开发工具导出部署包
上传至平台的微应用管理库中
手机端打开微应用中心进行下载或有系统自动下载

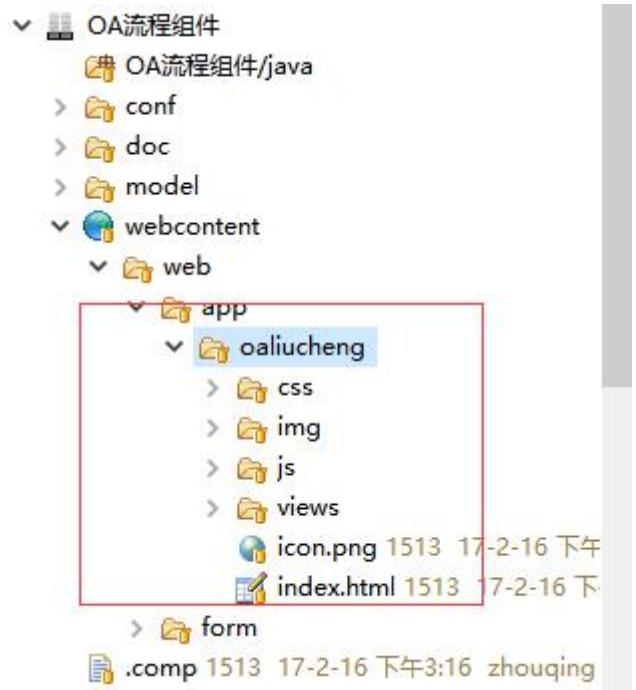
4.1.1. 创建组件及组件配置



4.1.2. 配置微应用



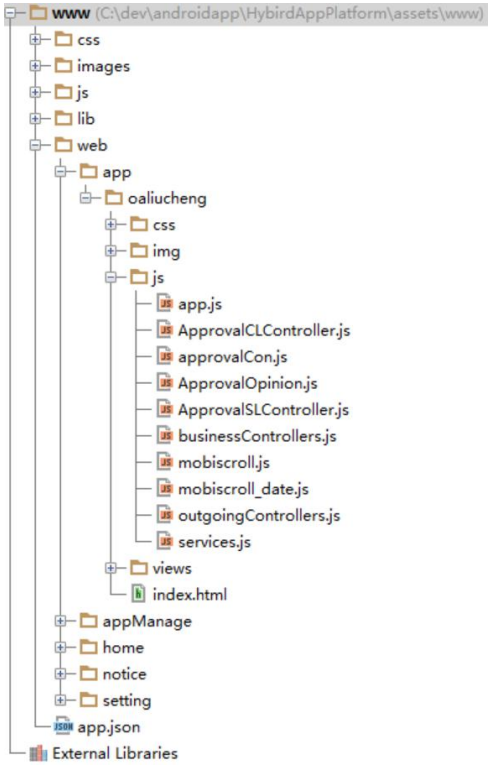
创建微应用目录结构，在 JReap Stuido 中目录结构如下，该目录结构在部署到，手机端时，呈现的目录结构有所不同。



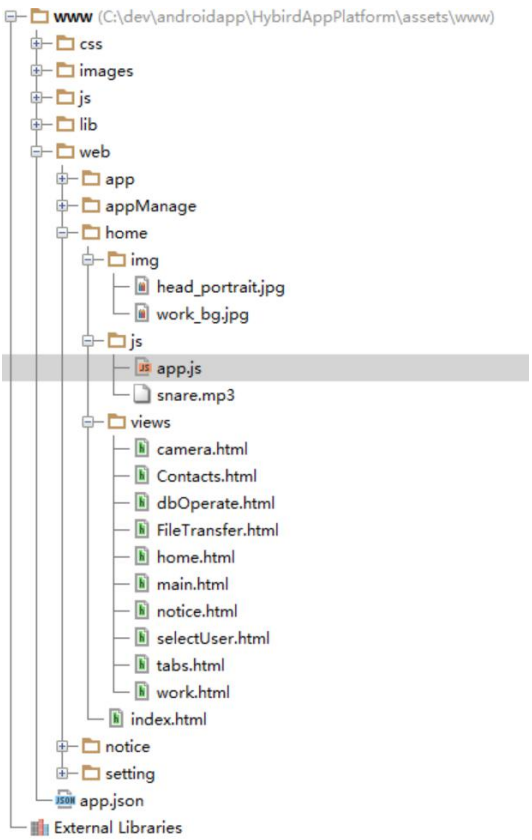
4.1.3. 手机端开发

微应用采用的是 Ionic+Angular.js 架构开发，类似 MVC 模式，分别包含：模型(model)，视图(view)，控制器(controller)。Ionic 封装了视图模块，对界面常用的布局和控件进行了简单的封装，方便在开发中使用。Angular.js 提供了模型和控制器，另外还对整个框架中的服务器部分与公共部分的处理，继承了 jquery 的大部分功能，并且是双向数据绑定的特点。

微应用的模块安装到手机端时，其目录结构如下图：



其中 css、image、js、lib 是移动平台的基础，微应用的开发模块放置与 web 目录下。用户根据项目的需要在 web 目录下增加新的微引用。下面介绍微应用开发，以 web 下的 home 为例。Home 为登陆进入平台的第一个页面，结构如下图：

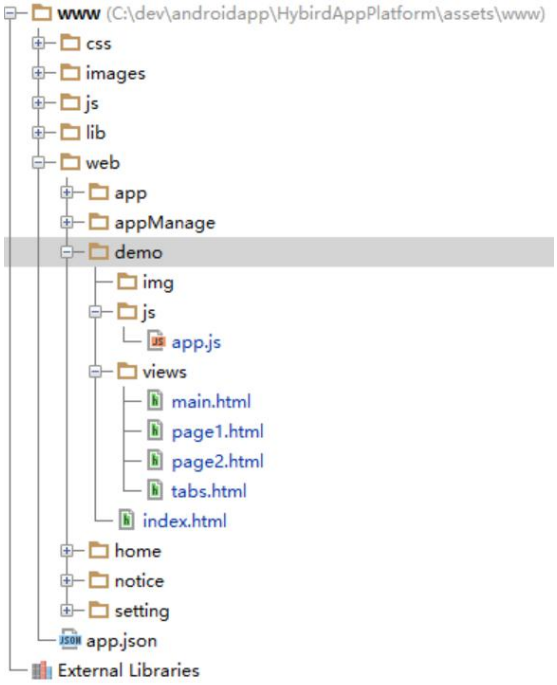


Home 中包含了 img、js、views、index.html。

1. index.html 为微应用的框架，这个页面作为载体，加载 views 中的其他页面，并且负责页面的切换，其他公共的引用部分都在这个页面中加载，实现全局的调用。
2. img 目录中是微应用中使用的图片资源
3. js 目录下放的是 js 文件，移动平台使用了 angular.js 开放方式。我们在是使用中可以创建一个 js 文件（app.js），用于定义微应用的模块，公共插件加载，路由设置，模型，控制器，服务。
4. views 目录下是微应用的每个功能模块页面。通过 Ionic 变成方式开发，也可以使用 H5 的代码。通过 app.js 中的路由功能实现页面的展示和切换。
5. 下面通过一个 Demo 来说明创建一个新的微应用的过程。实例的应用的过程是，打开一个微应用，首先进入的主页 main.html，所有的页面下面都有导航栏，导航分别是：首页，页面 1，页面 2。点击导航可以切换页面。每个页面上有各自的功能。

   1） 在 web 目录下创建 demo 文件夹，在 demo 文件夹下创建 img 目录、js 目

录、views 目录、index.html。在 views 目录下新建 tab.html、main.html、page1.html、page2.html，在 js 目录下创建 app.js 文件。如下图。



2) 打开 app.js，创建路由规则，设置页面导航（tabs.html），首页(main.html)，页面 1(page1.html),页面 2（page2.html），页面路由和页面对应的控制器。如下图。

```
define(["JReapApp","ionic"],function(JReapApp) {
    JReapApp.config(function ($stateProvider,$urlRouterProvider) {
        //导航栏设定
        $stateProvider
            .state("tab", {
                url: "/tab",
                abstract: true,
                templateUrl: "views/tabs.html"
            })
        //首页设定
            .state('tab.main', {
                url: '/main',
```

```
            views: {
                'tab-main': {
                    templateUrl: 'views/main.html',
                    controller: "mainCtrl"
                }
            }
        })
    //第一页设定
        .state('tab.page1', {
            url: '/page1',
            views: {
                'tab-page1': {
                    templateUrl: 'views/page1.html',
                    controller: "page1Ctrl"
                }
            }
        })
    //第二页设定
        .state('tab.page2', {
            url: '/page2',
            views: {
                'tab-page2': {
                    templateUrl: 'views/page2.html',
                    controller: "page2Ctrl"
                }
            }
        })
    //进入默认为首页
    $urlRouterProvider.otherwise("/tab/main");
});
```

```
    //首页控制器
    JReapApp.controller('mainCtrl', function($scope) {
    });
    //第一页控制器
    JReapApp.controller('page1Ctrl', function($scope) {
    });
    //第二页控制器
    JReapApp.controller('page2Ctrl', function($scope) {
    });
});
```

在 app.js 中，需要引用 JReapApp 模块，定义路由规则，每个页面需要有一个控制器。

3) 打开 index.html 页面，定义基础框架，代码如下

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-scalable=no, width=device-width">
    <script src="../../js/app.core.js"></script>
    <script type="text/javascript">
        JReap.load.include("css/ui-dialog.css", "ui-dialog");
        JReap.load.include("css/extend-style.css", "extendStyle");
        require(["asssets/demo/js/app"],function(app){
            angular.bootstrap(document, ['JReapApp']);
        });
    </script>
</head>
<body>
```

```
<ion-nav-bar>
</ion-nav-bar>
<ion-nav-view></ion-nav-view>

</body>
</html>
```

其中 ion-nav-bar 为导航栏区域，ion-nav-view 为显示区域。

4) 打开 tabs.html，设置导航栏，代码如下：

```
<ion-tabs class="tabs-icon-top">
    <ion-tab title="主 页 " icon-on="ion-ios-home" icon-off="ion-ios-home-outline"
ui-sref="tab.main">
        <ion-nav-view animation="slide-left-right" name="tab-main"></ion-nav-view>
    </ion-tab>
    <ion-tab title="页 面 1" icon-on="ion-ios-cloud" icon-off="ion-iOS-cloud-outline"
ui-sref="tab.page1">
        <ion-nav-view animation="slide-left-right" name="tab-page1"></ion-nav-view>
    </ion-tab>
    <ion-tab title="页面 2" icon-on="ion-ios-bookmarks" icon-off="ion-ios-bookmarks-outline"
ui-sref="tab.page2">
        <ion-nav-view animation="slide-left-right" name="tab-page2"></ion-nav-view>
    </ion-tab>
</ion-tabs>
```

导航栏中，要注意 ion-nav-view 的 name 属性，这个属性需要与 app.js 中路由属性对应，才能正确的将页面显示在对应的显示区域，ui-sref 类似 html 中的 a 标签属性 href,用于页面跳转执行,也需要对应 app.js 路由中定义的 state 属性。

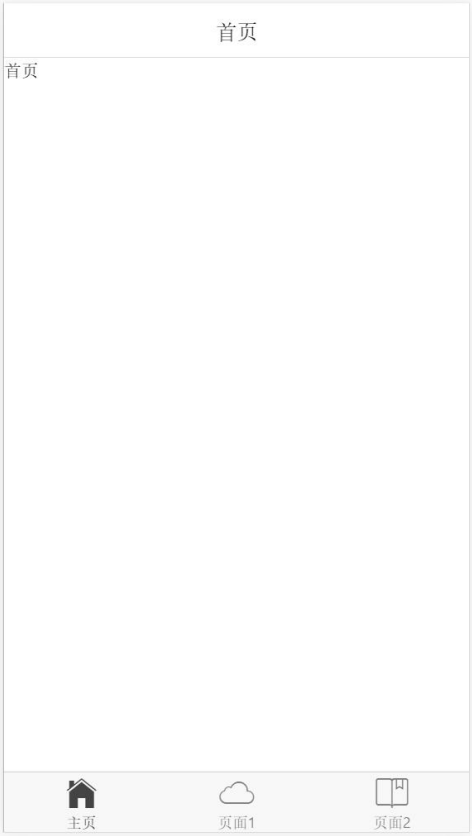
5) 打开 main.html，设置页面业务功能，代码如下：

```
<ion-view view-title="首页">
    <ion-content >
        这是首页
```

```
    </ion-content>
</ion-view>
```

页面的内容写正在 ion-content 中。另外两个页面也是通过标签 ion-view 和 ion-content 进行设置。

6) 运行项目，用 google 浏览器打开 index.html 页面，效果如下



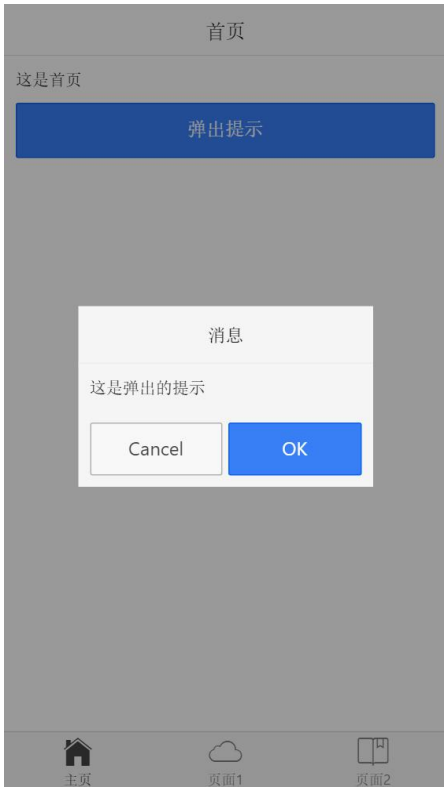
7) 控制器的运用，在 main.html 中定义一个按钮，并且通过 ng-click 属性定义一个点击事件，在 maincontrol 控制器中弹出一个提示框。

```
<ion-view view-title="首页">
    <ion-content padding="true">
        <p>
            这是首页
        </p>
        <a class="button button-block button-positive" ng-click="showmsg()">弹出提示</a>
    </ion-content>
</ion-view>
```

通过 a 标签中的 class=“button”定义了一个 ionic 按钮，ng-click 为 ionic 的点击事件，时间可以在控制器中截获并处理，控制器中的弹出提示框，下面是控制器中定义的 showmsg 方法。

```
JReapApp.controller('mainCtrl', function($scope,$ionicPopup) {
        $scope.showmsg = function(){
            var confirmPopup = $ionicPopup.confirm({
                title: '消息',
                template: '这是弹出的提示'
            });
        }
    });
```

运行点击按钮后的效果如下图，



8） 表格的使用，打开 page1.html 页面，在页面中显示联系人信息，做一个类似手机通讯，代码如下

```
<ion-view view-title="page1">
    <ion-content>
        <div class="list" padding="true">
```

```
            <div class="item item-divider">
                通讯录
            </div>
            <a class="item item-icon-left" href="#" ng-repeat="item in items">
                <i class="icon ion-person-stalker"></i>
                {{item.name}}
                <span class="item-note">
                  {{item.telephone}}
                </span>
            </a>
        </div>
    </ion-content>
</ion-view>
```

page1.html 页面使用 div 的 class 属性“list”做列表，a 标签做项目，标签中使用 ng-repeat 做循环，循环控制器中定义的 json 数据 items，将人员信息的名称和电话号码展示出来，效果如下图.

控制器中的代码通过$scope 定义了一个 json 数据 items,在 page1.html 页面可以直接使用 items 做循环。

```
JReapApp.controller('page1Ctrl', function($scope) {
        $scope.items = [
            {"name":"张三","telephone":"18911111111"},
            {"name":"李四","telephone":"18922222222"},
            {"name":"王五","telephone":"18933333333"},
            {"name":"赵六","telephone":"18944444444"}
        ]
    });
```

9) 调用原生代码。移动平台封装了一些接口可以直接通过 cordova 直接调用进行使用，用户也可以调用自己编写的原生代码进行处理，下面通过 page2.html 中的按钮事件，说明如何使用 cordova 调用原生的代码的过程。这个演示过程是页面有一个按钮，通过点击按钮实现调用安卓手机上的代码获取内部存储空间的过程。首先打开 page2.html 页面，新增一

个按钮，页面代码如下

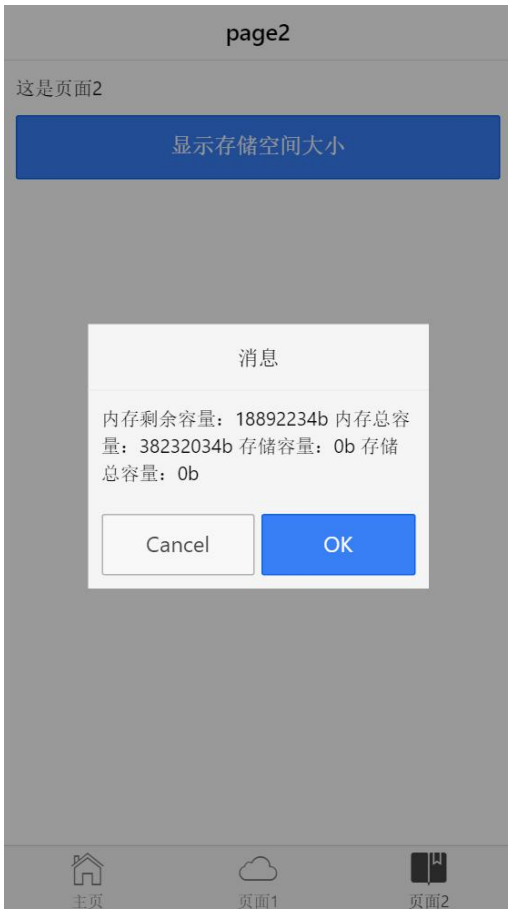
```
<ion-view view-title="page2">
    <ion-content >
        <p>
            这是页面 2
        </p>
        <a class="button button-block button-positive" ng-click="showromsize()">显示存储空间大小</a>
    </ion-content>
</ion-view>
```

页面中 a 标签定义了点击事件 showromsize 方法，通过控制器使用 cordova 调用原生类 SystemUtil 中的 getsdandromsize 方法，调用方式如下

```
JReapApp.controller('page2Ctrl', function($scope) {
        $scope.showromsize = function(){
            cordova.exec(function (res) {
                var ret =    JSON.parse(res);
                var roma = ret.romavailableSize;
                var romt = ret.romtotalsize;
                var sda = ret.sdavailableSize;
                var sdt = ret.sdtotalsize;
                var str = "内存剩余容量：" + roma + "b  内存总容量：" + romt + "b  存储容量：" + sda + "b  存储总容量：" + sdt+"b";
                var confirmPopup = $ionicPopup.confirm({
                    title: '消息',
                    template: str
                });
            }, function (err) {
                //获取不到登录信息
                alert(err)
```

```
        }, "SystemUtil", "getsdandromsize", []);
    }
});
```

结合 main.html 弹出对话框的方法，获取到移动设备的存储空间大小等数值，作为提示内容弹出显示，效果如下图



4.1.4. 服务端接口开发

服务端采用 Restful 风格开发 springMVC API 接口，这里以消息模块为例介绍接口开发过程

1） 定义服务接口 IMessageRestWebService

2） 定义接口方法

代码示例如下：

```
package framework.mobile.rest.service;


import javax.ws.rs.BeanParam;
```

```
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;

import org.apache.cxf.jaxrs.model.wadl.Description;

import framework.mobile.domain.PushMessage;
import framework.mobile.vo.MsgTaskVO;
import framework.mobile.vo.PushMessageVO;
import framework.right.webservice.vo.PageResultVO;
import framework.right.webservice.vo.ResultVO;

@Path("/messageRestWebService")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public interface IMessageRestWebService
{

    /**
     * save_message:发布通知
     * @param pushMessagevo
     * @return
     */
```

```
    @POST
    @Path("/save_message")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Description("发布通知")
    public       ResultVO<PushMessage>       saveMessage(@BeanParam
PushMessageVO pushMessagevo);

    /**
     * 删除消息接收人
     * @param msgId 消息ID
     * @param accountId 账号ID
     * @return
     */
    @Path("/recipient/{msgId}/{accountId}")
    @DELETE
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Description("删除消息接收人")
    public   ResultVO<Object>   deleteRecipient(@PathParam("msgId")
String msgId, @PathParam("accountId") String accountId);

    /**
     * 基于ID获取消息信息
     * @param id
     * @return
     */
    @GET
    @Path("/message/{id}")
    @Description("基于ID获取消息信息")
    public   ResultVO<PushMessage>   getMessageById(@PathParam("id")
String id, @QueryParam("account_id") String accountId);
```

```
    /**
     * getMessageByPage:分页查询消息列表. <br/>
     * 接受的消息
     * @param page 当前页
     * @param pagesize 每页显示条数
     * @param type 通知类别 1：我接收的 2：我发布的
     * @return
     */
    @GET
    @Path("/message_list/{account_id}")
    @Description("基于帐号ID获取消息信息")
    public PageResultVO<PushMessageVO>
getMessageByAccount(@PathParam("account_id") String accountId,
@QueryParam("page") int page, @QueryParam("pagesize") int pagesize,
            @QueryParam("type") @DefaultValue("1") int type);

    @GET
    @Path("/unread_msg_num/{account_id}")
    @Description("基于帐号ID获取未读消息数量")
    public ResultVO<Object>
queryUnreadMsgNum(@PathParam("account_id") String accountId);

    /**
     * getMessageByPage:分页查询消息列表. <br/>
     * @param pagesize 每页显示条数
     * @return
     */
    @GET
    @Path("/msg_task_list/{account_id}")
```

```
    @Description("基于帐号ID获取消息和任务信息")
    public                                    PageResultVO<MsgTaskVO>
getMsgTaskByAccount(@PathParam("account_id")   String   accountId,
@QueryParam("pagesize") int pagesize);

}
```

3） 创建接口实现 MessageRestWebService，实现接口的方法。

注：服务实现类需要定义@Service 注解，并用@Resource 注解来进行依赖注入；查询方法一般不需要加入事务控制，增加、修改、删除等需要加入事务控制的方法使用@Transactional 注解。

代码示例如下：

```
package framework.mobile.rest.service.impl;

import java.util.LinkedHashMap;
import java.util.Map;

import javax.annotation.Resource;
import org.apache.commons.lang.StringUtils;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import framework.common.dao.PaginationObject;
import framework.common.dao.PaginationParam;
import framework.mobile.domain.PushMessage;
import framework.mobile.dto.MessageDTO;
import framework.mobile.rest.service.IMessageRestWebService;
import framework.mobile.service.IPushMessageService;
import framework.mobile.vo.MsgTaskVO;
import framework.mobile.vo.PushMessageVO;
```

```
import framework.right.webservice.basic.AbstractBasicRestfService;
import framework.right.webservice.vo.PageResultVO;
import framework.right.webservice.vo.ResultVO;

@Service
public class MessageRestWebService extends AbstractBasicRestfService
implements IMessageRestWebService
{

    @Resource
    private IPushMessageService<PushMessage, String>
pushMessageService;

    @Override
    @Transactional
    public ResultVO<PushMessage> saveMessage(PushMessageVO
pushMessagevo)
    {
        MessageDTO msg = new MessageDTO(pushMessagevo);
        boolean ok = this.pushMessageService.saveMessage(msg);
        if (!ok)
        {
            return new ResultVO<PushMessage>(FLAG_FAILED, null, "发通
知失败");
        }
        return new ResultVO<PushMessage>();
    }

    @Override
    @Transactional
```

```
    public ResultVO<PushMessage> getMessageById(String id, String
accountId)
    {
        PushMessage pushMessage =
this.pushMessageService.getById(id);
        if (pushMessage == null)
        {
            return new ResultVO<PushMessage>("error", null, "找不到记
录");
        }
        if ((pushMessage.getAccount() != null
&& !accountId.equals(pushMessage.getAccount().getId())) &&
pushMessage.getRecipientsId().indexOf(accountId) < 0)
            return new ResultVO<PushMessage>("error", null, "找不到记
录");
        if (StringUtils.isNotBlank(accountId))
        {
            pushMessageService.updateMsgRecStatus(id, accountId);
        }
        return new ResultVO<PushMessage>(FLAG_SUCCESS, pushMessage,
null);
    }

    @Override
    public PageResultVO<PushMessageVO> getMessageByPage(int page,
int pagesize)
    {
        Map<String, Object> conditions = new LinkedHashMap<String,
Object>();
        PaginationParam param = new PaginationParam(page, pagesize);
```

```
        PaginationObject<PushMessageVO>  paginationObject  =  new
PaginationObject<PushMessageVO>();
        paginationObject                                          =
this.pushMessageService.getMessagePaginationObjectByParams4HQL(co
nditions, param);
        return new PageResultVO<PushMessageVO>(paginationObject);
    }

    @Override
    public  PageResultVO<PushMessageVO>  getMessageByAccount(String
accountId, int page, int pagesize, int type)
    {
        PaginationParam param = new PaginationParam(page, pagesize);
        PaginationObject<PushMessageVO>  paginationObject  =  new
PaginationObject<PushMessageVO>();
        paginationObject                                          =
this.pushMessageService.getMessagePaginationObjectByRecipientId(a
ccountId, param, type);
        //TODO 列表显示需要减少返回数据 ， 降低流量 排除掉详细字段
        return new PageResultVO<PushMessageVO>(paginationObject);
    }

    @Override
    public    PageResultVO<MsgTaskVO>    getMsgTaskByAccount(String
accountId, int pagesize)
    {
        Map<String, Object> conditions = new LinkedHashMap<String,
Object>();
        conditions.put("accountId", accountId);
        PaginationParam param = new PaginationParam(0, pagesize);
```

```
        PaginationObject<MsgTaskVO>          pageObject          =
this.pushMessageService.getMsgTaskByAccount(conditions, param);
        return new PageResultVO<MsgTaskVO>(pageObject);
    }

    @Override
    public ResultVO<Object> deleteRecipient(String msgId, String
accountId)
    {
        this.pushMessageService.deleteRecipient(msgId, accountId);
        return new ResultVO<Object>(FLAG_SUCCESS, null, null);
    }

    @Override
    public ResultVO<Object> queryUnreadMsgNum(String accountId)
    {
        int num = pushMessageService.queryUnreadMsgNum(accountId);
        return new ResultVO<Object>(FLAG_SUCCESS, num, null);
    }
}
```

4) service 和 dao 层开发和传统 SpringMvc 一样，这里不再阐述

5) 在对应 spring 配置文件中配置接口服务

```
<!-- 定时提醒待办任务数量end -->
<jaxrs:server id="mapiWebService" address="/mapi">
    <jaxrs:serviceBeans>
        <ref bean="mobileRestWebService" />
        <ref bean="messageRestWebService" />
        <ref bean="systemRestWebService" />
        <ref bean="questionRestWebService" />
        <ref bean="weChatRestWebService" />
    </jaxrs:serviceBeans>
    <jaxrs:providers>
        <ref bean="jacksonProvider" />
        <ref bean="dateParameterConverterProvider" />
```

6) 接下来在移动端可以测试调用已开发的接口，整个过程完成

4.1.5. 部署发布

4.1.5.1. 微应用管理

开发人员用 JReap Studio 开发的微应用，可在移动应用管理后台管理。

微应用管理包括上传、更新、删除等操作。

1、添加。管理员开发完微应用后打包，通过后台的的管理页面，上传微应用，上传后会产生一个微应用标识，和版本号，其中应用标识用于标识微应用，这个标识不是主键 id，同一个微应用可以有多条记录，标识相同，只是版本号不一样，用版本号来加以区分，版本号供升级时使用，初始版本号为 1。后台添加微应用后，手机 App 端便可添加及使用，后台添加的微应用分为系统微应用和用户可选微应用，系统微应用不需要用户自行添加，默认所有用户都已添加并可使用，用户可选微应用，需要用户自行添加到微应用中心后才可以使用。

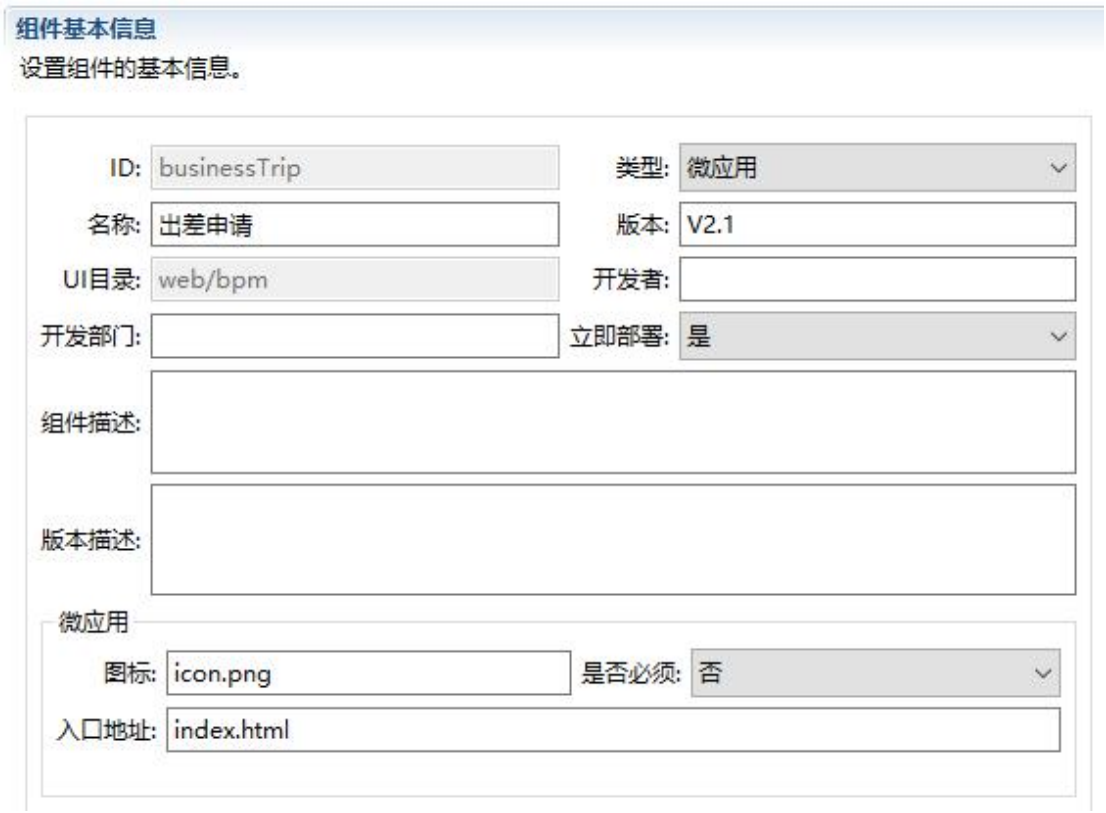
2、更新（升级）。更新微应用会新上传一个微应用包，标识相同，版本号在原版本号基础上加 1。

3、删除。对于过旧的老版本，可以予以删除。

4、列表展示。展示所有微应用列表，按微应用分组，组下面展示不同的版本号。

4.1.5.2. 微应用导出

用 JReap Studio 开发完微应用后，要先配置微应用相关信息，再导出，相关配置示例如下图所示：



4.2. 基于 android、iOS 原生开发

采用原生方式开发，即在平台的技术架构及源码基础之上进行个性化定制，该模式要求开发人员掌握 android、iOS 提供的 API 接口，门槛较高，在技术储备不足时，不推荐此种方案开发 APP。

4.2.1. 手机端定制

为满足定制化需求，开发人员需要了解整个应用的代码结构，特对主要功能做以下简要介绍：

4.2.1.1. 启动页

Android

SplashScreenActivity.java，对应的布局文件是 activity_splash_screen.xml

说明：

启动页会检测应用更新，有更新时，会弹出下载更新提示，对应处理逻辑在该类中。

iOS

LaunchScreen.storyboard

4.2.1.2. 登录页

Android

LoginActivity.java，对应的布局文件是 activity_login.xml

说明：

登录的接口对应文件 LoginHttpRunner.java

IOS

LoginViewController.xib

4.2.1.3. 主页

Android

MainActivity.java，对应的布局文件是 activity_main.xml

说明：

- 主页是 Fragment+ViewPager 实现的，底部有五个导航，分别是：首页，消息，工作，通讯录，我。其中工作和通讯录模块是原生页面，其他是网页。
- 第一次或者更新应用进入主页时，会执行 AssetsInitTask 任务。
- 进入主页还会检测自动更新的微应用。
- 首页是网页（单页面应用），对应的文件目录是 www/web/home
- 页面布局放在 views 目录
- 代码逻辑在 app.js 中

IOS

JreapTabViewController.m

4.2.1.4. 消息

- 消息模块是网页（单页面应用），对应的文件目录是 www/web/notice

- 消息模块包括消息列表，消息发布，消息查看，消息删除
- 页面布局放在 views 目录
- 代码逻辑在 app.js 中

4.2.1.5. 工作

Android

FragmentAppManage.java，对应的布局文件是 fragment_app_manage.xml

IOS

AppManageController.m

说明：

工作主要是微应用的管理

微应用添加列表是网页，对应的文件目录是 www/web/appManage

4.2.1.6. 通讯录

Android

通讯录：OrganizationViewHolderFragment.java，对应的布局文件是

layout_organization.xml

账户详情：AccountDetailActivity.java，对应的布局文件是

activity_contact_detail.xml

选择人员：SelectUserActivity.java，对应的布局文件是 layout_organization.xml

说明：

通讯录 Fragment+ViewPager 实现的

IOS

通讯录：ContactsViewController.m

账户详情：ContactsDetailController.m

选择人员：JSelectUserController.m

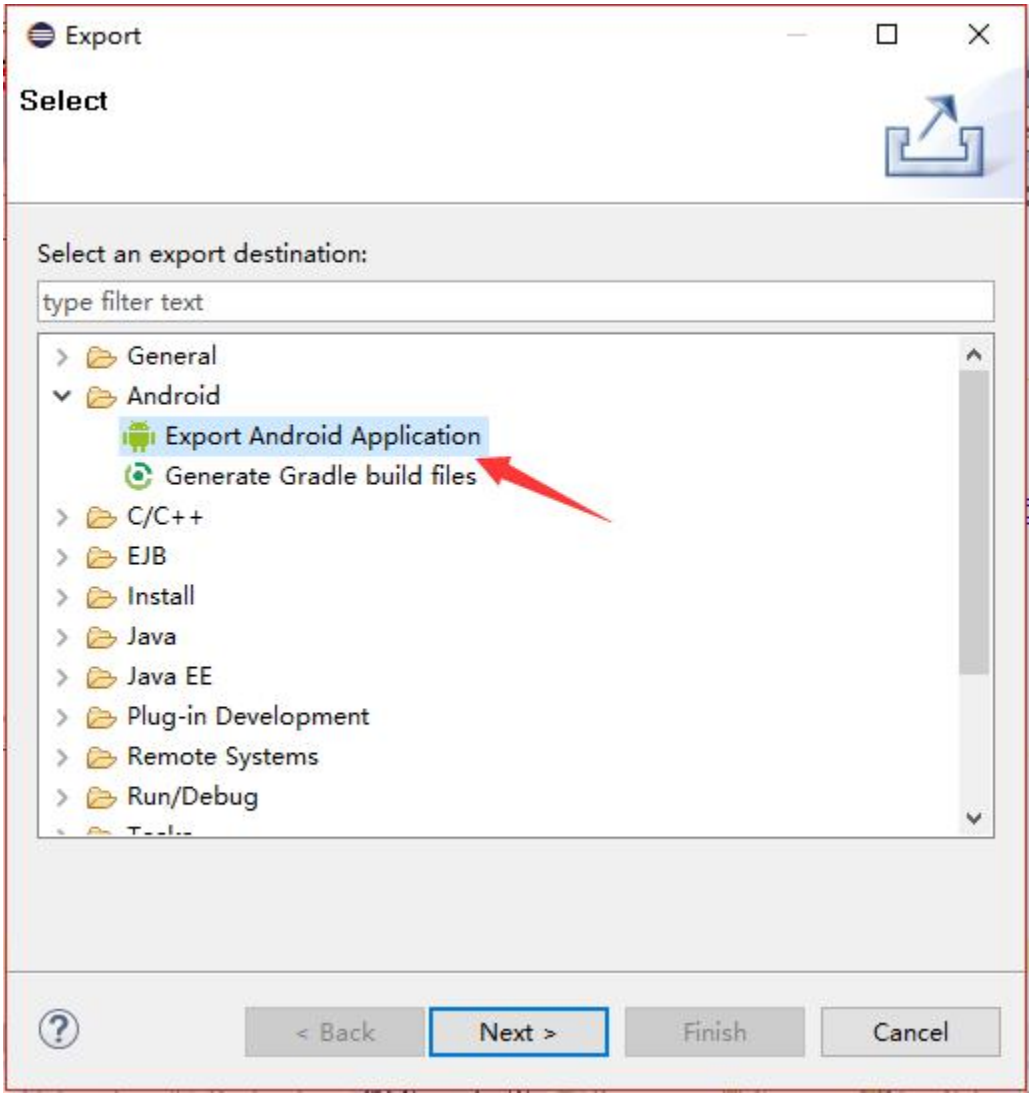
4.2.1.7. 个人中心

- 个人中心是网页（单页面应用），对应的文件目录是 www/web/setting
- 个人中心包含我的账号，意见与反馈，关于，退出
- 页面布局放在 views 目录
- 代码逻辑在 app.js 中

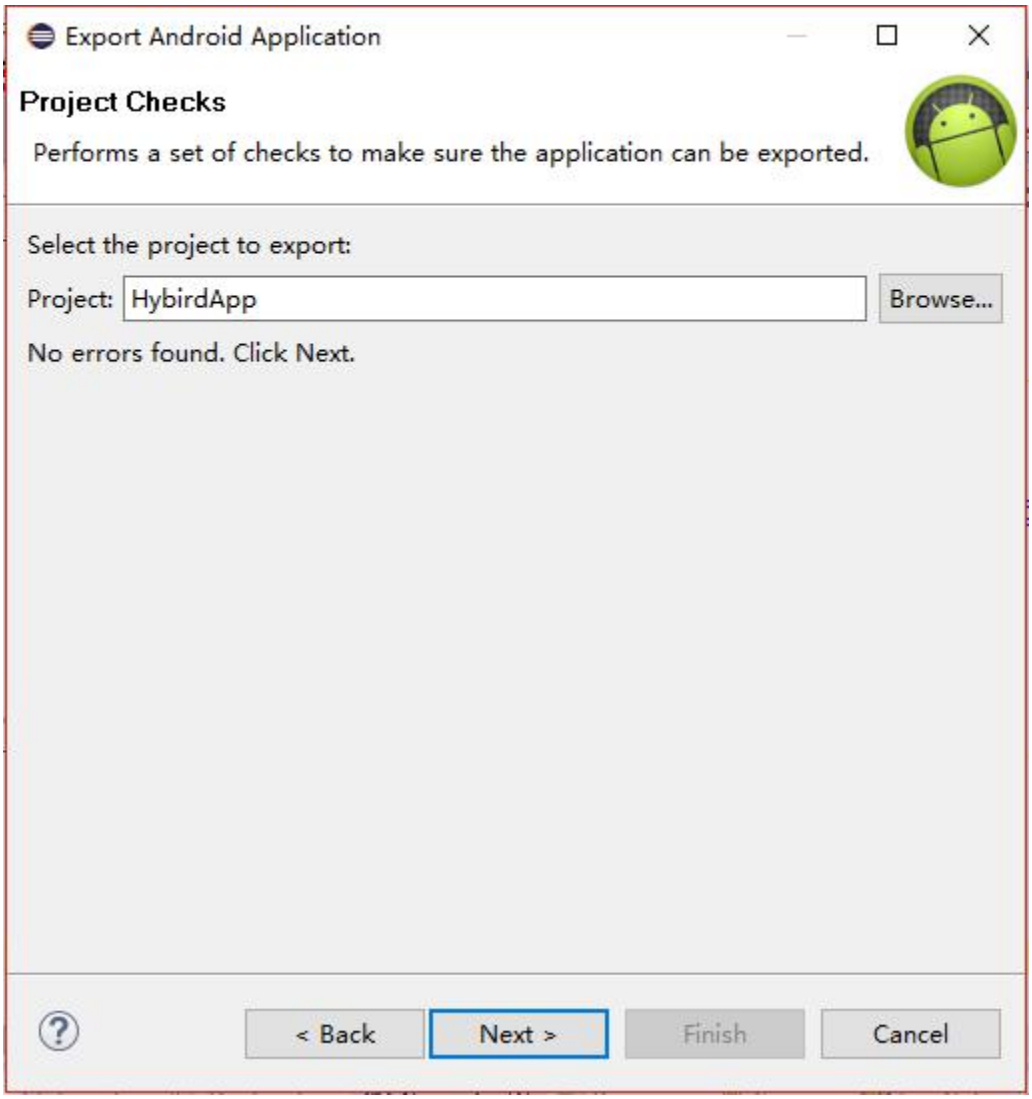
4.2.2. 移动端发布

4.2.2.1. Android 发布

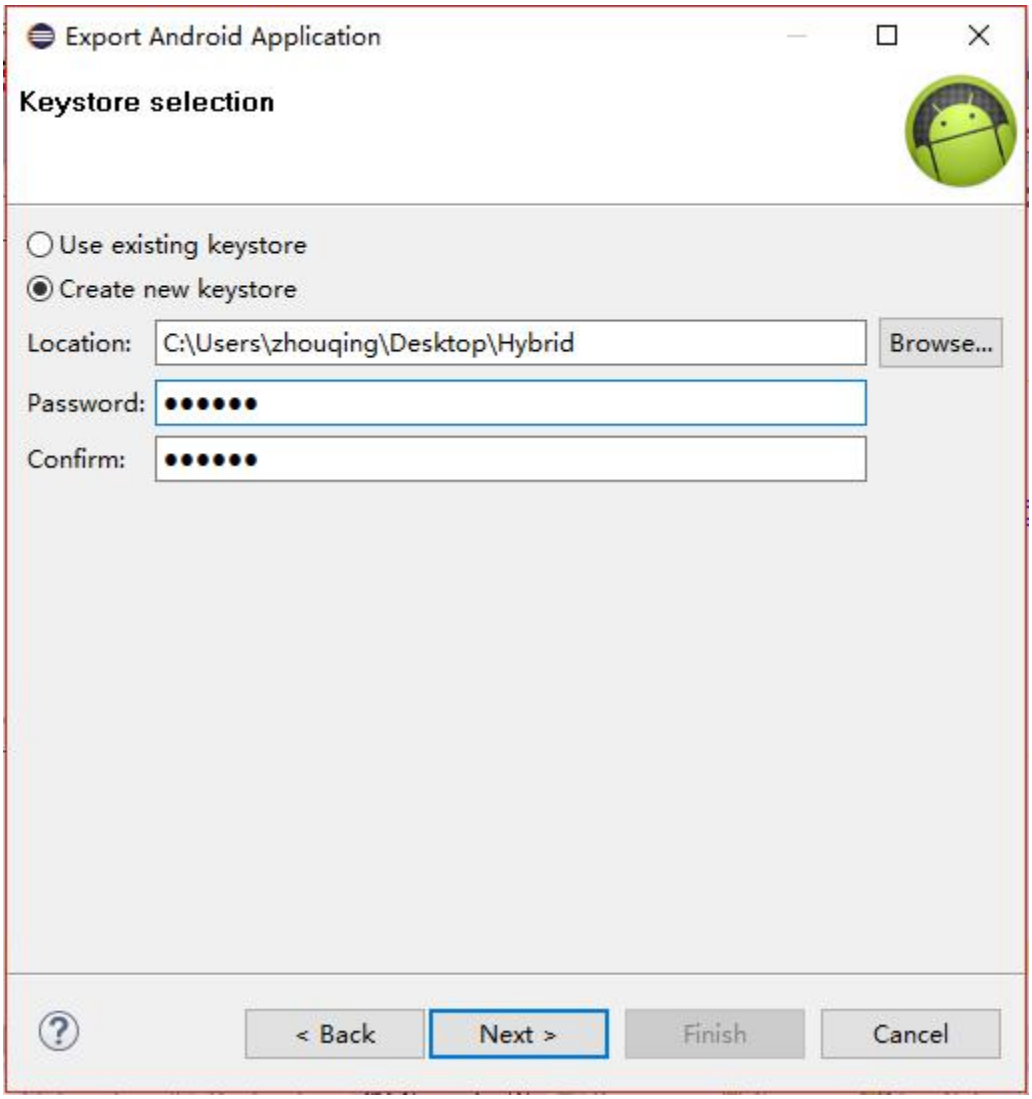
1) 在 Eclipse 中选择需要打包的项目，然后右键--选择 Export，会弹出一个打包的提示框，如下图所示：



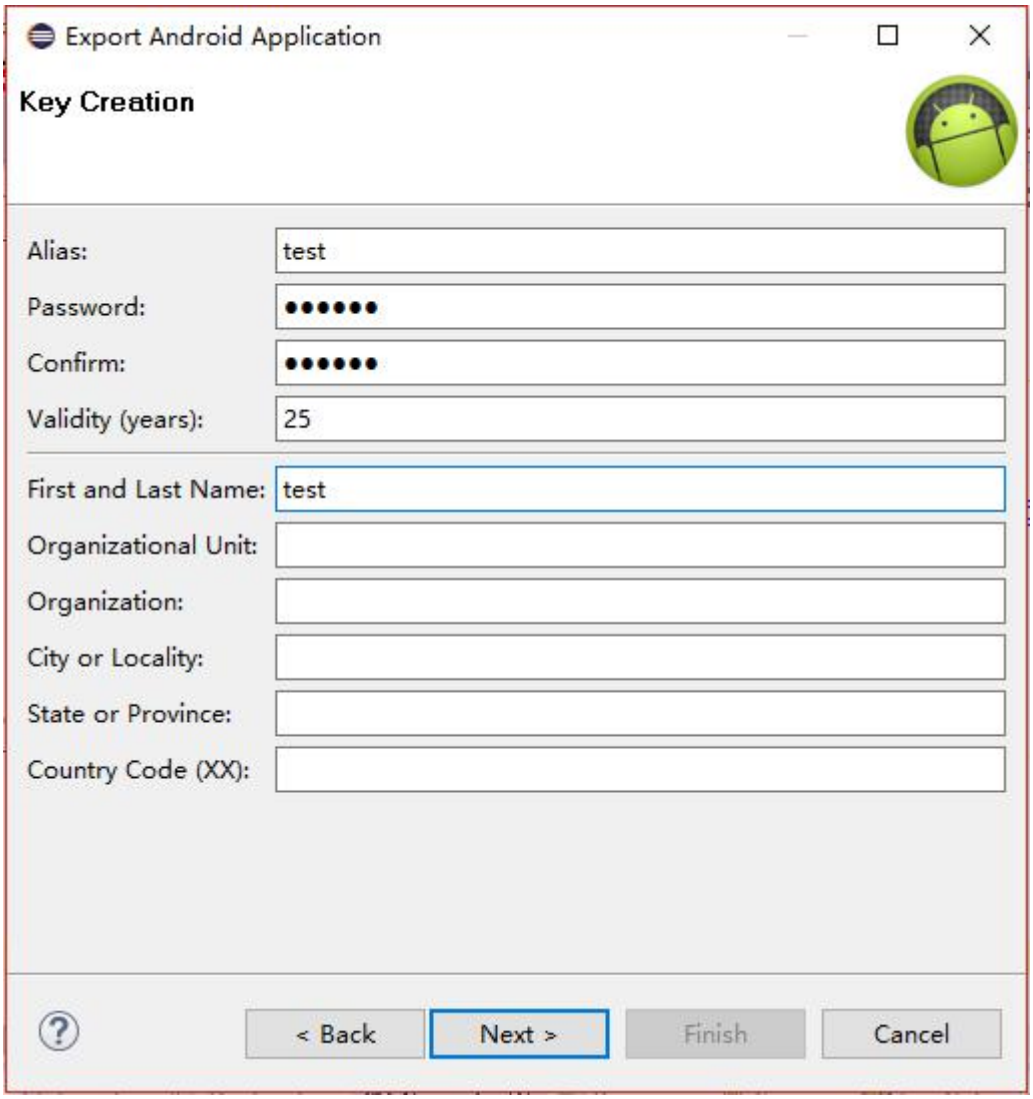
2) 按 Next 之后，会继续出现一个提示框，这里你可以选择自己需要打包的项目（默认是刚才选中的）如下图：



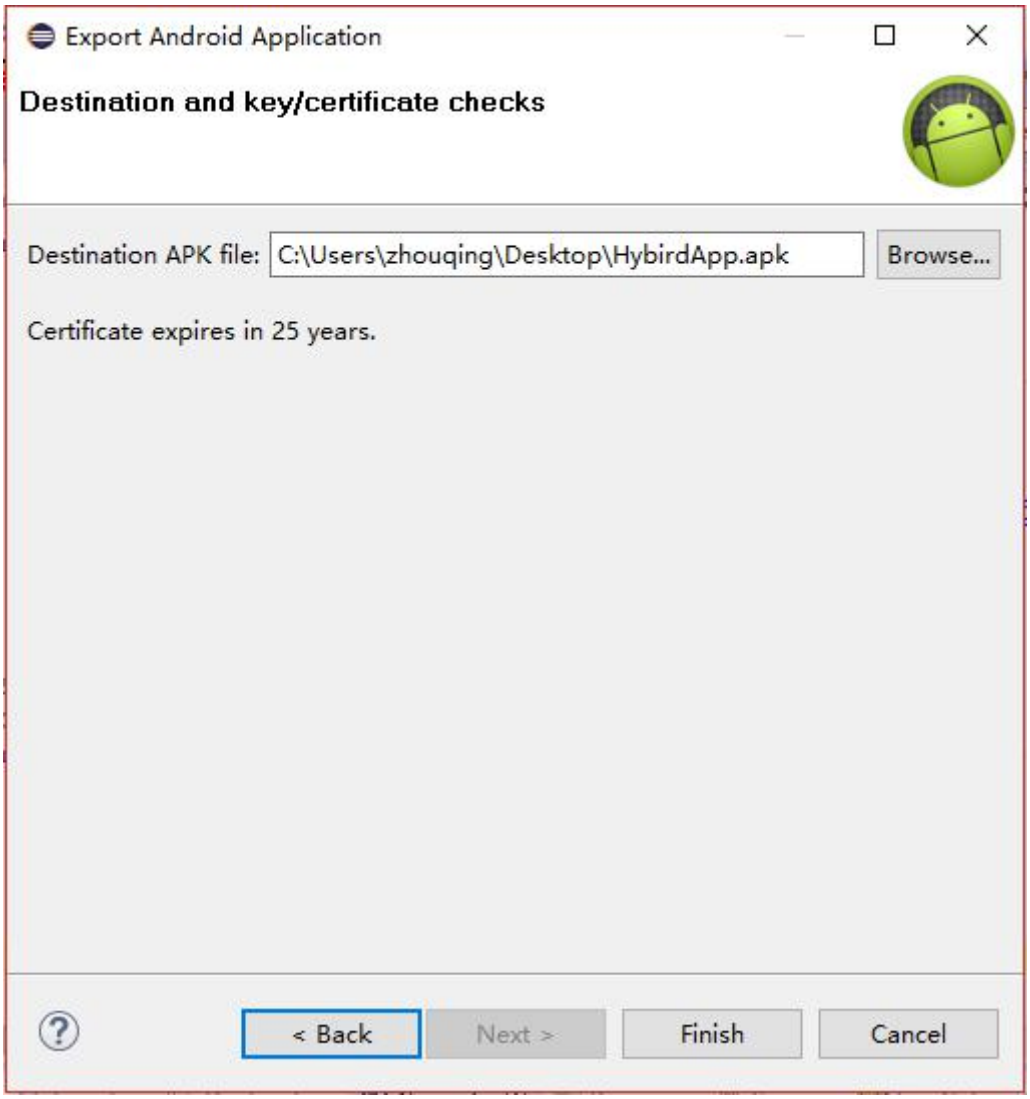
3) 按 Next 之后，会弹出一个关于“Keystore”的提示，选择“Create new Keystore”，并浏览、选择签名文件要保存的路径，如下图所示：



4) 按 Next 后，出现如下图所示的提示框：按照自己的实际情况和需求，填写相关信息。

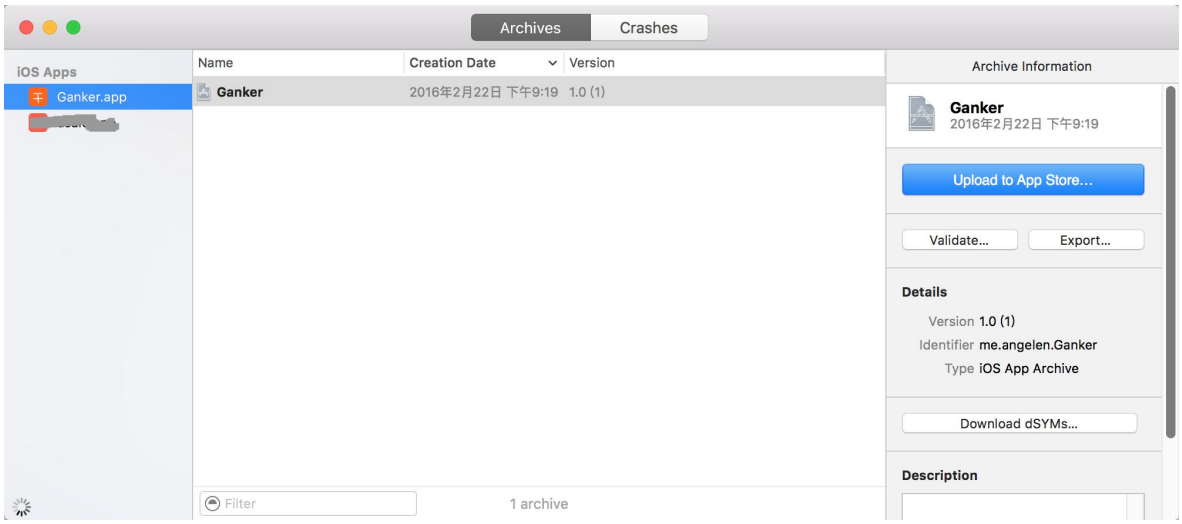


5) 按 Next 后，出现如下图所示的提示框：选择导出 APK 文件的路径，点击 Finish 完成。

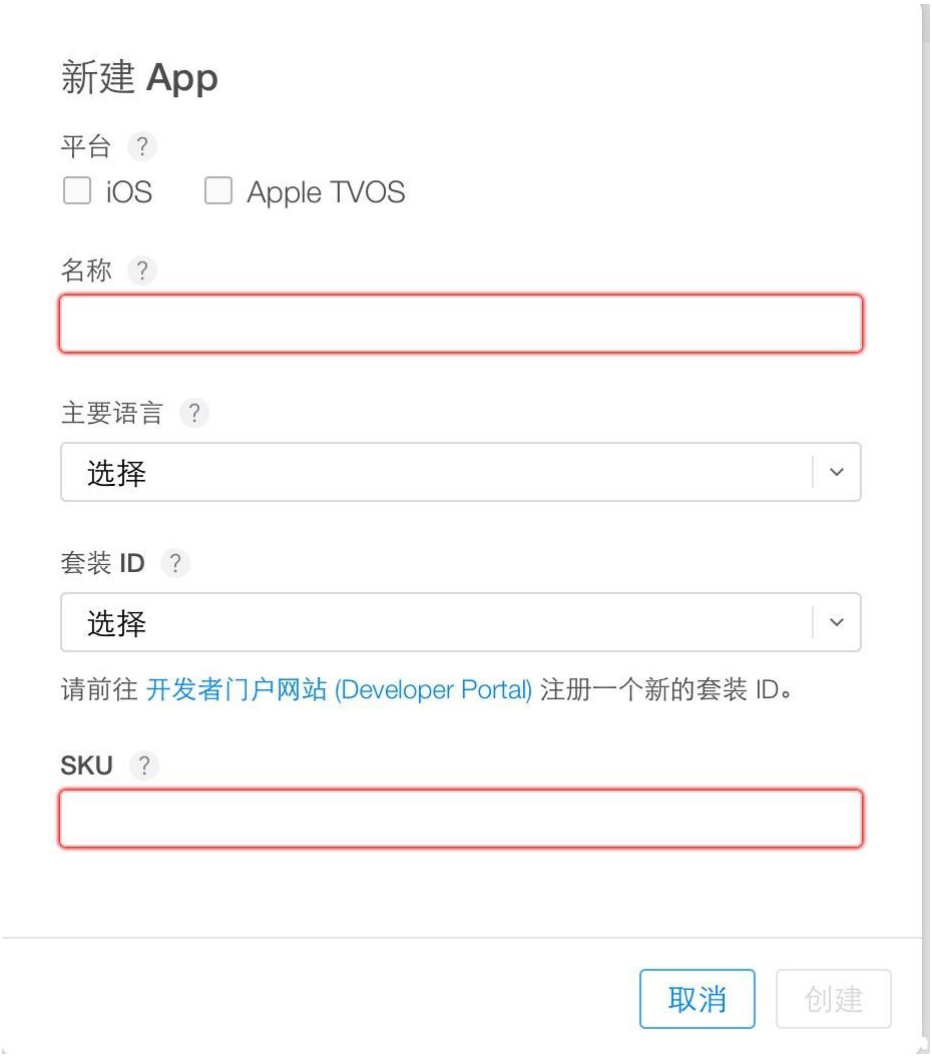


4.2.2.2. IOS 发布

1) 在 Xcode 中，点击 Product - Archive



2) 用 iTunes Connect 去注册一个该 app 的信息，点击“我的 App”，新建一个 App



3) 按提示操作一步步填写完相关信息后，在 Xcode 中上传打包的 app，提交审核，待审核通过后发布完成。

5. 平台开发资源

5.1. 工具类

5.1.1. Android

5.1.1.1. XUtils 框架（XUtils）

xUtils 简介

- xUtils 包含了很多实用的 android 工具。
- xUtils 最初源于 Afinal 框架，进行了大量重构，使得 xUtils 支持大文件上传，更全面的 http 请求协议支持(10种谓词)，拥有更加灵活的 ORM，更多的事件注解支持且不受混淆影响...
- xUitls 最低兼容 android 2.2 (api level 8)

目前 xUtils 主要有四大模块：

DbUtils 模块：

- android 中的 orm 框架，一行代码就可以进行增删改查；
- 支持事务，默认关闭；
- 可通过注解自定义表名，列名，外键，唯一性约束，NOT NULL 约束，CHECK 约束等（需要混淆的时候请注解表名和列名）；
- 支持绑定外键，保存实体时外键关联实体自动保存或更新；
- 自动加载外键关联实体，支持延时加载；
- 支持链式表达查询，更直观的查询语义，参考下面的介绍或 sample 中的例子。

ViewUtils 模块：

- android 中的 ioc 框架，完全注解方式就可以进行 UI，资源和事件绑定；
- 新的事件绑定方式，使用混淆工具混淆后仍可正常工作；
- 目前支持常用的20种事件绑定，参见 ViewCommonEventListener 类和包 com.lidroid.xutils.view.annotation.event。

HttpUtils 模块：

- 支持同步，异步方式的请求；
- 支持大文件上传，上传大文件不会 oom；
- 支持 GET，POST，PUT，MOVE，COPY，DELETE，HEAD，OPTIONS，TRACE，CONNECT 请求；
- 下载支持301/302重定向，支持设置是否根据 Content-Disposition 重命名下载的文件；
- 返回文本内容的请求(默认只启用了 GET 请求)支持缓存，可设置默认过期时间和针对当前请求的过期时间。

BitmapUtils 模块：

- 加载 bitmap 的时候无需考虑 bitmap 加载过程中出现的 oom 和 android 容器快速滑动时候出现的图片错位等现象；
- 支持加载网络图片和本地图片；
- 内存管理使用 lru 算法，更好的管理 bitmap 内存；

- 可配置线程加载线程数量，缓存大小，缓存路径，加载显示动画等...

使用 xUtils 快速开发框架需要有以下权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

混淆时注意事项：

- 添加 Android 默认混淆配置${sdk.dir}/tools/proguard/proguard-android.txt
- 不要混淆 xUtils 中的注解类型，添加混淆配置：-keep class * extends java.lang.annotation.Annotation { *; }
- 对使用 DbUtils 模块持久化的实体类不要混淆，或者注解所有表和列名称 @Table(name="xxx")，@Id(column="xxx")，@Column(column="xxx"),@Foreign(column="xxx",foreign="xxx")；

DbUtils 使用方法：

```
DbUtils db = DbUtils.create(this);

User user = new User(); //这里需要注意的是User对象必须有id属性，或者有通过@ID注解的属性

user.setEmail("wyouflf@qq.com");

user.setName("wyouflf");

db.save(user); // 使用saveBindingId保存实体时会为实体的id赋值



...

// 查找

Parent entity = db.findById(Parent.class, parent.getId());

List<Parent> list = db.findAll(Parent.class);//通过类型查找



Parent Parent =
db.findFirst(Selector.from(Parent.class).where("name","=","test"));
```

```
// IS NULL

Parent Parent = db.findFirst(Selector.from(Parent.class).where("name","=",
null));

// IS NOT NULL

Parent Parent = db.findFirst(Selector.from(Parent.class).where("name","!=",
null));



// WHERE id<54 AND (age>20 OR age<30) ORDER BY id LIMIT pageSize OFFSET pageOffset

List<Parent> list = db.findAll(Selector.from(Parent.class)

                                .where("id" ,"<", 54)

                                .and(WhereBuilder.b("age", ">", 20).or("age", " <
", 30))

                                .orderBy("id")

                                .limit(pageSize)

                                .offset(pageSize * pageIndex));



// op 为"in"时，最后一个参数必须是数组或 Iterable 的实现类(例如 List 等)

Parent test = db.findFirst(Selector.from(Parent.class).where("id", "in", new
int[]{1, 2, 3}));

// op 为"between"时，最后一个参数必须是数组或 Iterable 的实现类(例如 List 等)

Parent test = db.findFirst(Selector.from(Parent.class).where("id", "between", new
String[]{"1", "5"}));



DbModel dbModel =
db.findDbModelAll(Selector.from(Parent.class).select("name"));//select("name")
只取出 name 列

List<DbModel> dbModels =
db.findDbModelAll(Selector.from(Parent.class).groupBy("name").select("name",
"count(name)"));
```

```
...


List<DbModel> dbModels = db.findDbModelAll(sql); // 自定义 sql 查询

db.execNonQuery(sql) // 执行自定义 sql

...
```

ViewUtils 使用方法：

- 完全注解方式就可以进行 UI 绑定和事件绑定。
- 无需 findViewById 和 setClickListener 等。

```
// xUtils 的 view 注解要求必须提供 id，以使代码混淆不受影响。

@ViewInject(R.id.textView)

TextView textView;



//@ViewInject(vale=R.id.textView, parentId=R.id.parentView)

//TextView textView;



@ResInject(id = R.string.label, type = ResType.String)

private String label;



// 取消了之前使用方法名绑定事件的方式，使用 id 绑定不受混淆影响

// 支持绑定多个 id @OnClick({R.id.id1, R.id.id2, R.id.id3})

// or @OnClick(value={R.id.id1, R.id.id2, R.id.id3}, parentId={R.id.pid1,
R.id.pid2, R.id.pid3})

// 更多事件支持参见 ViewCommonEventListener 类和包
com.lidroid.xutils.view.annotation.event。

@OnClick(R.id.test_button)
```

```
public void testButtonClick(View v) { // 方法签名必须和接口中的要求一致

    ...

}

...

//在 Activity 中注入：

@Override

public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    ViewUtils.inject(this); //注入 view 和事件

    ...

    textView.setText("some text...");

    ...

}

//在 Fragment 中注入：

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {

    View view = inflater.inflate(R.layout.bitmap_fragment, container, false); //
加载 fragment 布局

    ViewUtils.inject(this, view); //注入 view 和事件

    ...

}

//在 PreferenceFragment 中注入：

public void onActivityCreated(Bundle savedInstanceState) {

    super.onActivityCreated(savedInstanceState);

    ViewUtils.inject(this, getPreferenceScreen()); //注入 view 和事件
```

```
    ...
}
// 其他重载
// inject(View view);
// inject(Activity activity)
// inject(PreferenceActivity preferenceActivity)
// inject(Object handler, View view)
// inject(Object handler, Activity activity)
// inject(Object handler, PreferenceGroup preferenceGroup)
// inject(Object handler, PreferenceActivity preferenceActivity)
```

HttpUtils 使用方法:

普通 get 方法

```
HttpUtils http = new HttpUtils();
http.send(HttpRequest.HttpMethod.GET,
    "http://www.lidroid.com",
    new RequestCallBack<String>(){
        @Override
        public void onLoading(long total, long current, boolean isUploading) {
            testTextView.setText(current + "/" + total);
        }

        @Override
        public void onSuccess(ResponseInfo<String> responseInfo) {
            textView.setText(responseInfo.result);
```

```
        }


        @Override
        public void onStart() {
        }


        @Override
        public void onFailure(HttpException error, String msg) {
        }
});
```

使用 HttpUtils 上传文件 或者 提交数据 到服务器（post 方法）

```
RequestParams params = new RequestParams();
params.addHeader("name", "value");
params.addQueryStringParameter("name", "value");


// 只包含字符串参数时默认使用BodyParamsEntity，
// 类似于UrlEncodedFormEntity（"application/x-www-form-urlencoded"）。
params.addBodyParameter("name", "value");


// 加入文件参数后默认使用MultipartEntity（"multipart/form-data"），
// 如需"multipart/related"，xUtils 中提供的MultipartEntity 支持设置subType 为"related"。
// 使用params.setBodyEntity(httpEntity)可设置更多类型的HttpEntity（如：
// MultipartEntity,BodyParamsEntity,FileUploadEntity,InputStreamUploadEntity,StringEntity）。
```

```
// 例如发送json参数: params.setBodyEntity(new StringEntity(jsonStr,charset));
params.addBodyParameter("file", new File("path"));
...

HttpUtils http = new HttpUtils();
http.send(HttpRequest.HttpMethod.POST,
    "uploadUrl....",
    params,
    new RequestCallBack<String>() {

        @Override
        public void onStart() {
            testTextView.setText("conn...");
        }

        @Override
        public void onLoading(long total, long current, boolean isUploading) {
            if (isUploading) {
                testTextView.setText("upload: " + current + "/" + total);
            } else {
                testTextView.setText("reply: " + current + "/" + total);
            }
        }

        @Override
        public void onSuccess(ResponseInfo<String> responseInfo) {
```

```
            testTextView.setText("reply: " + responseInfo.result);
        }

        @Override
        public void onFailure(HttpException error, String msg) {
            testTextView.setText(error.getExceptionCode() + ":" + msg);
        }
});
```

使用 HttpUtils 下载文件：

- 支持断点续传，随时停止下载任务，开始任务

```
HttpUtils http = new HttpUtils();
HttpHandler handler =
http.download("http://apache.dataguru.cn/httpcomponents/httpclient/source/httpcomponents-client-4.2.5-src.zip",
    "/sdcard/httpcomponents-client-4.2.5-src.zip",
    true, // 如果目标文件存在，接着未完成的部分继续下载。服务器不支持 RANGE 时将从新下载。
    true, // 如果从请求返回信息中获取到文件名，下载完成后自动重命名。
    new RequestCallBack<File>() {

        @Override
        public void onStart() {
            testTextView.setText("conn...");
        }

        @Override
```

```
        public void onLoading(long total, long current, boolean isUploading) {
            testTextView.setText(current + "/" + total);
        }

        @Override
        public void onSuccess(ResponseInfo<File> responseInfo) {
            testTextView.setText("downloaded:" + responseInfo.result.getPath());
        }


        @Override
        public void onFailure(HttpException error, String msg) {
            testTextView.setText(msg);
        }
});


...
//调用cancel()方法停止下载
handler.cancel();
```

BitmapUtils 使用方法：

```
BitmapUtils bitmapUtils = new BitmapUtils(this);

// 加载网络图片
bitmapUtils.display(testImageView,
"http://bbs.lidroid.com/static/image/common/logo.png");
```

```
// 加载本地图片(路径以/开头， 绝对路径)
bitmapUtils.display(testImageView, "/sdcard/test.jpg");

// 加载 assets 中的图片(路径以 assets 开头)
bitmapUtils.display(testImageView, "assets/img/wallpaper.jpg");

// 使用 ListView 等容器展示图片时可通过 PauseOnScrollListener 控制滑动和快速滑动过程中时候暂停加载图片
listView.setOnScrollListener(new PauseOnScrollListener(bitmapUtils, false, true));
listView.setOnScrollListener(new PauseOnScrollListener(bitmapUtils, false, true, customListener));
```

输出日志 LogUtils：

```
// 自动添加 TAG，格式： className.methodName(L:lineNumber)
// 可设置全局的 LogUtils.allowD = false，LogUtils.allowI = false...，控制是否输出 log。
// 自定义 log 输出 LogUtils.customLogger = new xxxLogger();
LogUtils.d("wyouflf");
```

5.1.1.2. 手机操作（PhoneUtil）

名称	方法	描述
拨打电话	Call(Context context, String number)	拨打电话 参数说明： context：Context 上下文 number：电话号码
发送短信	JumpToSMS(C	发送短信

	ontext context, String number, String body)	参数说明： context：Context 上下文 number：电话号码 body：发送短信内容

5.1.1.3. 双击判断（NoDoubleClickUtils）

名称	方法	描述
判断是否连续点击	isDoubleClick()	判断是否连续点击

5.1.1.4. 本地存储（SharedPreferencesUtils）

名称	方法	描述
保存本地数据	setParam(Context context, String key, Object object)	保存本地数据 参数说明： context：Context 上下文 key：key object：值 （类型：String,Integer,Boolean,Float,Long）
得到本地保存数据	getParam(Context context, String key, Object defaultObject)	得到本地保存数据 参数说明： context：Context 上下文 key：key defaultObject：默认值 （类型：String,Integer,Boolean,Float,Long）

5.1.1.5. 加解密处理（Encrypter）

名称	方法	描述
AES 解密	decryptByAes(String keys, String input)	AES 解密 参数说明： keys：密钥 input：输入字符串
AES 加密	encryptByAes(String keys, String input)	AES 加密 参数说明： keys：密钥 input：输入字符串
MD5 加密	encryptByMD5(String input)	MD5 加密 参数说明： input：输入字符串
SHA-1 加密	encryptBySHA1(String input)	SHA-1 加密 参数说明： input：输入字符串

5.1.1.6. 系统通用调用（SystemUtils）

名称	方法	描述
获取 ip	getLocalIpAddress()	获取 ip
获取 mac 地址	getMacAddress(Context context)	获取 mac 地址 参数说明： context：Context 上下文
是否模拟地点	isMockLocation(Context context)	是否模拟地点 参数说明： context：Context 上下文
获取定位信	getLocation	获取定位信息

息	(Context context)	参数说明： context：Context 上下文
获得 SD 卡总大小	getSDTotalSize()	获得 SD 卡总大小
获得 sd 卡剩余容量	getSDAvailableSize()	获得 sd 卡剩余容量，即可用大小
获得机身内存总大小	getRomTotalSize()	获得机身内存总大小
获得机身可用内存	getRomAvailableSize()	获得机身可用内存

5.1.1.7. json 转换工具（JsonConvertUtils）

名称	方法	描述
将 object 转化为 json 字符串	writeValueAsString(Object objValue)	将 object 转化为 json 字符串 参数说明： objValue：对象
将 json 转化为 object	readValue(String content, Type valueType)	将 json 转化为 object 参数说明： content：内容 valueType：类型

5.1.2. iOS

5.1.2.1. HttpRunner

名称	方法	描述
get 请求	doGet:(NSString *)url	get 方式请求

	parameter:(NSDictionary *)parameter success:(void(^)(id obj))success failuer:(void(^)(NSError *error))failure	参数说明： url：请求地址 parameter：请求参数 success：成功处理 failure：失败处理
post 请求	doPost:(NSString *)url parameter:(NSDictionary *)parameter success:(void(^)(id responseObject))success failure:(void(^)(NSError *error))failure	Post 方式请求 参数说明： url：请求地址 parameter：请求参数 success：成功处理 failure：失败处理
json 提交请求	doJsonPost:(NSString *)url parameter:(NSDictionary *)parameter success:(void(^)(id responseObject))success failure:(void(^)(NSError *error))failure	json 提交请求 参数说明： url：请求地址 parameter：请求参数 success：成功处理 failure：失败处理

5.1.2.2. MyUtil

名称	方法	描述
创建按钮	createBtnFrame:(CGRect)frame title:(NSString *)title bgImageName:(NSString *)bgImageName setImageName:(NSString	参数说明： frame：CGRect 矩形 title：标题 bgImageName：背景图片名称 imageName：图片名称

	*)imageName	color：按钮标题颜色
	btnTitleColo:(UIColor *)color	fontSize：字体大小
	btnTitleFont:(CGFloat)fontSize	target：目标
	target:(id)target action:(SEL)action	action：方法
创建一个	createViewFrame:(CGRect)frame	参数说明：
View	backgroudColor:(UIColor *)color	frame：CGRect 矩形
		color：背景颜色

5.1.2.3. SecurityUtil

名称	方法	描述
AES 加密	encryptAESData:(NSString*)string	将 string 转成带密码的 data 参数说明： string：加密字符串
AES 解密	decryptAESData:(NSString*)string	将 string 转成解密的 data 参数说明： string：密文

5.1.2.4. 字典转模型（MJExtension）

【示例】

The most simple JSON -> Model【最简单的字典转模型】

```
typedef enum {

    SexMale,

    SexFemale

} Sex;



@interface User : NSObject
```

```objc
@property (copy, nonatomic) NSString *name;

@property (copy, nonatomic) NSString *icon;

@property (assign, nonatomic) unsigned int age;

@property (copy, nonatomic) NSString *height;

@property (strong, nonatomic) NSNumber *money;

@property (assign, nonatomic) Sex sex;

@property (assign, nonatomic, getter=isGay) BOOL gay;

@end


/**********************************************/


NSDictionary *dict = @{

    @"name" : @"Jack",

    @"icon" : @"lufy.png",

    @"age" : @20,

    @"height" : @"1.55",

    @"money" : @100.9,

    @"sex" : @(SexFemale),

    @"gay" : @"true"

//   @"gay" : @"1"

//   @"gay" : @"NO"

};


// JSON -> User

User *user = [User mj_objectWithKeyValues:dict];
```

```
NSLog(@"name=%@, icon=%@, age=%zd, height=%@, money=%@, sex=%d, gay=%d", user.name, user.icon, user.age, user.height, user.money, user.sex, user.gay);

// name=Jack, icon=lufy.png, age=20, height=1.550000, money=100.9, sex=1
```

JSONString -> Model【JSON 字符串转模型】

```
// 1.Define a JSONString

NSString *jsonString = @"{\"name\":\"Jack\", \"icon\":\"lufy.png\", \"age\":20}";



// 2.JSONString -> User

User *user = [User mj_objectWithKeyValues:jsonString];



// 3.Print user's properties

NSLog(@"name=%@, icon=%@, age=%d", user.name, user.icon, user.age);

// name=Jack, icon=lufy.png, age=20
```

Model contains model【模型中嵌套模型】

```
@interface Status : NSObject

@property (copy, nonatomic) NSString *text;

@property (strong, nonatomic) User *user;

@property (strong, nonatomic) Status *retweetedStatus;

@end



/*********************************************/



NSDictionary *dict = @{
```

```
    @"text" : @"Agree!Nice weather!",
    @"user" : @{
        @"name" : @"Jack",
        @"icon" : @"lufy.png"
    },
    @"retweetedStatus" : @{
        @"text" : @"Nice weather!",
        @"user" : @{
            @"name" : @"Rose",
            @"icon" : @"nami.png"
        }
    }
};

// JSON -> Status
Status *status = [Status mj_objectWithKeyValues:dict];

NSString *text = status.text;
NSString *name = status.user.name;
NSString *icon = status.user.icon;
NSLog(@"text=%@, name=%@, icon=%@", text, name, icon);
// text=Agree!Nice weather!, name=Jack, icon=lufy.png

NSString *text2 = status.retweetedStatus.text;
NSString *name2 = status.retweetedStatus.user.name;
```

```
NSString *icon2 = status.retweetedStatus.user.icon;

NSLog(@"text2=%@, name2=%@, icon2=%@", text2, name2, icon2);

// text2=Nice weather!, name2=Rose, icon2=nami.png
```

Model contains model-array【模型中有个数组属性，数组里面又要装着其他模型】

```
@interface Ad : NSObject

@property (copy, nonatomic) NSString *image;

@property (copy, nonatomic) NSString *url;

@end


@interface StatusResult : NSObject

/** Contatins status model */

@property (strong, nonatomic) NSMutableArray *statuses;

/** Contatins ad model */

@property (strong, nonatomic) NSArray *ads;

@property (strong, nonatomic) NSNumber *totalNumber;

@end


/*********************************************/


// Tell MJExtension what type model will be contained in statuses and ads.

[StatusResult mj_setupObjectClassInArray:^NSDictionary *{

    return @{

             @"statuses" : @"Status",
```

```
            // @"statuses" : [Status class],
            @"ads" : @"Ad"
            // @"ads" : [Ad class]
        };
}];
// Equals: StatusResult.m implements +mj_objectClassInArray method.

NSDictionary *dict = @{
    @"statuses" : @[
                    @{
                        @"text" : @"Nice weather!",
                        @"user" : @{
                            @"name" : @"Rose",
                            @"icon" : @"nami.png"
                        }
                    },
                    @{
                        @"text" : @"Go camping tomorrow!",
                        @"user" : @{
                            @"name" : @"Jack",
                            @"icon" : @"lufy.png"
                        }
                    }
                ],
    @"ads" : @[
```

```
                @{
                    @"image" : @"ad01.png",
                    @"url" : @"http://www.ad01.com"
                },
                @{
                    @"image" : @"ad02.png",
                    @"url" : @"http://www.ad02.com"
                }
            ],
    @"totalNumber" : @"2014"
};

// JSON -> StatusResult
StatusResult *result = [StatusResult mj_objectWithKeyValues:dict];

NSLog(@"totalNumber=%@", result.totalNumber);
// totalNumber=2014

// Printing
for (Status *status in result.statuses) {
    NSString *text = status.text;
    NSString *name = status.user.name;
    NSString *icon = status.user.icon;
    NSLog(@"text=%@, name=%@, icon=%@", text, name, icon);
}
```

```
// text=Nice weather!, name=Rose, icon=nami.png

// text=Go camping tomorrow!, name=Jack, icon=lufy.png



// Printing

for (Ad *ad in result.ads) {

    NSLog(@"image=%@, url=%@", ad.image, ad.url);

}

// image=ad01.png, url=http://www.ad01.com

// image=ad02.png, url=http://www.ad02.com
```

Model name - JSON key mapping【模型中的属性名和字典中的 key 不相同(或者需要多级映射)】

```
@interface Bag : NSObject

@property (copy, nonatomic) NSString *name;

@property (assign, nonatomic) double price;

@end



@interface Student : NSObject

@property (copy, nonatomic) NSString *ID;

@property (copy, nonatomic) NSString *desc;

@property (copy, nonatomic) NSString *nowName;

@property (copy, nonatomic) NSString *oldName;

@property (copy, nonatomic) NSString *nameChangedTime;

@property (strong, nonatomic) Bag *bag;

@end
```

```
/**********************************************/

// How to map
[Student mj_setupReplacedKeyFromPropertyName:^NSDictionary *{
    return @{
              @"ID" : @"id",
              @"desc" : @"desciption",
              @"oldName" : @"name.oldName",
              @"nowName" : @"name.newName",
              @"nameChangedTime" : @"name.info[1].nameChangedTime",
              @"bag" : @"other.bag"
          };
}];
// Equals: Student.m implements +mj_replacedKeyFromPropertyName method.

NSDictionary *dict = @{
    @"id" : @"20",
    @"desciption" : @"kids",
    @"name" : @{
        @"newName" : @"lufy",
        @"oldName" : @"kitty",
        @"info" : @[
                  @"test-data",
                  @{
```

```
                        @"nameChangedTime" : @"2013-08"
                    }
                ]
    },
    @"other" : @{
        @"bag" : @{
            @"name" : @"a red bag",
            @"price" : @100.7
        }
    }
};

// JSON -> Student
Student *stu = [Student mj_objectWithKeyValues:dict];

// Printing
NSLog(@"ID=%@, desc=%@, oldName=%@, nowName=%@, nameChangedTime=%@",
      stu.ID, stu.desc, stu.oldName, stu.nowName, stu.nameChangedTime);
// ID=20, desc=kids, oldName=kitty, nowName=lufy, nameChangedTime=2013-08
NSLog(@"bagName=%@, bagPrice=%f", stu.bag.name, stu.bag.price);
// bagName=a red bag, bagPrice=100.700000
```

JSON array -> model array【将一个字典数组转成模型数组】

```
NSArray *dictArray = @[
                    @{
                        @"name" : @"Jack",
```

```
                            @"icon" : @"lufy.png"
                        },
                        @{
                            @"name" : @"Rose",
                            @"icon" : @"nami.png"
                        }
                    ];

// JSON array -> User array
NSArray *userArray = [User mj_objectArrayWithKeyValuesArray:dictArray];

// Printing
for (User *user in userArray) {
    NSLog(@"name=%@, icon=%@", user.name, user.icon);
}
// name=Jack, icon=lufy.png
// name=Rose, icon=nami.png
```

Model -> JSON【将一个模型转成字典】

```
// New model
User *user = [[User alloc] init];
user.name = @"Jack";
user.icon = @"lufy.png";

Status *status = [[Status alloc] init];
status.user = user;
```

```
status.text = @"Nice mood!";


// Status -> JSON
NSDictionary *statusDict = status.mj_keyValues;
NSLog(@"%@", statusDict);
/*
 {
 text = "Nice mood!";
 user =     {
 icon = "lufy.png";
 name = Jack;
 };
 }
 */


// More complex situation
Student *stu = [[Student alloc] init];
stu.ID = @"123";
stu.oldName = @"rose";
stu.nowName = @"jack";
stu.desc = @"handsome";
stu.nameChangedTime = @"2018-09-08";


Bag *bag = [[Bag alloc] init];
bag.name = @"a red bag";
```

```
bag.price = 205;

stu.bag = bag;



NSDictionary *stuDict = stu.mj_keyValues;

NSLog(@"%@", stuDict);

/*

{

    ID = 123;

    bag =     {

        name = "\U5c0f\U4e66\U5305";

        price = 205;

    };

    desc = handsome;

    nameChangedTime = "2018-09-08";

    nowName = jack;

    oldName = rose;

}

 */
```

Model array -> JSON array【将一个模型数组转成字典数组】

```
// New model array

User *user1 = [[User alloc] init];

user1.name = @"Jack";

user1.icon = @"lufy.png";



User *user2 = [[User alloc] init];
```

```
user2.name = @"Rose";

user2.icon = @"nami.png";



NSArray *userArray = @[user1, user2];



// Model array -> JSON array

NSArray *dictArray = [User mj_keyValuesArrayWithObjectArray:userArray];

NSLog(@"%@", dictArray);

/*

 (

 {

 icon = "lufy.png";

 name = Jack;

 },

 {

 icon = "nami.png";

 name = Rose;

 }

 )

 */
```

Core Data

```
NSDictionary *dict = @{

                        @"name" : @"Jack",

                        @"icon" : @"lufy.png",

                        @"age" : @20,
```

```
                        @"height" : @1.55,
                        @"money" : @"100.9",
                        @"sex" : @(SexFemale),
                        @"gay" : @"true"
                    };

// This demo just provide simple steps
NSManagedObjectContext *context = nil;
User *user = [User mj_objectWithKeyValues:dict context:context];

[context save:nil];
```

Coding

```
#import "MJExtension.h"

@implementation Bag
// NSCoding Implementation
MJExtensionCodingImplementation
@end

/*********************************************/

// what properties not to be coded
[Bag mj_setupIgnoredCodingPropertyNames:^NSArray *{
    return @[@"name"];
}];
```

```
// Equals: Bag.m implements +mj_ignoredCodingPropertyNames method.


// Create model
Bag *bag = [[Bag alloc] init];
bag.name = @"Red bag";
bag.price = 200.8;


NSString *file = [NSHomeDirectory() stringByAppendingPathComponent:@"Desktop/bag.data"];
// Encoding
[NSKeyedArchiver archiveRootObject:bag toFile:file];


// Decoding
Bag *decodedBag = [NSKeyedUnarchiver unarchiveObjectWithFile:file];
NSLog(@"name=%@, price=%f", decodedBag.name, decodedBag.price);
// name=(null), price=200.800000
```

Camel -> underline【统一转换属性名（比如驼峰转下划线）】

```
// Dog
#import "MJExtension.h"


@implementation Dog
+ (NSString *)mj_replacedKeyFromPropertyName121:(NSString *)propertyName
{
    // nickName -> nick_name
    return [propertyName mj_underlineFromCamel];
```

```
}

@end


// NSDictionary

NSDictionary *dict = @{

                     @"nick_name" : @"旺财",

                     @"sale_price" : @"10.5",

                     @"run_speed" : @"100.9"

                     };

// NSDictionary -> Dog

Dog *dog = [Dog mj_objectWithKeyValues:dict];


// printing

NSLog(@"nickName=%@, scalePrice=%f runSpeed=%f", dog.nickName, dog.salePrice,
dog.runSpeed);
```

NSString -> NSDate, nil -> @""【过滤字典的值（比如字符串日期处理为NSDate、字符串 nil 处理为@""）】

```
// Book

#import "MJExtension.h"


@implementation Book

- (id)mj_newValueFromOldValue:(id)oldValue property:(MJProperty *)property

{

    if ([property.name isEqualToString:@"publisher"]) {

        if (oldValue == nil) return @"";
```

```
    } else if (property.type.typeClass == [NSDate class]) {
        NSDateFormatter *fmt = [[NSDateFormatter alloc] init];
        fmt.dateFormat = @"yyyy-MM-dd";
        return [fmt dateFromString:oldValue];
    }

    return oldValue;
}
@end

// NSDictionary
NSDictionary *dict = @{
                      @"name" : @"5分钟突破 iOS 开发",
                      @"publishedTime" : @"2011-09-10"
                      };
// NSDictionary -> Book
Book *book = [Book mj_objectWithKeyValues:dict];

// printing
NSLog(@"name=%@, publisher=%@, publishedTime=%@", book.name, book.publisher,
book.publishedTime);
```

5.1.2.5. 提示框（ProgressHUD）

【使用】

1.在头部引入下面的文件

```
#import "ProgressHUD.h"
```

2.用下面的方式显示提示信息

```
[ProgressHUD show:@"Please wait..."];
```

3.调用完成关闭提示

```
[ProgressHUD dismiss];
```

5.2. Cordova 本地扩展调用接口

基于 angular 方式调用 Cordova 接口

5.2.1. 系统操作（system）

名称	函数	描述
toast 短提示	showShort(message, successCallback, errorCallback)	toast 短时间提示 调用示例： $cordovaJReapExtend.system.showShort("提示信息", null, null); 参数说明： message：提示内容 successCallback：成功回调函数 errorCallback：失败回调函数
toast 长提示	showLong(message, successCallback, errorCallback)	toast 长时间提示 调用示例： $cordovaJReapExtend.system.showLong("提示信息", null, null); 参数说明： message：提示内容 successCallback：成功回调函数 errorCallback：失败回调函数

取消 toast 提示	cancel(successCallback, errorCallback)	取消 toast 提示 调用示例： `$cordovaJReapExtend.system.cancel(null, null);` 参数说明： successCallback：成功回调函数 errorCallback：失败回调函数
退出登录	logout()	退出登录 调用示例： `$cordovaJReapExtend.system.logout();`
退出 APP	exitApp()	退出 APP 调用示例： `$cordovaJReapExtend.system.exitApp();`
退出到登录页	logoutToLogin()	退出到登录页 调用示例： `$cordovaJReapExtend.system.logoutToLogin();`
获得版本名称	getVersionName(successCallback)	获得版本名称 调用示例： `$cordovaJReapExtend.system.getVersionName(null);` 参数说明： successCallback：成功回调函数
获取 ComponetKey	getAppComponetKey(successCallback)	获取用户微应用中心已添加应用 ComponetKey 调用示例： `$cordovaJReapExtend.system.getAppComponetKey(function(componetKey) { alert(componetKey); });` 参数说明： successCallback：成功回调函数
二维码扫描	openScanCode(successCa	二维码扫描 调用示例：

	llback)	$cordovaJReapExtend.system.openScanCode(function (data) {
		alert(JSON.stringify(data));
		});
		参数说明：
		successCallback：成功回调函数

图片添加文字水印	getWatermark(successCallback,filePath,words)	图片添加文字水印
		调用示例：
		$cordovaJReapExtend.system.getWatermark(function(success){
		alert(success);
		},$scope.filePath,"这是水印文字");
		参数说明：
		successCallback：成功回调函数
		filePath：图片路径
		words：水印文字

拨打电话	call(tel, successCallback, errorCallback)	拨打电话 调用示例： $cordovaJReapExtend.system.call("15866668888", null, null); 参数说明： tel：电话号码 successCallback：成功回调函数 errorCallback：失败回调函数

选择人员/部门	openSelectUser(successCallback,params)	选择人员/部门
		调用示例：
		$cordovaJReapExtend.system.openSelectUser(function(data){
		$scope.$apply(function (){
		$scope.accountitem.selectedAccountIds = data.selectedIds;
		$scope.accountitem.selectedAccountNames = data.selectedNames;

		}); },{isMultiSelect:**true**, selectMode:**false**, needReSelect:**true**, reSelectedIds:$scope.accountitem.selectedAccountIds}); 参数说明： successCallback：成功回调函数 params：参数（{ isMultiSelect:**true**,//是否多选 true：多选 false：单选 selectMode:**true**,// 选 择模式 true：选择部门 false：选择员工 needReSelect:**false**,//是否需要回选 true：需要 false：不需要 reSelectedIds:""// 回 选用户ids }）
打开网页	openMicroAppView(successCallback, url)	打开网页 调用示例： $cordovaJReapExtend.system.openMicroAppView(**null**, **"http://www.baidu.com"**); 参数说明： successCallback：成功回调函数 url：url 地址

APP 更新	appUpdate(successCallback, errorCallback,fileUrl, params)	APP 更新
		调用示例：
		$cordovaJReapExtend.system.appUpdate(**function**(res) {
		},
		function(error) {
		$cordovaJReapExtend.system.showShort(JSON.stringify(error), **null**, **null**);
		},$scope.upgradeUrl, {overwrite: **true**});
		参数说明：
		successCallback：成功回调函数

errorCallback：失败回调函数

fileUrl：下载路径

Params：参数

5.2.2. 定位（Location）

名称	函数	描述
是否模拟地点	isMockLocation(successCallback, errorCallback)	是否模拟地点 调用示例： $cordovaJReapExtend.location.isMockLocation(function(res) {}, function(err) {}); 参数说明： successCallback：成功回调函数 errorCallback：失败回调函数
定位	getCurrentPosition(successCallback, errorCallback)	定位 调用示例： $cordovaJReapExtend.location.getCurrentPosition(function(res) {}, function(err) {}); 参数说明： successCallback：成功回调函数（返回定位数据） errorCallback：失败回调函数
停止定位	stop(successCallback, errorCallback)	停止定位 调用示例： $cordovaJReapExtend.location.stop(function(res) {}, function(err) {}); 参数说明： successCallback：成功回调函数 errorCallback：失败回调函数
地图展示	openBaiduMap(method,lat,lng,address)	原生地图展示 调用示例： $cordovaJReapExtend.location.openBaiduMap(false,111.11,222.22,"湖南省长沙市天心区"); 参数说明：

method：单条 / 多条（false/true）

lat：纬度

lng：经度

address：地址

5.2.3. 本地数据库操作（dbOperate）

注：该功能 iOS 暂不支持

名称	函数	描述

创建表	createTable(successCallback,createTableSql)	创建表
		调用示例：
		$scope.createTableSql = "CREATE TABLE IF NOT EXISTS test (testID TEXT NOT NULL, name TEXT NOT NULL, adress TEXT)";
		$cordovaJReapExtend.dbOperate.createTable(function(data){
		if(data == "success"){
		$cordovaJReapExtend.system.showShort("创建成功", null, null);
		}else{
		$cordovaJReapExtend.system.showShort("创建失败", null, null);
		}
		},$scope.createTableSql);
		参数说明：
		successCallback：成功回调函数
		createTableSql：建表语句

删除表	dropTable(successCallback,tableName)	删除表
		调用示例：
		$cordovaJReapExtend.dbOperate.dropTable(function(data){
		if(data == "success"){

```
    $cordovaJReapExtend.system.showShort("删除成功", null, null);
                }else{

    $cordovaJReapExtend.system.showShort("删除失败", null, null);
                }
            },tableName);
```

参数说明：

successCallback：成功回调函数

tableName：表名

执行 sql	execNonQuery(successCallback,sql)	执行 sql(新增，更新，删除)
		调用示例：
		`$scope.sql = "insert into test (testID, name, adress) values('4','ddd','asdds')";`
		`$cordovaJReapExtend.dbOperate.execNonQuery(function(data){`
		`if(data == "success"){`
		`$cordovaJReapExtend.system.showShort("操作成功", null, null);`
		`}else{`
		`$cordovaJReapExtend.system.showShort("操作失败", null, null);`
		`}`
		`},$scope.sql);`
		参数说明：
		successCallback：成功回调函数
		sql：sql 语句

执行查询 sql	execQuery(successCallback,querySql)	执行查询 sql
		调用示例：
		`$scope.querySql = "select * from test";`
		`$cordovaJReapExtend.dbOperate.execQuery(function(data){`

```
                if(data == "fail"){

    $cordovaJReapExtend.system.showShort("查询失败", null, null);
                }else{
                    alert(data);
                }
            },$scope.querySql);
```

参数说明：

successCallback：成功回调函数

querySql：查询 sql 语句

5.3. Cordova 调用示例及说明

5.3.1. 文件操作

$cordovaFile

方法(Methods)

getFreeDiskSpace()

获取设备磁盘总空闲空间

返回值 String

checkDir(path, directory)

检测某路径的文件目录是否存在

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
directory	String	检查的目录名称

返回值 Object

checkFile(path, file)

检测某路径中的文件是否存在

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	检查的文件路径

返回值 Object

createDir(path, directory, replace)

在特定路径中创建一个新目录。replace boolean 参数确定是否以相同的名称替换已存在的目录。如果存在目录存在且替换值为 false，则承诺将失败并返回一个错误。

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
directory	String	创建目录名称
replace	Boolean	如果为 true 替换目录,如果为 false 有相同名称的文件时,返回一个错误

返回值 Object

createFile(path, file, replace)

在特定路径中创建一个新文件。replace boolean 参数确定是否以相同的名称替换已存在的文件。如果存在的文件存在且替换值是错误的，则该承诺将失败并返回一个错误。

参数	类型	说明

path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	创建文件名称
replace	Boolean	如果为 true 替换目录,如果为 false 有相同名称的文件时,返回一个错误

返回值 Object

removeFile(path, file)

从所需的位置删除一个文件。

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	删除文件名称

返回值 Object

removeDir(path, directory)

从所需的位置删除一个目录。

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	删除目录名称

返回值 Object

removeRecursively(path, directory)

从所需的位置删除所有文件和目录。

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
directory	String	删除目录名称

返回值 Object

writeFile(path, file, data, replace)

写入一个新文件，如果需要的话，可以替换。

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	写入文件名称
data	String	Text/data 写入文件的数据
replace	String	如果为 true 替换文件,如果为 false 有相同名称的文件时,返回一个错误

返回值 Object

writeExistingFile(path, file, data)

写入一个存在的文件。

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	写入文件名称

data	String	Text/data 写入文件的数据

返回值 Object

readAsText(path, file)

readAsDataURL(path, file)

readAsBinaryString(path, file)

readAsArrayBuffer(path, file)

在各种方法中读取文件。

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	读取文件名称

返回值 Object

moveDir(path, directory, newPath, newDirectory)

移动一个目录

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
directory	String	移动目录名称
newPath	String	新位置的基文件系统请参考以上的 iOS 和 Android 的文件系统

newDirectory	String	新目录名称

返回值 Object

moveFile(path, file, newPath, newFile)

移动一个文件

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	移动文件名称
newPath	String	新位置的基文件系统请参考以上的 iOS 和 Android 的文件系统
newFile	String	新文件名称

返回值 Object

copyDir(path, directory, newPath, newDirectory)

复制一个目录

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
directory	String	复制目录名称
newPath	String	新位置的基文件系统请参考以上的 iOS 和 Android 的文件系统
newDirectory	String	新目录名称

返回值 Object

copyFile(path, file, newPath, newFile)

复制一个文件

参数	类型	说明
path	FileSystem	基本文件系统。请参考以上的 iOS 和 Android 的文件系统
file	String	复制文件名称
newPath	String	新位置的基文件系统请参考以上的 iOS 和 Android 的文件系统
newFile	String	新文件名称

返回值 Object

代码实例

```
module.controller('MyCtrl', function($scope, $cordovaFile) {
  document.addEventListener('deviceready', function () {
    $cordovaFile.getFreeDiskSpace()
      .then(function (success) {
        // success in kilobytes
      }, function (error) {
        // error
      });
    // CHECK
    $cordovaFile.checkDir(cordova.file.dataDirectory, "dir/other_dir")
      .then(function (success) {
```

```
    // success
  }, function (error) {
    // error
  });
$cordovaFile.checkFile(cordova.file.dataDirectory, "some_file.txt")
  .then(function (success) {
    // success
  }, function (error) {
    // error
  });
// CREATE
$cordovaFile.createDir(cordova.file.dataDirectory, "new_dir", false)
  .then(function (success) {
    // success
  }, function (error) {
    // error
  });

$cordovaFile.createFile(cordova.file.dataDirectory, "new_file.txt", true)
  .then(function (success) {
    // success
  }, function (error) {
```

```
        // error
    });
// REMOVE
$cordovaFile.removeDir(cordova.file.dataDirectory, "some_dir")
    .then(function (success) {
        // success
    }, function (error) {
        // error
    });
$cordovaFile.removeFile(cordova.file.dataDirectory, "some_file.txt")
    .then(function (success) {
        // success
    }, function (error) {
        // error
    });
$cordovaFile.removeRecursively(cordova.file.dataDirectory, "")
    .then(function (success) {
        // success
    }, function (error) {
        // error
    });
// WRITE
```

```
$cordovaFile.writeFile(cordova.file.dataDirectory, "file.txt", "text", true)
    .then(function (success) {
      // success
    }, function (error) {
      // error
    });
$cordovaFile.writeExistingFile(cordova.file.dataDirectory, "file.txt", "text")
    .then(function (success) {
      // success
    }, function (error) {
      // error
    });
// READ
$cordovaFile.readAsText(cordova.file.dataDirectory, $scope.inputs.readFile)
    .then(function (success) {
      // success
    }, function (error) {
      // error
    });
// MOVE
$cordovaFile.moveDir(cordova.file.dataDirectory,  "dir",  cordova.file.tempDirectory,
"new_dir")
```

```
        .then(function (success) {
            // success
        }, function (error) {
            // error
        });
    $cordovaFile.moveFile(cordova.file.dataDirectory, "file.txt", cordova.file.tempDirectory)
        .then(function (success) {
            // success
        }, function (error) {
            // error
        });
    // COPY
    $cordovaFile.copyDir(cordova.file.dataDirectory, "dir", cordova.file.tempDirectory, "new_dir")
        .then(function (success) {
            // success
        }, function (error) {
            // error
        });
    $cordovaFile.copyFile(cordova.file.dataDirectory, "file.txt", cordova.file.tempDirectory, "new_file.txt")
```

```
            .then(function (success) {
                // success
            }, function (error) {
                // error
            });
    });
});
```

5.3.2. 文件上传下载

$cordovaFileTransfer

方法(Methods)

```
download(url, targetPath, options, trustHosts)
```

从服务器下载文件

参数	类型	说明
url	String	文件服务器 URL 下载地址
targetPath	String	文件保存路径
options	Object	可选参数
trustAllHosts	Boolean	如果设置为 true，接受所有安全证书

返回值	类型	说明

success	Object	返回成功下载的文件路径和更多的信息
progress	Object	返回下载文件的进度

```
upload(server, targetPath, options, trustAllHosts)
```

上传文件到服务器

参数	类型	说明
url	String	服务器上传地址
targetPath	String	上传文件路径
options	Object	可选参数
trustAllHosts	Boolean	如果设置为 true，接受所有安全证书

返回值	类型	说明
success	Object	返回成功对象的文件路径和更多的信息
progress	Object	返回上传文件的进度

选项参数是一个 Object，有以下几个关键的属性：

属性	类型	说明
encodeURI	Boolean	编码服务器的 URL 使用 encodeURI。默认设置为 true
timeout	Integer	超时时间已毫秒为单位

代码实例

```
module.controller('MyCtrl', function($scope, $timeout, $cordovaFileTransfer) {
```

```
document.addEventListener('deviceready', function () {
    var url = "http://cdn.wall-pix.net/albums/art-space/00030109.jpg";
    var targetPath = cordova.file.documentsDirectory + "testImage.png";
    var trustHosts = true;
    var options = {};

    $cordovaFileTransfer.download(url, targetPath, options, trustHosts)
      .then(function(result) {
        // Success!
      }, function(err) {
        // Error
      }, function (progress) {
        $timeout(function () {
          $scope.downloadProgress = (progress.loaded / progress.total) * 100;
        });
      });
  }, false);
document.addEventListener('deviceready', function () {
    $cordovaFileTransfer.upload(server, filePath, options)
      .then(function(result) {
        // Success!
      }, function(err) {
```

```
            // Error
        }, function (progress) {
            // constant progress updates
        });
    }, false);
});
```

5.3.3. 对话框

$cordovaDialogs

方法(Methods)

```
alert(message, title, buttonName)
```

$cordovaDialogs 显示一个普通对话框

参数	类型	说明
message	String	对话框提示的一段文字
title	String	对话框标题 默认:alert
buttonName	String	对话框按钮名称 默认:ok

```
confirm(message, title, buttonArray)
```

$cordovaDialogs 显示一个带有指定消息和取消及取消按钮的对话框(可以自定义两个按钮的名称)

参数	类型	说明

message	String	对话框提示的一段文字
title	String	对话框标题 默认:alert
buttonArray	Array	按钮名名称 是一个数组 默认:[‘ok’,’Cancel’]

返回值 Integer 1或2取决于你点击了哪一个按钮

```
prompt(message, title, buttonArray, defaultText)
```

显示可提示用户进行输入的对话框

参数	类型	说明
message	String	对话框提示的一段文字
title	String	对话框标题 默认:alert
buttonArray	Array	按钮名名称 是一个数组 如:[‘ok’,’Cancel’]
defaultText	String	用户输入提示信息

返回值 **Object** 用于接受用户输入:**result.input1** 判断用户点击那哪一个按钮 返回一个索引:**result.buttonIndex**

```
beep(repetitions)
```

显示可提示用户进行输入的对话框

参数	类型	说明
repetitions	Integer	设置弹出对话框延迟以秒为单位

代码实例

```
module.controller('MyCtrl', function($scope, $cordovaDialogs) {
```

```
    $cordovaDialogs.alert('message', 'title', 'button name')
      .then(function() {
        // callback success
      });
    $cordovaDialogs.confirm('message', 'title', ['button 1','button 2'])
      .then(function(buttonIndex) {
        // no button = 0, 'OK' = 1, 'Cancel' = 2
        var btnIndex = buttonIndex;
      });
    $cordovaDialogs.prompt('msg', 'title', ['btn 1','btn 2'], 'default text')
      .then(function(result) {
        var input = result.input1;
        // no button = 0, 'OK' = 1, 'Cancel' = 2
        var btnIndex = result.buttonIndex;
      });
    // beep 3 times
    $cordovaDialogs.beep(3);
});
```

5.3.4. 拍照

```
$scope.Camera = function(){
                var options = {
                    quality: 100,
                    destinationType: Camera.DestinationType.DATA_URL,
                    sourceType: Camera.PictureSourceType.CAMERA,
                    allowEdit: false,
```

```
                encodingType: Camera.EncodingType.JPEG,
                targetWidth: 200,
                targetHeight: 200,
                //popoverOptions: CameraPopoverOptions,
                saveToPhotoAlbum: false
                //correctOrientation: true
            };


    $cordovaCamera.getPicture(options).then(function(imageData) {
                var image = document.getElementById('myImage');
                image.src = "" + imageData;

                $scope.logtext = $scope.logtext + "拍摄照片路径：" +
"" + imageData + "\n";
            }, function(err) {
                // error
            });

        };
```

5.3.5. 录音

注：该功能 iOS 暂不支持

```
    $scope.Media = function () {
        $scope.media = $cordovaMedia.newMedia("snare.mp3");
        $scope.msg = "开始录音";
        $scope.media.startRecord();
        var timer_b = $timeout(
            function () {
                $scope.media.stopRecord();
                $scope.msg = "结束录音";
            },
            10000
        ).then(
            function () {
                $timeout.cancel(timer_b);
            }
        );
    };
```

5.3.6. 获取存储空间大小

```
    $scope.sdromsize = function () {
        cordova.exec(function (res) {
            var ret =  JSON.parse(res);
```

```
                var roma = ret.romavailableSize;
                var romt = ret.romtotalsize;
                var sda = ret.sdavailableSize;
                var sdt = ret.sdtotalsize;
                var str = "内存剩余容量：" + roma + "b 内存总容量：" + romt +
"b 存储容量：" + sda + "b 存储总容量：" + sdt+"b";
                alert(str);
            }, function (err) {
                //获取不到登录信息
                alert(err)
            }, "SystemUtil", "getsdandromsize", []);
        }
```

5.3.7. 联系人

```
JReapApp.controller('Contacts',function($scope,$cordovaContacts,$cord
ovaCamera,$cordovaJReapExtend){
        $scope.logtext="";

        $scope.contact = {
          "displayName":"",
          "phoneNumbers": [
            {
              "value": "",
              "type": "phone"
            },
          ],
          "emails": [
            {
              "value": "abc@gmail.com",
              "type": "home"
            }
          ],
          "photos": [
            {
              "type": "base64",
              "value": ""

            }
          ]
        };
        $scope.ImageData = "";
        $scope.camera = function(){
          var options = {
```

```
            destinationType: Camera.DestinationType.FILE_URL,
            sourceType: Camera.PictureSourceType.CAMERA,
            allowEdit: true,
            encodingType: Camera.EncodingType.JPEG,
            targetWidth: 100,
            targetHeight: 100,
            //popoverOptions: CameraPopoverOptions,
            saveToPhotoAlbum: false,
            correctOrientation: true
        };

        $cordovaCamera.getPicture(options).then(function(imageData) {
          var image = document.getElementById('headPortrait');
          image.src = imageData;
          $scope.ImageData = imageData;
        }, function(err) {
          // error
        });
      }

      $scope.save = function(){
      $scope.logtext = "";//重置信息
      var regMobile = /^1\d{10}$/;
      var regEmail = /^(\w-*\.*)+@(\w-?)+(\.\w{2,})+$/;
      if($scope.contact.displayName ==""){
          $cordovaJReapExtend.system.showShort("请填写姓名", null,
null);
              return;
          }else if($scope.contact.phoneNumbers[0].value ==""){
              $cordovaJReapExtend.system.showShort("请填写电话", null,
null);
              return;
          }else
if(!regMobile.test($scope.contact.phoneNumbers[0].value)){
              $cordovaJReapExtend.system.showShort("请填写正确的电话",
null, null);
              return;
          }else if($scope.contact.emails[0].value ==""){
              $cordovaJReapExtend.system.showShort("请填写邮箱", null,
null);
              return;
          }else if(!regEmail.test($scope.contact.emails[0].value)){
              $cordovaJReapExtend.system.showShort("请填写正确的邮箱",
null, null);
```

```
                return;
            }
          $scope.contact.photos[0].value = $scope.ImageData;
          $cordovaContacts.save($scope.contact).then(function(result) {
            // Contact saved
            $scope.logtext = $scope.logtext + "保存成功";
            $("#logtext").html($scope.logtext);
          }, function(err) {
            // Contact error
            $scope.logtext = $scope.logtext + "保存失败" + err;
            $("#logtext").html($scope.logtext);
          });
        }
        $scope.getAll = function(){
        $scope.logtext = "";//重置信息
          var options = {};
          options.filter = "";
          options.multiple = true;
              $cordovaContacts.find(options).then(function(allContacts)
{ //omitting parameter to .find() causes all contacts to be returned
           for (var i in allContacts){
                                  $scope.logtext   =   $scope.logtext   +
allContacts[i].displayName+"<br/>";
            for(var x in allContacts[i].phoneNumbers){
                                    $scope.logtext   =   $scope.logtext   +
allContacts[i].phoneNumbers[x].value+"<br/>";
             }
            }
            $("#logtext").html($scope.logtext);
          });
        }
        $scope.call = function(tel){//

    $cordovaJReapExtend.system.call(tel,function(data){},function(err
or){});
        }
          });
```

联系人页面

```
<ion-view view-title="联系人">
  <ion-content      style="padding:      15px;"      id="diy-content"
class="diy-content-center">
       <div style="border: 1px solid #ccc;border-radius: 15px; height:
96px;width:96px;margin:0 auto;overflow: hidden" ng-click="camera()">
```

```
        <img src="img/head_portrait.jpg" alt="" style="height: 100%;"
id="headPortrait"/>
      </div>
     <div style="width: 100%;margin:25px 0">
      <label class="item item-input">
        <input      type="text"      placeholder="     姓     名     "
ng-model="contact.displayName">
      </label>

      <label class="item item-input">
        <input      type="text"      placeholder="     电     话     "
ng-model="contact.phoneNumbers[0].value">
      </label>
      <label class="item item-input">
        <input      type="text"      placeholder="     邮     箱     "
ng-model="contact.emails[0].value">
      </label>
      <button class="button button-full button-assertive diy-button"
ng-click="call(10086)">
        拨打电话：10086
      </button>
     </div>

   <button      class="button      button-block      button-positive"
ng-click="save()">
     保存
   </button>
   <button      class="button      button-block      button-positive"
ng-click="getAll()">
     获取通讯录
   </button>
   <div id="logtext">{{logtext}}</div>
  </ion-content>
</ion-view>
```

5.4. 系统相关接口

[accountRestWebService]

接口名称	路径	参数	返回值	接口描述
获取某账户信息接口	/account/{id}	id:账号 id	AccountVO Json Object	通过帐号 ID 获取账户信息
修改密码接口	/account/modifyPassword	accountVO	ResultVO json object	修改用户登录密码
修改账户信息接口	/account/modify	accountId：用户 ID type 类型：1.电话 2.	ResultVO json object	修改账户信息

		邮箱 txtValue：修改后的值		
机构信息接口	/account/select_org/{comp_id}	comp_id：当前登录人的单位 cur_org_id：当前机构 id	ResultVO json object	机构信息 按照层级存储
获取机构下用户列表接口	/account/query	comp_id：单位 id org_id：机构 id keyword：查询关键字	ResultVO json object	通过机构 id 获取账户信息
数据资源授权接口	/account/data_auth	sys_id：系统 id account_ids：账号 id data_res_id：数据资源 id	String	数据资源授权

[filerest]

接口名称	路径	参数	返回值	接口描述
文件上传接口	/filerest/upload	request	ResultVO json object	文件上传

[messageRestWebService]

接口名称	路径	参数	返回值	接口描述
发布通知接口	/save_message	pushMessageVO	ResultVO Json Object	发布通知
删除消息接收人	/recipient/{msgId}/{accountId}	msgId：消息 id accountId：接收人 id	ResultVO Json Object	删除某条消息的被指定的某个接收人
获取消息详情	/message/{id}	id:消息 id	ResultVO Json Object	通过消息 ID 获取消息详细信息
获取消息列表	/message_list/{account_id}	account_id：接收人 id page：页码 pagesize：每页条目数	PageResultVO json object	获取某用户所接收到的消息分页列表
获取未读消息数	/unread_msg_num/{account_id}	account_id：接收人 id	ResultVO json object	获取某用户未读消息的记录总数目
获取消息和任务列表	/msg_task_list/{account_id}	account_id：接收人 id pagesize：获取多少条	PageResultVO json object	获取某接收人接收到的消息和任务列表

[mobileRestWebService]

接口名称	路径	参数	返回值	接口描述
登录接口	/login_info	login_name password	Account Json Object	提供用户名和加密后的密码
获取账户信息	/colleague/{id}/{cid}	id：登录人 id cid：同事 id	AccountVO Json Object	获取指定用户信息并验证是否拥有某同事的访问权限
绑定推送的通道	/bind_channel	vo：MobileDeviceInfoVO	ResultVO Json Object	登录后，绑定推送的通道 id，账户和设备信息为必须数据
获取资源	/get_resource	Null	ResultVO Json	获取资源列表

			Object	
资源授权	/lisence	resourceLicense ： ResourceLicenseVO	ResultVO json object	用户授权自己的资源（日报、任务等等）可供某些人查阅
获取资源授权信息	/get_license/{license_code}	license_code：资源编码 licensor：授权人 id	ResultVO json object	获取某用户的某资源的授权信息
判断资源授权	/validate_license/{license_code}	license_code：资源编码 licensor：授权人 id licensee：被授权人 id	ResultVO json object	判断某用户（被授权人）是否拥有某用户（授权人）的某资源的访问权限
判断资源授权	/validate_license	licensor：授权人 id licensee：被授权人 id	ResultVO json object	判断某用户（被授权人）拥有某用户（授权人）的哪些资源的访问权限

[questionRestWebService]

接口名称	路径	参数	返回值	接口描述
问题反馈	/save_question	questionvo：QuestionVO	ResultVO json object	提交反馈的问题
问题反馈列表	/question_list/{account_id}	account_id：账户 id page：当前页 pagesize：每页显示条数	PageResultVO json object	基于帐号 ID 分页查询问题反馈列表

[systemRestWebService]

接口名称	路径	参数	返回值	接口描述
获取系统版本号	/getNewestVersion		VersionInfoVO	获取系统最新的版本号

[microAppRestService] 微应用 REst 服务接口（componentLibRestWebService）

接口名称	路径	参数	返回值	接口描述
获取自己已添加的微应用接口	/getmricoapps	account_id:账号 id	ResultVO<Object>	获取自己已添加的微应用
移除微应用接口	/removemciroapp	component_key：微应用标识（id:version, 如 business:v1.0 account_id:账号 id	ResultVO<Object>	移除不需要的微应用，移除并不会删除微应用，只是从我的微应用中心列表中移除。
更新微应用接口（回调）	/updatemcripapp	component_key：微应用标识（id:version, 如 business:v1.0） account_id:账号 id	ResultVO<Object>	更新新版本微应用，替换原来的旧版本，更新成功后回写 **MicroAppLink** 表，包括新增或修改
获取微应用更新列表接口	/getmcripapps4up	component_keys ： 已添加的微应用标识（ id:version, 如 business:v1.0），多个以逗号分隔	ResultVO<Object>	获取微应用更新列表，仅获取已添加应用的更新列表。
获取必须添	/getmcripapps4ne	account_id:账号 id	ResultVO<Obj	获取必须添加微应用列

加的微应用列表接口	c	component_keys：已添加的微应用标识（id:version，如business:v1.0），多个以逗号分隔	ect>	表， 必须添加的微应用包括： * 未添加而标识为必须的（necessary 字段为 true） * microapplink 表中有记录而 app 端未添加的 * 已经添加标识为必须的且有更新的
获取可添加的微应用列表接口	/getmcripapps4add	component_keys：已添加的微应用标识（id:version，如business:v1.0），多个以逗号分隔	ResultVO<Object>	获取微应用更新列表，仅获取已添加应用的更新列表。
下载微应用接口	/download/{component_key}	component_key：已添加的微应用标识（id:version，如business:v1.0）	Response	下载微应用

5.5. 其他

5.5.1. 获取登录信息

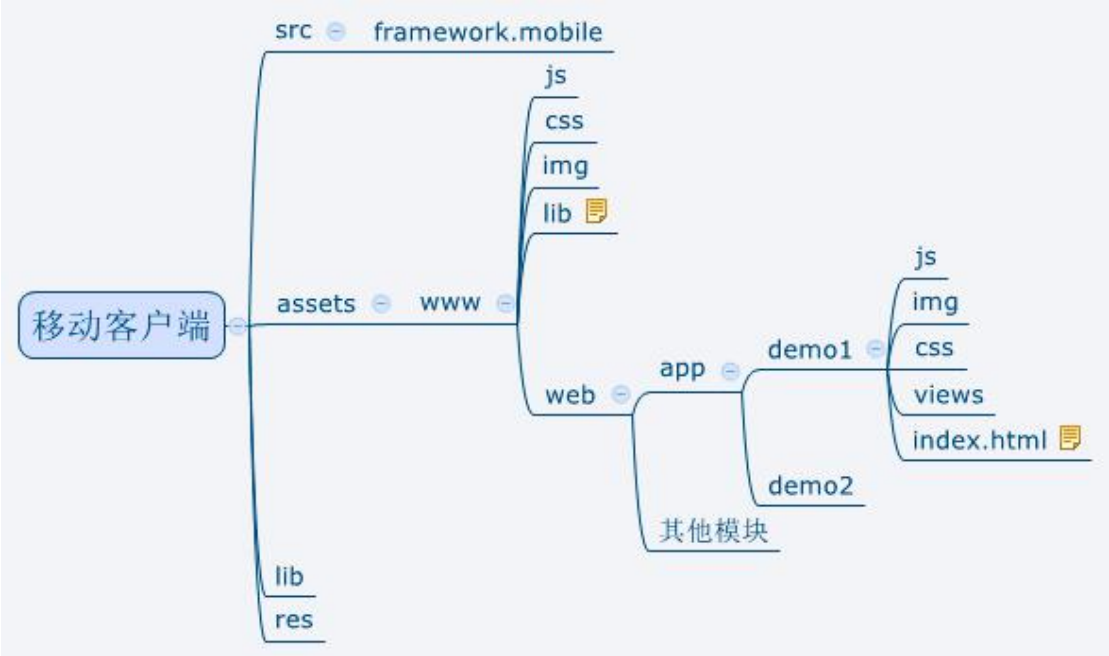
```
JReap.system.onAsyncFetchLogin(function (id) {
    alert(id);
});
```

5.5.2. 服务调用示例

```
cdvService.ready().then(function(){
                cdvService.sendReq({
    url:'camel/mapi/messageRestWebService/message/'+$stateParams.id,
                    method:'GET',
                    data:{"account_id": userId},
                    onSuccess:function(data){
                    $scope.messageDetail = data.result;
                },
                    onError:function(data){
                        $cordovaJReapExtend.system.showShort("数据加载失
败，原因："+data, null, null);
                    }
                });
            });
```

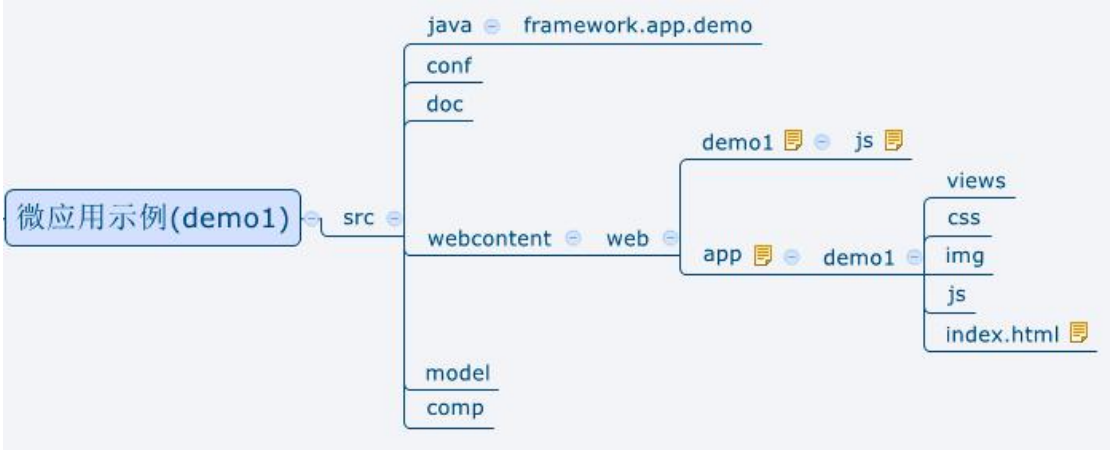
6. 约定

6.1. 移动端目录结构约定



注：android 与 iOS 原生代码有些许差异，www 目录保持一致

6.2. 微应用目录结构约定



微应用图标放在 web/app/目录下

微应用配置文件目录结构：

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<component id="微应用 id" type="microapp"><!—微应用 type 为固定值-->

<attribute name="webcontentpath" value=""/>

```
    <categoryid></categoryid>
    <categoryname></categoryname>
    <code>微应用 code</code>
    <department></department>
    <desc4ver></desc4ver>
    <description></description>
    <developer></developer>
    <name>微应用名称</name>
    <svn></svn>
    <uidir>web/bpm</uidir>
    <version>2.0</version>
    <isdeploy>false</isdeploy>   <!—是否立即部署-->
    <microapp>
        <icon>图标文件【默认为 icon.png】</icon>
        <entrance>入口地址</entrance>
        <necessary>true/false</necessary><!—是否为必须的微应用-->
    </microapp>
</component>
```

6.3. 数据表与实体之间映射约定

以下为实体与数据表之间的映射约定，如果实体某部分不符合约定，就需要对该部分编制 JPA 注解。

1) 实体类名由大小写字母组成，其对应的数据表表名约定为功能组前缀及实体类名的大写字母前加“_”。如实体类名为 BusinessSystem,对应的数据表名为 security_business_system,其中[security]为组织权限功能组名称；

2) 实体属性名由大小写字母组成，其对应的数据表字段名约定为属性名的大写字母前加“_”。如实体的属性名为 DictName，对应的字段名为 dict_name；

3) 标识实体唯一的属性名为 Id，对应数据表主键字段名为“ID”；如果实体 Id 属性类型为 UUID，对应数据表类型为 varchar2（oracle）；

4) 数据表自关联父外键字段名为“parent_id”；

5) 数据表非自关联外键字段名为“关联表名+_+id”。如数据表 security_business_system 关联数据表 security_organization 外键字段名为“org_id”；

6) 多对多中间表名为“a 表名+_+b 表名”，字段为“a 表名+_+id”和“b 表名+_+id”；

7) 实体内没有 Set 方法的属性不会映射为数据表字段；

8) 枚举类型的属性值在数据库中以数字方式存储。

说明：

1) 数据库表与实体之间对照不区分大小写；

2) 只要数据表符合映射约定，就可以不用再写 JPA 注解，如果某些表需要自定义，须在实体属性中用 JPA 声明映射关系，建议在 get\set 方法中定义注解；

3) 本约定说明是基于 JPA2.0 进行阐述。

6.4. 应用服务约定

1) 服务接口命名：I+名称＋Service,实现命名：名称＋Service；

2) 服务接口约束：应用服务实现标记为@Service，Spring 容器将自动识别并进行装载；@Description 标记可用于描述接口用途。实现内部引用的其他应用服务接口或数据访问层接口，统一采用@Resource 注解；

3) 服务接口其他约定：应用与工作流的服务接口需继承 DefaultProcessHandler 类，并实现 IBizProcessHandler 接口。其他领域服务或应用服务无此约束；

4) Dto 对象命名：一般为实体名称＋Dto；

5) Dto 对象约束：Dto 对象需实现 ITransferObject 标识接口。

说明：

1） 服务实现标注@Service 注解，服务将自动注入到 IOC 容器中，否则的话需要人工编写代码或配置文件来注入到容器。

6.5. 领域对象约定

1） 领域层由值对象、实体、聚合根、领域服务组成，所有业务的处理逻辑、业务规则、业务一致性保证都须在领域对象内实现。值对象所有属性对外建议只读（写保护），实体和聚合根除描述性属性外建议所有属性对外只读（写保护）。聚合内非聚合根对象可以对本聚合内的聚合根直接引用，聚合内的实体对象可以直接引用其他聚合的聚合根对象。

6.6. 数据访问服务约定

1） 数据访问服务（DAO）采用接口＋实现的方式；

2） 服务接口从“IBaseDAO <T, ID>”接口继承或从已继承“IBaseDAO < T, ID >”的接口继承；

3） 接口命名：I+实体类名＋DAO, 实现命名：实体类名＋DAO；

服务接口实现：数据访问服务实现可继承 DefaultDataSourceDAO ，或从已经继承 AbstractBaseDAO 的类继承。

7. 常见问题

7.1. 基准请求路径配置

android 在 config.xml 配置文件中配置：

```
<preference name="rootbasepath"
value="http://192.168.15.93:8080/jreap-4.0/" />
```

iOS 在 AppConfig.h 中配置：

```
#define WEB_BASE_ADDRESS @"http://192.168.23.23:8080/jreap-4.0/"
#define WEB_ADDRESS @"http://192.168.23.23:8080/jreap-4.0/camel"
```

7.2. APP 版本维护

android 在 AndroidManifest.xml 中配置 versionName

iOS 在 Info.plist 中配置：

CFBundleShortVersionString 和 CFBundleVersion

服务端在 web/mobile/upgrade/version.xml 文件中维护版本信息，当前版本和服务端版本不一致，会弹出更新应用的信息，用户可选择下载更新应用

7.3. 权限配置

android 权限配置在 AndroidManifest.xml 文件中配置，例如：

```
<!-- 通讯录 -->
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

IOS 权限配置在 Info.plist 中配置，例如：

```
<key>NSContactsUsageDescription</key>
<string>获取通讯录</string>
```

7.4. 服务端部署

1. 数据库安装

   数据库安装有两种方式，一种直接导入平台的 dmp 文件，另一种是运行平台提供的数据库安装脚本。

   - 导入 dmp 文件：导入发布包/dbbak/frameworkminisys.dmp 文件到数据库中，此种方式只对 oracle 数据库有效。
   - 运行脚本：数据库建好用户后，使用新建用户登录，依次导入发布包/dbback 下的“pub 相关表.sql”、“脚本.sql”、“jreap 初始功能资源-菜单导入.sql”三个文件；如果数据库为 Sqlserver，需要导入“兼容 Sqlserver 建视图脚本.txt”。

2. 数据库配置

   修改 WEB-INF/classes/bitronix-datasources.properties 文件：

```
resource.defaultDS.driverProperties.user = jreap
resource.defaultDS.driverProperties.password = jreap
resource.defaultDS.driverProperties.URL =
jdbc:oracle:thin:@192.168.1.1:1521:ORCL
```

   - 修改数据库连接 URL、数据库用户名和密码。

3. 开发模式配置

   修改 WEB-INF/classes/resource.properties：

```
devMode=false
```

- 生产环境需要修改 devMode 为 false,使加载 js 文件能从缓存中获取。

4. 日志配置

修改 WEB-INF/classes/ logback.xml 文件：

```
<substitutionProperty name="log.filepath"
value="d:\\logback_logspace"/>
    <!-- dao输出到文件 -->
<!--    <appender name="logfile-dao"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator name="myEva_dao">
                <expression>message.contains("dao.impl")</expression>
            </evaluator>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
        <encoding>utf-8</encoding>
        <file>${log.filepath}/${applicationName}_dao.log</file>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

    <fileNamePattern>${log.filepath}.%d{yyyy-MM-dd}_dao.log.zip</file
NamePattern>
        </rollingPolicy>
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>%date [%thread] %-5level %logger{80}
- %msg%n</pattern>
        </layout>
    </appender> -->

    <!-- service输出到文件 -->
<!--    <appender name="logfile-service"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator name="myEval_service">

    <expression>message.contains("service.impl")</expression>
            </evaluator>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
        <encoding>utf-8</encoding>
        <file>${log.filepath}/${applicationName}_service.log</file>
```

```
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

    <fileNamePattern>${log.base}.%d{yyyy-MM-dd}_service.log.zip</file
NamePattern>
        </rollingPolicy>
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>%date [%thread] %-5level %logger{80}
- %msg%n</pattern>
        </layout>
    </appender> -->

    <!-- action输出到文件 -->
<!--    <appender name="logfile-action"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator name="myEval_action">
                <expression>message.contains("action")</expression>
            </evaluator>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
        <encoding>utf-8</encoding>
        <file>${log.filepath}/${applicationName}_action.log</file>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

    <fileNamePattern>${log.base}.%d{yyyy-MM-dd}_action.log.zip</fileN
amePattern>
        </rollingPolicy>
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>%date [%thread] %-5level %logger{80}
- %msg%n</pattern>
        </layout>
    </appender> -->
<!--    <appender-ref ref="logfile-dao"/>
        <appender-ref ref="logfile-service"/>
        <appender-ref ref="logfile-action"/> -->
```

- 修改“log.filepath”的值为日志文件的保存路径
- 将“logfile-dao”、“logfile-service”、“logfile-action”三个 appender 的注释全部放开，再将下面的配置“<appender-ref ref=″logfil

e-dao"/><appender-ref ref="logfile-service"/><appender-ref ref="logfile-action"/> ”的注释放开，这样就能在日志路径下产生* _dao.log、* _service.log、* _action.log 的三种文件，分别记录 dao、service、action 包下的日志。

- 可以不使用默认的配置，自定义日志配置，如增加 appender 记录数据库日志等，只要符合 logback 的规范即可。

7.5. 常用请求地址配置

android 在 AppConfig.java 中配置

IOS 在 AppConfig.h 中配置