

可以从文章末尾的参考资料中进行学习

Android 路由框架ARouter详细使用教程

版本: 2019/2/3-1(12:28)

- Android 路由框架ARouter详细使用教程
 - 路由库
 - 设计方案
 - 路由注册
 - 汇总路由表
 - ARouter
 - 基础入门
 - 依赖添加
 - SDK初始化
 - 路径统一管理
 - 增加注解
 - 发起路由操作
 - 混淆处理
 - 路由表自动加载
 - 基础用法
 - 跳转到Activity
 - 跳转到Fragment
 - 进阶用法
 - 监听路由操作/降级策略
 - 降级策略
 - 拦截器
 - 拦截器的process没有调用
 - 跳转到Uri指定的目标
 - 带参数跳转并获取到参数
 - 转场动画
 - 低版本动画
 - 高版本动画(API >=16)
 - 共享元素动画
 - Uri跳转
 - 网页url跳转Activity
 - 网页url跳转-携带常用类型参数

- 网页url跳转-传递json自动转为自定义对象
- 跳转拦截
- 报错/处理
 - ARouter::There is no route match the path
- 总结
- 知识储备
- 参考资料

1、什么是路由？

1. 根据路由表将页面请求分发到指定页面
2. 映射页面跳转关系，也包含跳转相关的一切功能。

2、使用场景

1. App接收到一个通知，点击通知打开App的某个页面
2. 浏览器App中点击某个链接打开App的某个页面
3. 运营活动需求，动态把原生的页面替换成H5页面
4. 打开页面需要某些条件，先验证完条件，再去打开那个页面
5. 不合法的打开App的页面被屏蔽掉
6. H5打开链接在所有平台都一样，方便统一跳转
7. App存在就打开页面，不存在就去下载页面下载,只有Google的App Link支持

3、为什么要有路由？

1. 显示Intent：项目庞大以后，类依赖耦合太大，不适合组件化拆分
2. 隐式Intent：协作困难，调用时候不知道调什么参数
3. 每个注册了Scheme的Activity都可以直接打开，有安全风险
4. AndroidManifest集中式管理比较臃肿
5. 无法动态修改路由，如果页面出错，无法动态降级
6. 无法动态拦截跳转，譬如未登录的情况下，打开登录页面，登录成功后接着打开刚才想打开的页面
7. H5、Android、iOS地址不一样，不利于统一跳转
8. 复杂的业务场景下（比如电商），灵活性比较强，很多功能都是运营人员动态配置的，比如下发一个活动页面，事先并不知道具体的目标页面，提前做好页面映射，便可以自由配置。
9. App一般都会走向组件化、插件化的道路，而组件化、插件化的前提就是解耦，首先要做的就是解耦页面之间的依赖关系。
10. 简化代码。数行跳转代码精简成一行代码。

4、怎样是好的路由库

1. 使用简单、入侵低、便于维护
- 2.

5、路由库的本质

1. 注册再转发, 围绕着转发可以进行各种操作, 拦截, 替换, 参数获取等等
2. 其他Apt、Rxjava都只是为了方便使用

6、应用场景

1. 从外部URL映射到内部页面，以及参数传递与解析
2. 跨模块页面跳转，模块间解耦
3. 拦截跳转过程，处理登陆、埋点等逻辑
4. 跨模块API调用，通过控制反转来做组件解耦

路由库

1、目前市面上路由库有哪些？

1. 阿里 ARouter
2. Airbnb DeepLinkDispatch
3. 天猫 统跳协议
4. ActivityRouter
5. OkDeepLink

A	B	C	D	E	F	G
对比项	CC	得到DComponentForAndroid	ModularizationArchitecture	阿里Router (阿里多组件化方案的演进引擎)	美团F方案 (基于ReactNative)	ActivityRouter
开始时间	2017-11	2017-9	2017-1	2016-12	2016-9	2016-4
介绍文章	[wiki](https://github.com/luckybilly/CC/wiki)	[Android底库组件化方案实 践](https://www.jianshu.com/p/b1d7f758e6f4)	[Android底库参考(模块化、多运 程)](http://blog.sina.com.cn/article_118/28/android_mod ularization/)	[开源最真实: Android平台页面路由框架 Router](https://yq.aliyun.com/articles/110875?spm=5176.100263.searchlog.1.8a960)	[美团组件化实践之 路](https://juejin.im/post/5ab442501882812865ee4c)	[ActivityRouter路由框架: 通过注册实现动态打开 Activity](https://juejin.im/post/5b4162610a216006/D0A0tV) ActivityRouter/)
通信机制	组件总线	路由 + 接口下沉	路由 + 接口下沉	路由 + 接口下沉	路由 + 接口下沉	路由 + 接口方法
是否支持降级处理	✔️	❌	✔️	✔️	✔️	✔️
activity变量自动注入	❌	1. 通过get生成或注入代码取代@Cr. AutoRouteFactory.getInstance().createO. utroutev(this); 或者者useRouteActivity	✖️ 仅支持Activity	1. 通过get生成或解析参数的getCr.<T>. <onContext> 方法中用 @Router.getInstance().in<ject>(this); 实现自动注入	✔️ 支持Activity/Fragment	✖️ 仅支持Activity
startActivityForResult	✖️ 支持Activity/Fragment, 但不建议使用@Cr.<T>使用统一 的组件调用方式	✖️ 仅支持Activity	✖️ 仅支持Activity	✔️ 仅支持Activity	✔️ 支持Activity/Fragment	✔️ 仅支持Activity
调用方式 (页面跳转)	同步及链式返回结果或异步回调结果: CCResult result = CC.obtainBuilder("ComponentA").build().call()<T>; CC.obtainBuilder("ComponentA").build().call()<T>; CC.obtainBuilder("ComponentA").build().callAsync(fnc => { ComponentCallback()... });	onActivityResult返回结果 onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>:	✖️ 仅支持Activity	onActivityResult返回结果 onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>:	onActivityResult返回结果 onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>:	onActivityResult返回结果 onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>: onActivityResult()<T>:
调用方式 (调用服务)	与页面跳转相同	@Router.getInstance().getService(RendlessService.class).getSingleIn stance();	与页面跳转相同	@Router.getInstance().getService(RendlessService.class).getSingleIn stance();	与页面跳转相同	与页面跳转相同
组件向外提供服务	与页面跳转一致，在IComponent中实现	接口下沉到base中，组件中实现该接口并在ApplicationLike中添加代码注册到Router中	与页面跳转一致，实现一个对应的Action并在其所属的Provider中注册即可	与页面跳转一致，实现一个对应的Action并在其所属的Provider中注册即可	接口下沉到base中，组件中实现该接口并在ApplicationLike中间接间接管理类new ProviderManager.register(CorePip.class, new CorePipImpl());	在静态方法上添加参数暴露服务，但不支持返回值，且参数固定包名(context, bundle)
Fragment组件化支持	在IComponent中实现，支持后链Fragment内部功能调用	调用服务的方式实现，未支持后链Fragment内部的功能调用	不支持	调用服务的方式实现，未支持后链Fragment内部的功能调用	调用服务的方式实现，未支持后链Fragment内部的功能调用	不支持
组件自治运行方案	TransformAPI + AOP解耦组件类(IComponent接口实现类) 并封装成ComponentManager等，无侵入于端侧业务逻辑中	apt生成每个module的路由表由Cr.TransformAPI + javaassist实现 ApplicationLike的信息代码生成或到自定义定义application.onCreate方法中，无参于端侧业务逻辑列表	未实现自治运行，Cr.<T>. Action在其所属的Provider中注 册@Cr.<T>. Provider在其所属的ApplicationEngin中注册 @Cr.<T>. ApplicationEngin在App的Application中注册	1. apt生成每个module的路由表由Cr.<T>. Arouter初始化的 时自动生成describable而无需自己编写，通过反射进行统 一性注册	1. apt生成每个module的路由表由Cr.<T>. RouterInitial creator 创建，在IComponentPackage中定义和RouterInitialcreator 的签名@Cr.<T>. InbaseApplication()反射所有的包名来识别类 名称和RouterInitialcreator的签名，需要手动维护Com ponentPackage类中的包名列表	1. apt生成每个module的路由表由Cr.<T>. apt在 application(module)通过注解Cr.<T>. RouterInitialcreator 解决组件类问题
组件单独运行的方式	模块级library/application的方式，提供两种方式， Cr.<T>. module/build.gradle切块 ext.runsapp=@Cr.False @Cr.<T>. inLocal.properties中切块 module=module(application) (前者使用的方案，不 会交叉到任何包中去)	模块级library/application方式编译，在Module/build.gradle.properties中切块 module=module(application)=false	模块级library/application方式编译，框架本身没有提供切块 方式，开发者自行实现	模块级library/application方式编译，框架本身没有提 供切块方式，开发者自行实现	组件module将以library方式编译，整个应用壳子，可以 按需要将多个组件按照需求一起打包，@Cr.<T>好处是所有组件测试 时包名相同，能避免一些包名冲突等问题三三三SBR对包名的要求	1. UriScheme原生支持@Cr.<T>应用，组件同时安装在设备上即 可@Cr.<T>通过中介Activity转发:RouterActivity
跨app调用组件支持	✔️	❌	✔️	✔️	❌	✔️
跨app调用开关及权限设置	✔️	/	✔️	/	/	✔️
组件@apit运行时调用及其它特性	组件同时安装在设备上即可，实际开发中一般是当前正在开发的组件去安装其它组件的依赖，@Cr.<T>通过UriScheme，Service + LocalSocket实现，没有UriScheme使用时弹出的选择框	将需要调用的组件一起打包才能调用	组件同时安装在设备上即可，实际开发中一般当前正在开发的组件和其它app上的组件互相调用，@Cr.<T>通过AIDL实现	一起打包或者通过UriScheme来统一转发	将需要调用的组件一起打包才能调用	UriScheme原生支持@Cr.<T>应用，组件同时安装在设备上即可@Cr.<T>通过中介Activity转发:RouterActivity
组件依赖解耦	无源依赖，完全解耦	✔️	✔️	✔️	✔️	✔️
AOP支持	拦截器 + 组件内嵌Action执行AOP	❌	❌	✔️ 组件内部Action进行AOP	❌ 拦截器AOP	❌
拦截器	✔️	❌	❌	✔️	✔️	❌
组件调用时传入设置	✔️	❌	❌	✔️	✔️	❌
组件调用的取消	✔️	❌	❌	✔️	✔️	❌
动态注册/注销组件	✔️	✔️	❌	❌	❌	❌
特点	1. 可以跨app调用，初期实施时可单独靠路由组件运行 ②. 低耦合，组件调用方式和调用方式 不是是@Cr.<T> 调用，而是通过，需要调用方写少量代码 @Cr.<T>，相 应组件方，无参于端侧@Cr.<T>，提供 @Cr.<T>Processor 实现	1. 编码期间组件被强制通过操作进行解耦，避免直接调用其它组件的代码 @Cr.<T>，提供了类似Router的类和@Cr.<T>，组件自治运行，无参于端侧	1. 可以跨app，app内跨进程调用@Cr.<T>，组件运行在各自进程中，最初执行行代码打点切块时不需要造成任何@Cr.<T>，相 互而前部逐步完善起来是逐步完善，调用组件时候一拿到 @Cr.<T>，有各自自己的代码打点切块，调用RouterResponse，getData()去获取数据，但异步获取数据需 自己维护缓存	阿里的出品，使用者众多，QQ群交流也比较活跃 @Cr.<T>，自治组件调用方式和使用之前跟目前所有@Cr.<T> 相比各方面都更稳定且有加固+高兼容性内省@Cr.<T> 升级兼容负载	组件module可以以纯库以library方式编译，由统一的@Cr.<T>壳子来 安装测试，不需要自己写@Cr.<T>，@Cr.<T>好处是所有组件测试 时包名相同，能避免一些包名冲突等问题三三三SBR对包名的要求	1. 体系原生的组件化支持@Cr.<T>，通过注册解耦方法 的方案部署服务
组件定义代码侵入性	新增IComponent接口的方法实现来定义组件，侵入性低	注册定义方法和由参数自动注入，侵入性高	新增调用时实现类，侵入性低	注册定义方法和由参数自动注入，侵入性高	注册定义方法和由参数自动注入，侵入性高	注册定义方法和由参数自动注入，侵入性高
组件调用代码侵入性	高	高	** * ** * ** *	高	高	高
配置项	所有下沉接口，框架中相关接口的实现类为高	所有下沉接口，框架中相关接口的实现类为高	所有下沉接口，框架中相关接口的实现类为高	所有下沉接口，框架中相关接口的实现类为高	所有RouterInitialCreator类	框架中的所有类
老项目改造成本评估	低	一般	一般	一般	一般	低
方案使用的学习成本评估	低	一般	一般	一般	一般	一般
后续维护成本评估	低	一般	一般	一般	一般	一般
QQ群	686844583	693097923	无	692278657 / 336755078	108895931	无

设计方案

1、路由的设计方案需要涉及到哪些方面？

1. 路由注册
2. 路由查找
3. 路由分发

4. 动态替换
5. 动态拦截
6. 安全拦截
7. 方法调用
8. 结果返回
9. Module接入不同app

路由注册

1、AndroidManifest的缺点

AndroidManifest里面的activity声明scheme码是不安全的(所有App都可以打开这个页面)

2、三种路由注册方式:

1. 注解产生路由表, 通过DispatchActivity转发
2. AndroidManifest注册, 将其export=false, 但是再通过DispatchActivity转发Intent(天猫的做法), 好处是路由查找都是系统调用, 省掉了维护路由表的过程, 但是AndroidManifest配置还是比较不方便
3. 注解自动修改AndroidManifest, 可以避免路由表汇总的问题: 用自定义Lint扫描出注解相关的Activity, 然后在processManifestTask后面修改Manifest

汇总路由表

1、APT存在的问题

1. 使用Apt会造成每个module都要手动注册
2. 因为APT是在javacompile任务前插入了一个task, 所以只对自己的module处理注解

ARouter

1、ARouter的特点

1. 支持直接解析标准URL进行跳转, 并自动注入参数到目标页面中
2. 支持多模块工程使用
3. 支持添加多个拦截器, 自定义拦截顺序
4. 支持依赖注入, 可单独作为依赖注入框架使用
5. 支持InstantRun
6. 支持MultiDex(Google方案)
7. 映射关系按组分类、多级管理, 按需初始化
8. 支持用户指定全局降级与局部降级策略
9. 页面、拦截器、服务等组件均自动注册到框架
10. 支持多种方式配置转场动画
11. 支持获取Fragment

[阿里巴巴ARouter-Github链接点击这里](#)

基础入门

依赖添加

```
build.gradle(Module:app)
```

```
android {
    defaultConfig {
        ....
        javaCompileOptions {
            annotationProcessorOptions {
                arguments = [AROUTER_MODULE_NAME: project.getName()]
            }
        }
    }
}

dependencies {
    ....
    // 替换成最新版本，需要注意的是api要与compiler匹配使用，均使用最新版可以保证兼容
    implementation 'com.alibaba:arouter-api:1.4.0'
    annotationProcessor 'com.alibaba:arouter-compiler:1.2.0'
}
```

SDK初始化

在Application中进行ARouter SDK初始化，并且控制调试信息。

```
public class BaseApplication extends Application{
    //ARouter debug开关: true-open;false-close
    private boolean isDebugARouter = true;
    @Override
    public void onCreate() {
        super.onCreate();
        // 1.必须在init之前调用
        if (isDebugARouter) {           // These two lines must be written before init, otherwise
            ARouter.openLog();           // Print log
            ARouter.openDebug();         // Turn on debugging mode (If you are running in InstantRun
        }
        // 2.初始化
        ARouter.init(BaseApplication.this); // As early as possible, it is recommended to init
    }
}
```

路径统一管理

路径统一管理, 也可以不用该方式。

```
public class ARouterConstants {  
    //路径必须要最少是/xx/xx  
    public static final String ACTIVITY_URL_LOGIN= "/app/LoginActivity";  
}
```

增加注解

```
//使用该注解，并且指明path。  
@Route(path = ARouterConstants.ACTIVITY_URL_LOGIN)  
public class LoginActivity extends AppCompatActivity{  
}
```

发起路由操作

直接跳转或者携带参数。

```
// 1. 应用内简单的跳转(通过URL跳转在'进阶用法'中)  
ARouter.getInstance().build(ARouterConstants.ACTIVITY_URL_LOGIN).navigation();  
  
// 2. 跳转并携带参数  
ARouter.getInstance().build(ARouterConstants.ACTIVITY_URL_LOGIN)  
    .withLong("key1", 666L)  
    .withString("key2", "888")  
    .withSerializable("key3", new ProfileBean())  
    .navigation();
```

混淆处理

添加混淆规则(如果使用了Proguard)

```
-keep public class com.alibaba.android.arouter.routes.**{*};  
-keep class * implements com.alibaba.android.arouter.facade.template.ISyringe{*};  
  
# 如果使用了 byType 的方式获取 Service，需添加下面规则，保护接口  
-keep interface * implements com.alibaba.android.arouter.facade.template.IProvider  
  
# 如果使用了 单类注入，即不定义接口实现 IProvider，需添加下面规则，保护实现  
-keep class * implements com.alibaba.android.arouter.facade.template.IProvider
```

路由表自动加载

使用 Gradle 插件实现路由表的自动加载(在整个项目的build.gradle中添加)

```
apply plugin: 'com.alibaba.arouter'

buildscript {
    repositories {
        jcenter()
    }

    dependencies {
        classpath "com.alibaba:arouter-register:1.0.2"
    }
}
```

1. 可选使用
2. 该插件必须搭配 api 1.3.0 以上版本使用!
3. 通过 ARouter 提供的注册插件进行路由表的自动加载(power by AutoRegister), 默认通过扫描 dex 的方式 进行加载通过 gradle 插件进行自动注册可以缩短初始化时间解决应用加固导致无法直接访问 dex 文件, 初始化失败的问题,

基础用法

跳转到Activity

1、跳转到Activity

```
ARouter.getInstance().build(ARouterConstants.ACTIVITY_URL_LOGIN).navigation();
```

跳转到Fragment

2、跳转到Fragment

1. 先获取到Fragment
2. 在Activity中将该Fragment加载到容器中

```
//1. 获取到目标Fragment
Fragment fragment = (Fragment) ARouter.getInstance().build(ARouterConstants.FRAGMENT_URL).navigation()
//2. 在Activity中加载该Fragment
getSupportFragmentManager()
    .beginTransaction()
    .replace(R.id.container, fragment)
    .commit();
```

进阶用法

监听路由操作/降级策略

1、监听路由操作, 回调相应方法

```
ARouter.getInstance()  
    .build(ARouterConstants.ACTIVITY_URL_HOST)  
    .navigation(MDActivity.this, new NavigationCallback() {  
  
        @Override  
        public void onFound(Postcard postcard) {  
            // 找到目标  
        }  
  
        @Override  
        public void onLost(Postcard postcard) {  
            // 丢失目标  
        }  
  
        @Override  
        public void onArrival(Postcard postcard) {  
            // 已经跳转到目标页面  
        }  
  
        @Override  
        public void onInterrupt(Postcard postcard) {  
            // 跳转被拦截  
        }  
    });
```

降级策略

2、降级策略是什么？

1. 跳转过程中如果出现错误，可以进行处理，比如跳转到其他页面。
2. 处理方法1：利用坚挺路由操作的回调方法 `NavigationCallback`
3. 处理方法2：全局处理(实现 `DegradeService` 接口)

3、实现DegradeService进行全局处理

1. 如果同时有单个的 `NavigationCallback`，优先级比全局处理的高，不会再进入 `DegradeService`。
2. 实现 `DegradeService`，Route注解采用任意的Path都可以。
3. `onLost()`会在跳转失败后回调。


```
@Route(path = "/degrade/Service")
public class DegradeServiceImpl implements DegradeService {

    private static final String TAG = DegradeServiceImpl.class.getName();

    private Context mContext;

    @Override
    public void onLost(Context context, Postcard postcard) {
        Logger.t(TAG).d(postcard.toString());
        ARouter.getInstance()
            .build(ARouterUtils.ARouterConstants.ACTIVITY_URL_MAIN)
            .withString(MAIN_FRAGMENT_TYPE, FRAGMENT_URL_LOGIN)
            .navigation();
    }

    @Override
    public void init(Context context) {
        mContext = context;
    }
}
```

拦截器

1、拦截器的作用

1. 修改参数
2. 拦截跳转(比如未登陆时进行操作, 先跳转到登陆页面。)

2、拦截器的使用

```

/**
 * priority: 优先级, 适用于多个拦截器的情况。
 */
@Interceptor(priority = 8,name = "拦截器")
public class MyInterceptor implements IInterceptor{
    @Override
    public void process(Postcard postcard, InterceptorCallback callback) {

        // 如果目标是"影视页面"
        if(postcard.getPath().equals(ARouterConstants.FRAGMENT_URL_MOVIE)){
            // 如果没有登录, 跳转到登陆页面。
            if(true){
                postcard.setPath(ARouterConstants.FRAGMENT_URL_LOGIN);
            }
            // 也可以增加参数
            postcard.withString("account", "10001");
        }

        /**
         * 继续跳转或者终止跳转, 如果不调用, 路由直接结束。
         */
        // 1.继续跳转
        callback.onContinue(postcard);

        // 2.终止跳转
        //callback.onInterrupt(null)
        //callback.onInterrupt(new RuntimeException()) // 抛出异常
    }

    @Override
    public void init(Context context) {
        // 会在sdk初始化的时候调用且仅调用一次
        Log.d("feather","拦截器初始化");
    }
}

```

拦截器的process没有调用

3、在触发路由操作后, 拦截器的process方法并没有调用

1. 出现该场景是在路由到Fragment时出现, Activity不会出现该问题。
2. 根据官方的Issue和源码, Fragment采用绿色通道, 不会进行拦截, 因此无法使用拦截器拦截Fragment的跳转。
3. 解决办法:
 1. 使用PathReplaceService(只能根据判断来修改Path, 感觉用处不大)
 2. 不能直接navigation目标Fragment了, Navigation一个Activity, 并且将Fragment通过参数传入, 这样就能被拦截器所拦截, 并进行后续的处理。

跳转到Uri指定的目标

1、Activity在AndroidManifest中设置

```

<activity
    android:name=".LoginActivity"
    xxx>

    <intent-filter>
        <category android:name="android.intent.category.DEFAULT"/>
        <data
            android:scheme="feather"
            android:host="www.feather.com"
            android:port="1226" />
        </intent-filter>
    </activity>

```

2、Activity中指定Path

```

@Route(path = ARouterConstants.ACTIVITY_URL_LOGIN)
public class LoginActivity{
    xxx
}

```

3、通过Uri进行跳转

```

// 1. 默认路径
Uri uri = Uri.parse(ACTIVITY_URL_LOGIN);
ARouter.getInstance().build(uri).navigation();

// 2. 完整路径
Uri uri = Uri.parse("feather://www.feather.com:1226" + ACTIVITY_URL_LOGIN);
ARouter.getInstance().build(uri).navigation();

```

带参数跳转并获取到参数

1、携带参数跳转

```

ARouter.getInstance().build(ARouterConstants.ACTIVITY_URL_LOGIN)
    .withString("name", "feather")
    .withInt("age", 18)
    .withBoolean("male", true)
    .navigation();

```

2、获取到参数(名字对应)

```

@Route(path = ARouterConstants.ACTIVITY_URL_LOGIN)
public class LoginActivity extends AppCompatActivity{

    @Autowired
    public String name;
    @Autowired
    public int age;
    @Autowired
    public boolean male;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ARouter.getInstance().inject(this);
        //显示
        Logger.getLogger("Login").info("name = " + name + " age = " + age + " sex(is male) = "
    }
}

```

2、获取到参数(变量名和参数名一致)

```

@Route(path = ARouterConstants.ACTIVITY_URL_LOGIN)
public class LoginActivity extends AppCompatActivity{

    @Autowired
    public String name; //变量名必须要和参数名一致
    @Autowired

    public int age;
    @Autowired
    public boolean male;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //需要注入
        ARouter.getInstance().inject(this);
        //显示
        Logger.getLogger("Login").info("name = " + name + " age = " + age + " sex(is male) = "
    }
}

```

变量必须要为Public修饰

3、获取到参数(变量名和参数名不一致)

```

@Route(path = ARouterConstants.ACTIVITY_URL_LOGIN)
public class LoginActivity extends AppCompatActivity{

    @Autowired(name = "name")
    public String mName;
    @Autowired(name = "age")
    public int mAge;
    @Autowired(name = "male")
    public boolean mIsMale;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //需要注入
        ARouter.getInstance().inject(this);
        //显示
        Logger.getLogger("Login")
            .info("name = " + mName +
                "\n age = " + mAge +
                "\n sex(is male) = " + mIsMale);
    }
}

```

转场动画

低版本动画

使用withTransition方法，添加 进入动画，退出动画即可

```

ARouter.getInstance()
    .build(ARouterConstants.ACTIVITY_URL_PROFILE)
    //参数1为打开的Activity的进入动画，参数2为当前的Activity的退出动画
    .withTransition(R.anim.router_scale_in, R.anim.router_scale_out)
    .navigation(StarActivity.this);

```

高版本动画(API >=16)

withOptionsCompat添加ActivityOptionsCompat对象

// 设置动画效果

```
if (Build.VERSION.SDK_INT >= 16) {
    ActivityOptionsCompat compat = ActivityOptionsCompat.
        makeScaleUpAnimation(v, v.getWidth() / 2, v.getHeight() / 2, 0, 0);

    ARouter.getInstance()
        .build(ARouterConstants.ACTIVITY_URL_PROFILE)
        .withOptionsCompat(compat)
        .navigation(StarActivity.this);
} else {
    Toast.makeText(this, "API < 16,不支持新版本动画", Toast.LENGTH_SHORT).show();
}
```

共享元素动画

共享元素动画：需要在navigation中传入当前Activity

1-StarActivity(当前Activity)

// 1、共享元素(三个元素)

```
Pair<View, String> pairOne = new Pair<View, String>(holder.mHeadImg, "stars_head_img");
Pair<View, String> pairTwo = new Pair<View, String>(holder.mNameTxt, "stars_name_txt");
Pair<View, String> pairThreee = new Pair<View, String>(holder.mAgeTxt, "stars_age_txt");
```

// 2. API23才有共享元素动画

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
```

// 3. 进入和退出动画

```
getWindow().setEnterTransition(new Explode().setDuration(600).setInterpolator(new BounceInter
getWindow().setExitTransition(new Explode().setDuration(600));
```

// 4. 共享元素动画

```
ActivityOptionsCompat compat = ActivityOptionsCompat.
    makeSceneTransitionAnimation(StarActivity.this, pairOne, pairTwo, pairThreee);
```

// 5. ARouter进行跳转：带入参数、进行共享元素动画

```
ARouter.getInstance()
    .build(ARouterConstants.ACTIVITY_URL_PROFILE)
    .withOptionsCompat(compat)
    .withString("headImg", datas.get(position).getHeadImg())
    .withString("name", datas.get(position).getName())
    .withString("age", datas.get(position).getAge())
    .withString("profile", datas.get(position).getProfile())
    .navigation(StarActivity.this);
} else {
    Toast.makeText(StarActivity.this, "API < 23,不支持共享元素动画", Toast.LENGTH_SHORT).show();
}
```

2-ProfileActivity: 目标Activity

```

@Route(path = ACTIVITY_URL_PROFILE)
public class ProfileActivity extends AppCompatActivity {

    @Autowired
    public String headImg;
    @Autowired
    public String name;
    @Autowired
    public String age;
    @Autowired
    public String profile;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ARouter.getInstance().inject(this);

        //设置内容
        setContentView(R.layout.activity_md);

        //获取控件
        ImageView imageView = findViewById(R.id.md_imageview);
        TextView nameTextView = findViewById(R.id.md_name_text);
        TextView ageTextView = findViewById(R.id.md_age_text);

        // 共享元素
        ViewCompat.setTransitionName(imageView, "stars_head_img");
        ViewCompat.setTransitionName(nameTextView, "stars_name_txt");
        ViewCompat.setTransitionName(ageTextView, "stars_age_txt");

        /**
         * 设置头像、姓名和年龄
         */
        Glide.with(this).load(headImg).into(imageView);
        nameTextView.setText(name);
        ageTextView.setText(age);
    }
}

```

Uri跳转

网页url跳转Activity

1、网页Uri跳转Activity是什么？

1. 效果：点击一个web页面上面的链接，就能跳转到我们的APP页面。

2、网页url跳转Activity的思路

1. 创建中转Activity(SchemeFilterActivity)
2. AndroidManifest中对中转Activity进行设置

3. 在中转Activity中进行对应页面的跳转。

3、功能实现

1-中转Activity(SchemeFilterActivity.java)

```
public class SchemeFilterActivity extends AppCompatActivity{
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // 1. 获取到Uri
        Uri uri = getIntent().getData();
        // 2. 进行跳转
        ARouter.getInstance()
            .build(uri)
            .navigation();
        // 3.销毁该Activity
        finish();
    }
}
```

2-AndroidManifest.xml对SchemeFilterActivity进行配置

```
<activity android:name=".SchemeFilterActivity">
    <intent-filter>
        <data
            android:host="host"
            android:scheme="scheme"/>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
    </intent-filter>
</activity>
```

3-目标页面

```
@Route(path = ARouterConstants.ACTIVITY_URL_LOGIN)
public class LoginActivity{
    xxx
}
```

4-测试的web页面，将该html文件在手机浏览器中打开，并且点击跳转链接。


```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title></title>
  </head>


  <body>
    <h2>跳转测试</h2>
    <h2>自定义Scheme[通常来说都是这样的]</h2>
    <p><a href="scheme://host/app/LoginActivity">scheme://host/app/LoginActivity</a></p>

  </body>
</html>
```

网页url跳转-携带常用类型参数

在Html页面的跳转中给Url拼接参数

```
<p><a href="scheme://host/app/LoginActivity?name=feather&age=18&male=true">scheme://host/app/Lc
```



目标Activity接收参数。

```
@Autowired(name = "name")
public String mName;
@Autowired(name = "age")
public int mAge;
@Autowired(name = "male")
public boolean mIsMale;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ARouter.getInstance().inject(this);
    //xxx
}
```

网页url跳转-传递json自动转为自定义对象

跳转拦截

报错/处理

ARouter::There is no route match the path

解决办法:

1. 不同module的一级路径必须不同，否则会导致一个module中的一级路径失效。
2. 原因:
ARouter通过一级目录"app"找到了route，并且在groupIndex中删除了这个路径，代表已经加载到了内存。不同的module使用了相同的一级路径，在ARouter第一次寻找到route的时候便删除了这个一级路径的group，因为一级路径的重复，再调用另一个module的一级路径是"app"的路由时，由于之前Warehouse.groupsIndex已经删除，便导致了there's no route matched的错误。

总结

1、href跳转到app的流程

1. 系统浏览器发现要加载一个连接，是以arouter开头的协议，但是其自身无法处理该协议，此时就会把此协议往系统层抛
2. 系统会根据所有安装的app的清单文件决定要将此协议交给哪个app来处理。
3. 如果要在浏览器上访问，那么浏览器就要自己处理此协议，否在就会交给某个具体app处理

2、ARouter解决了WebView中URL跳转问题

1. 如果在WebView中需要跳转到一个Android页面，需要在shouldOverrideUrlLoading中拦截。但是业务的增多，会导致逻辑比较臃肿。
2. 如果要从其他App跳转到自己APP的某个页面，webView的方法也无法实现了。
3. ARouter通过在一个中转Activity统一调度，能简单安全的实现这种需求。

3、ARouter如何实现重定向？

1. 实现 PathReplaceService接口
2. 对URL按照一定规则进行处理。

4、ARouter全局降级策略是什么？

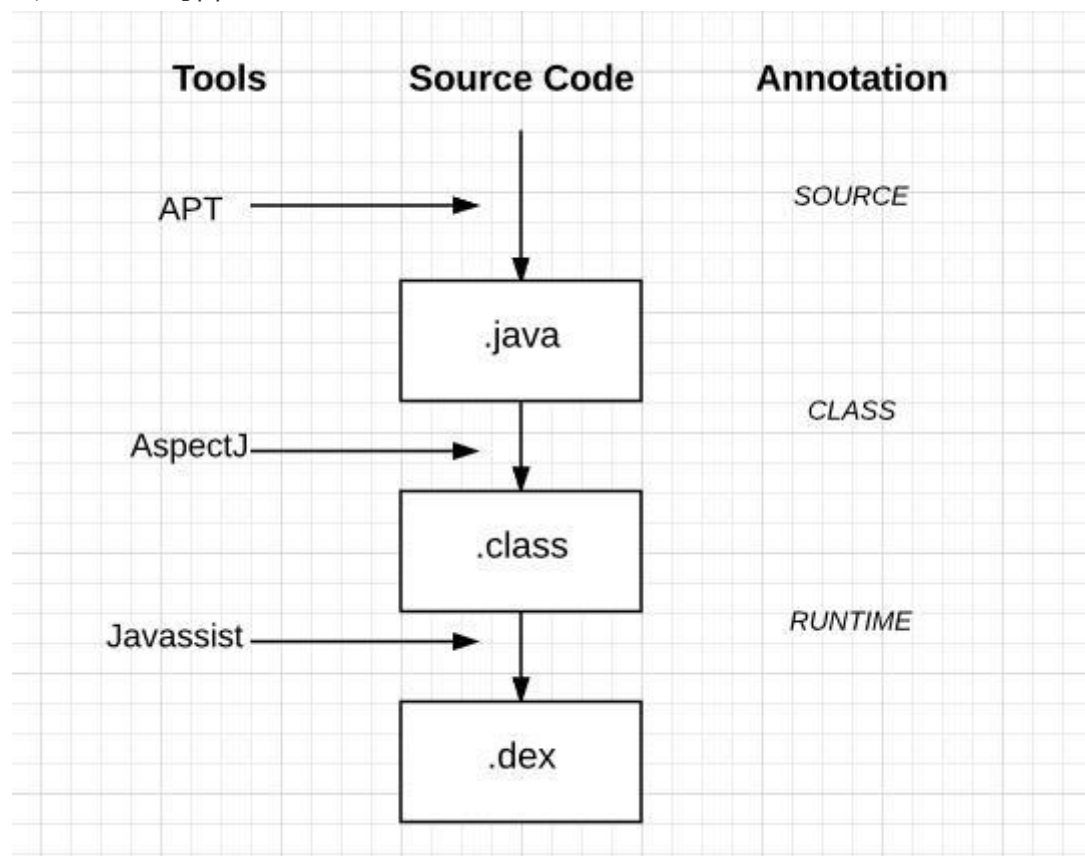
1. 如果出错或者页面不存在，不会出错，而是可以进行降级处理。
2. 通过 DegradeService 接口来实现该功能

知识储备

1、AOP是什么？

1. AOP:面向切面编程(Aspect-Oriented Programming)。AOP是把涉及到众多模块的某一类问题进行统一管理。而OOP如果是把问题划分到单个模块。
2. Android AOP就是通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。
3. 利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，提高开发效率。

2、AOP三剑客



3、APT是什么？

1. APT(Annotation Processing Tool 的简称), 可以在代码编译期解析注解, 并且生成新的 Java 文件, 减少手动的代码输入。
2. 现在很多主流库都使用APT, 比如 Dagger2, ButterKnife, EventBus3 等

参考资料

1. [Android跳转-ARouter详细使用教程](#)
2. [Android 路由框架ARouter最佳实践](#)
3. [ARouter解析三: URL跳转本地页面源码分析](#)
4. [Android 组件化 —— 路由设计最佳实践](#)
5. [Android 路由框架ARouter最佳实践](#)
6. [谈谈App的统一跳转和ARouter](#)
7. [安卓AOP三剑客:APT,AspectJ,Javassist](#)
8. [ARouter there's no route matched解决方法](#)