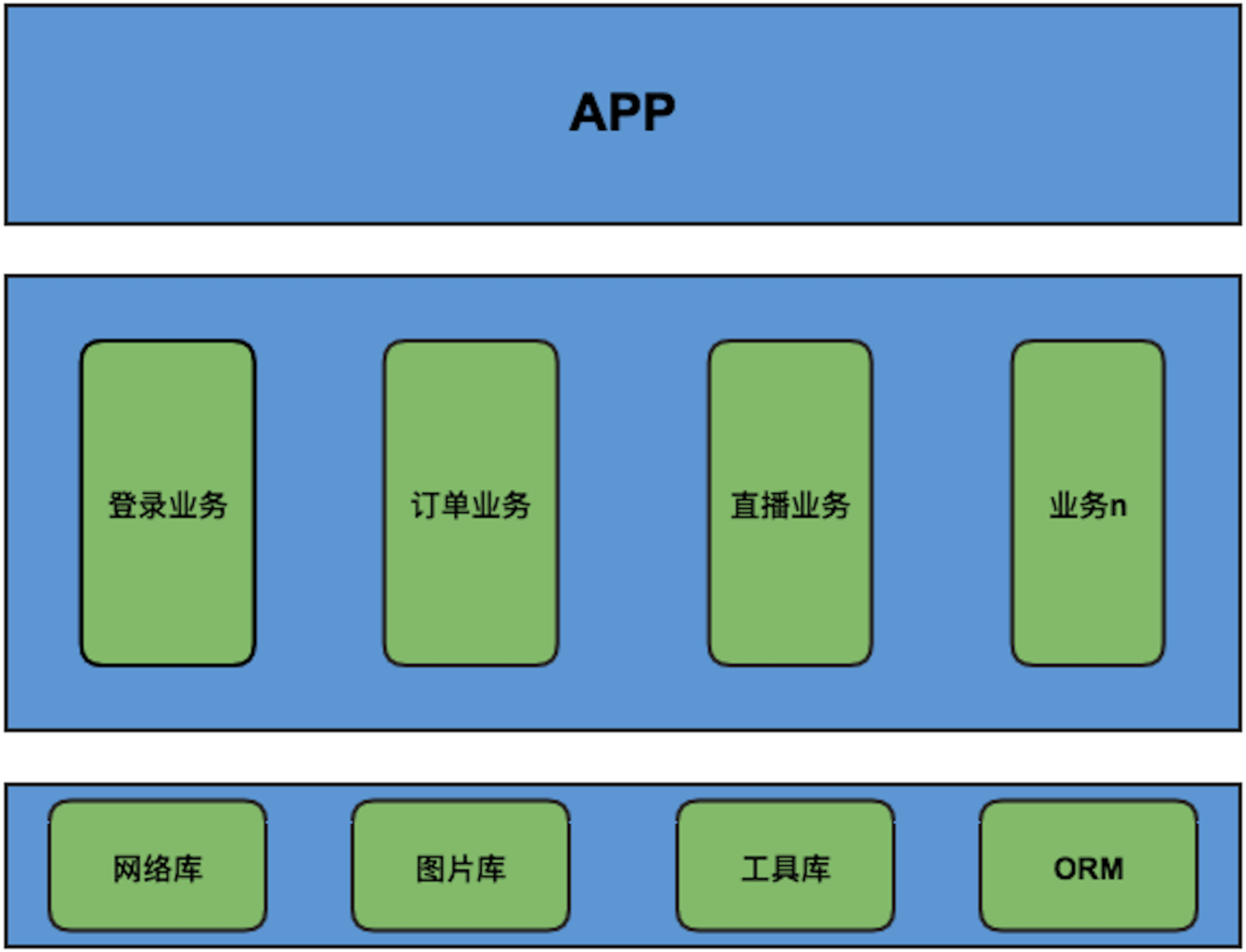
AOP技术在APP架构上的应用

刘小明


APP
登录业务
订单业务
直播业务
业务n
网络库
图片库
工具库
ORM

传统应用架构的困惑

- 模块独立
- 业务解耦
- 方便多团队协作
- 业务可灵活扩展

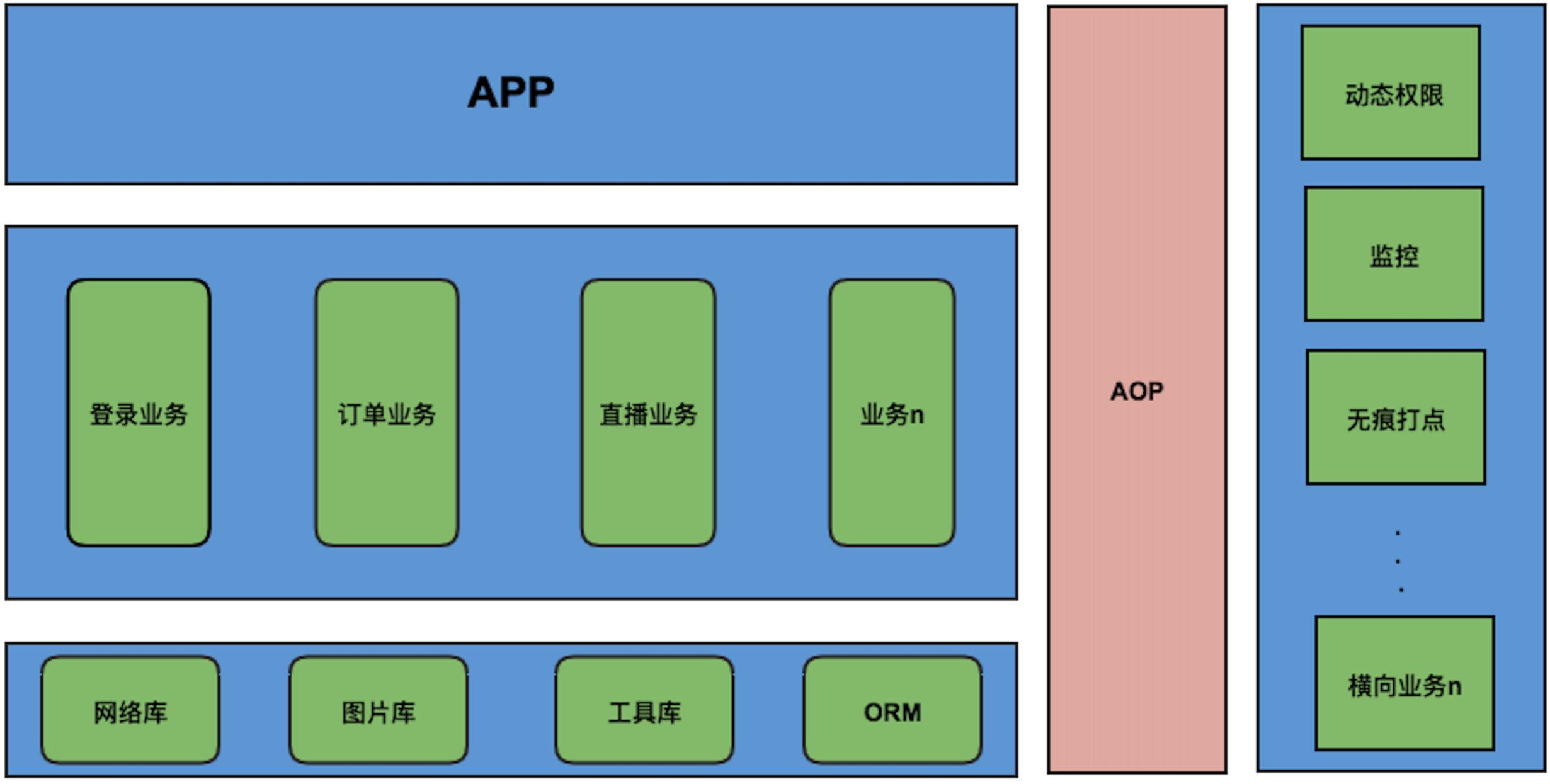
横向需求落地困难

—侵入业务模块

—易引入新Bug

—团队协作困难

—迭代慢，成本高


APP
登录业务
订单业务
直播业务
业务n
网络库
图片库
工具库
ORM
AOP
动态权限
监控
无痕打点
横向业务n

举个栗子

```java
public class CActivity extends AppCompatActivity {

    private static final int PERMISSION_REQUEST_CODE = 1;

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

        setContentView(R.layout.c_activity_layout);

//        doWork();

        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.CAMERA}, PERMISSION_REQUEST_CODE);
    }

    private void doWork() {

    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
        if (requestCode == PERMISSION_REQUEST_CODE) {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                doWork();
            } else {
//                do not work
            }
        } else {
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }
}
```

```
    private void doWork() {
        CheckPermission.instance(this).check(new PermissionItem(Manifest.permission.CAMERA), new PermissionListener() {
            @Override
            public void permissionGranted() {
//                do real work
            }

            @Override
            public void permissionDenied() {
//                do not work
            }
        });
    }
```

动态权限带来的困惑

- 对既有代码进行结构调整
- 依赖库代码调整
- 第三方库怎么办
- 调整成本大，时间周期长，无法满足版本快速迭代

Annotation方式申请动态权限

```
@NeedPermission(permissions = {Manifest.permission.CAMERA})
public class BActivity extends AppCompatActivity {
```

```
    @NeedPermission(permissions = {Manifest.permission.RECORD_AUDIO})
    private static void function(Context context) {
        ToastUtils.show(context, "function that need record audio permission");
    }
```

java接口方式注入权限

```
AOPSDK.addCheckPermissionItem(new CheckPermissionItem("com.hujiang.normandy.MainActivity", Manifest.permission.RECORD_AUDIO));
AOPSDK.addCheckPermissionItem(new CheckPermissionItem("com.hujiang.normandy.playActivity", Manifest.permission.CAMERA));
```

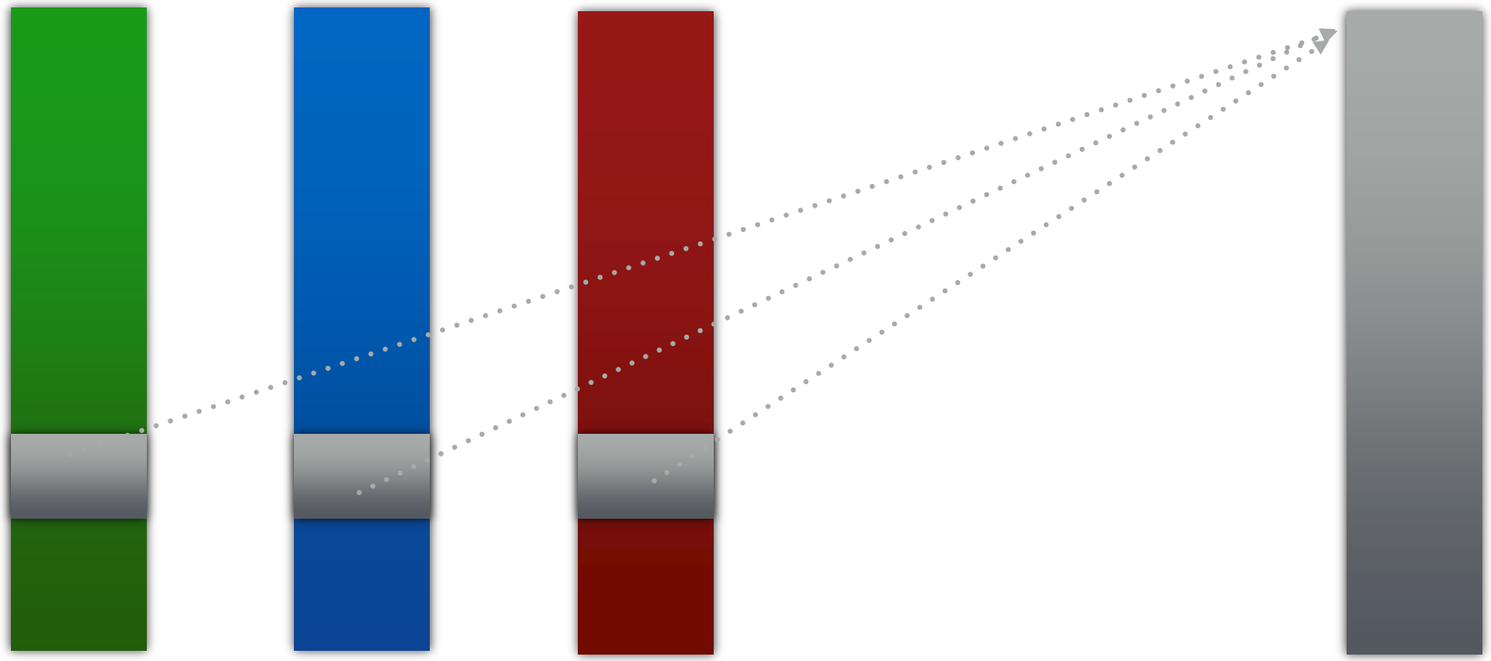
AOP代码直接切入方法

```
@Aspect
public class PermissionAspect {

    @Pointcut(value = "execution(* com.hujiang.normandy.permission.PermissionActivity.function(..))")
    private void pcFunction() {}

    @Around("pcFunction()")
    public void adFunction(final ProceedingJoinPoint joinPoint) throws Throwable {
        CheckPermission.instance(RunTimeManager.instance().getApplication())
                .check(new PermissionItem(Manifest.permission.CAMERA), new PermissionListener() {
                    @Override
                    public void permissionGranted() {
                        try {
                            joinPoint.proceed();
                        } catch (Throwable throwable) {
                            throwable.printStackTrace();
                        }
                    }

                    @Override
                    public void permissionDenied() {

                    }
                });
    }
}
```

- 动态权限
- 域名备份方案
- 性能监控
- 无痕打点
- log
- token验证
- 调试
- 测试
- 国际化版本开发
- Bug快速修复

什么是AOP

- 动态地将代码切入到类的指定方法、指定位置的编程思想
- 它是面向对象编程的一种补充



选择合适方式落地

- 高效，不影响软件运行效率
- 稳定，不影响软件质量
- 可维护，开发成本低，维护成本低
- 混淆不影响
- 可以切入第三方库
- 方便复用，可以作为独立第三方库存在
- 同时支持java, kotlin

AspectJ

AspectJX

基本概念

- @Aspect
- @Pointcut
- @After
- @Before
- @Around

切点匹配规则

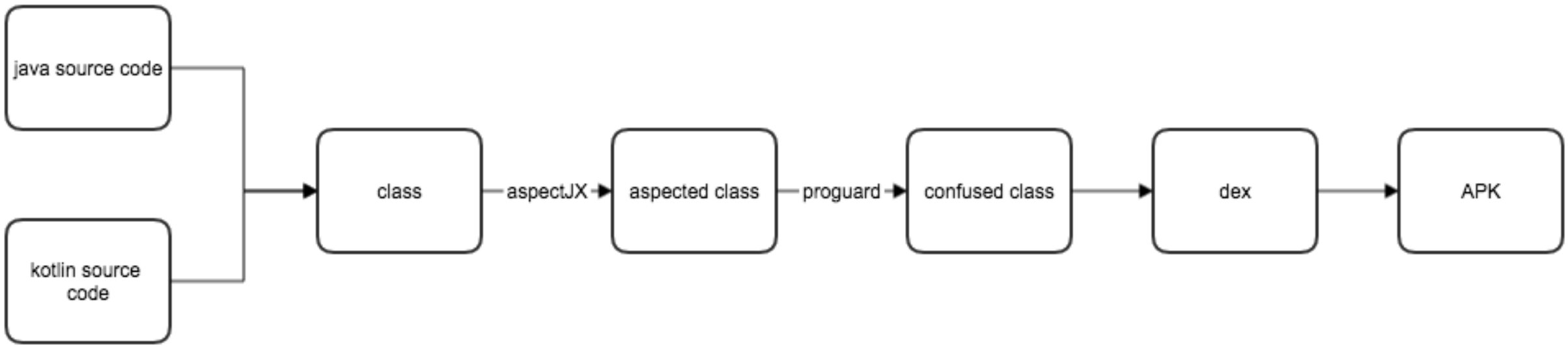
- 类正则方式：., .., ?, *
- args()
- ！
- target()

切入方式

- call
- execution

支持切点

- annotation
- 方法
- 构造方法
- 变量
- 异常


java source code
kotlin source code
class
aspectJX
aspected class
proguard
confused class
dex
APK


com.hujiang.wedjat.network
aop
HttpClientAspect
MseberaHttpClientAspect
OkHttp2Aspect
OkHttp3Aspect
URLConnectionAspect

```java
@Aspect
public class OkHttp3Aspect {
    @Pointcut(value = "execution(* okhttp3.RealCall.getResponseWithInterceptorChain())")
    private void pcExecute() {}

    @Around("pcExecute()")
    public Response adExecute(ProceedingJoinPoint joinPoint) throws Throwable {
        Call call = (Call) joinPoint.getTarget();
        final HealthyData healthyData = new HealthyData();
        if (call != null) {
            Request request = call.request();
            if (request != null) {
                healthyData.setUrl(request.url().toString())
                        .setMethod(request.method())
                        .setNetworkType(NetworkHelper.getNetworkType(NetworkMonitor.instance().getApplicationContext()))
                        .setServerIP(NetworkHelper.getIP(healthyData.getHost()));
            }
        }
        long startTime = System.nanoTime();
        try {
            Response response = (Response) joinPoint.proceed();
            long endTime = System.nanoTime();

            healthyData.setHttpCode(response.code())
                    .setResponseTime(TimeUnit.NANOSECONDS.toMillis(endTime - startTime));

            return response;
        } catch (Throwable e) {
            healthyData.setErrorType(NetworkErrorType.getErrorType(e));
            throw e;
        } finally {
            NetworkMonitor.instance().addHealthyData(healthyData);
        }
    }
}
```

Q&A