Android Testing

Gemini

https://github.com/geminiwen

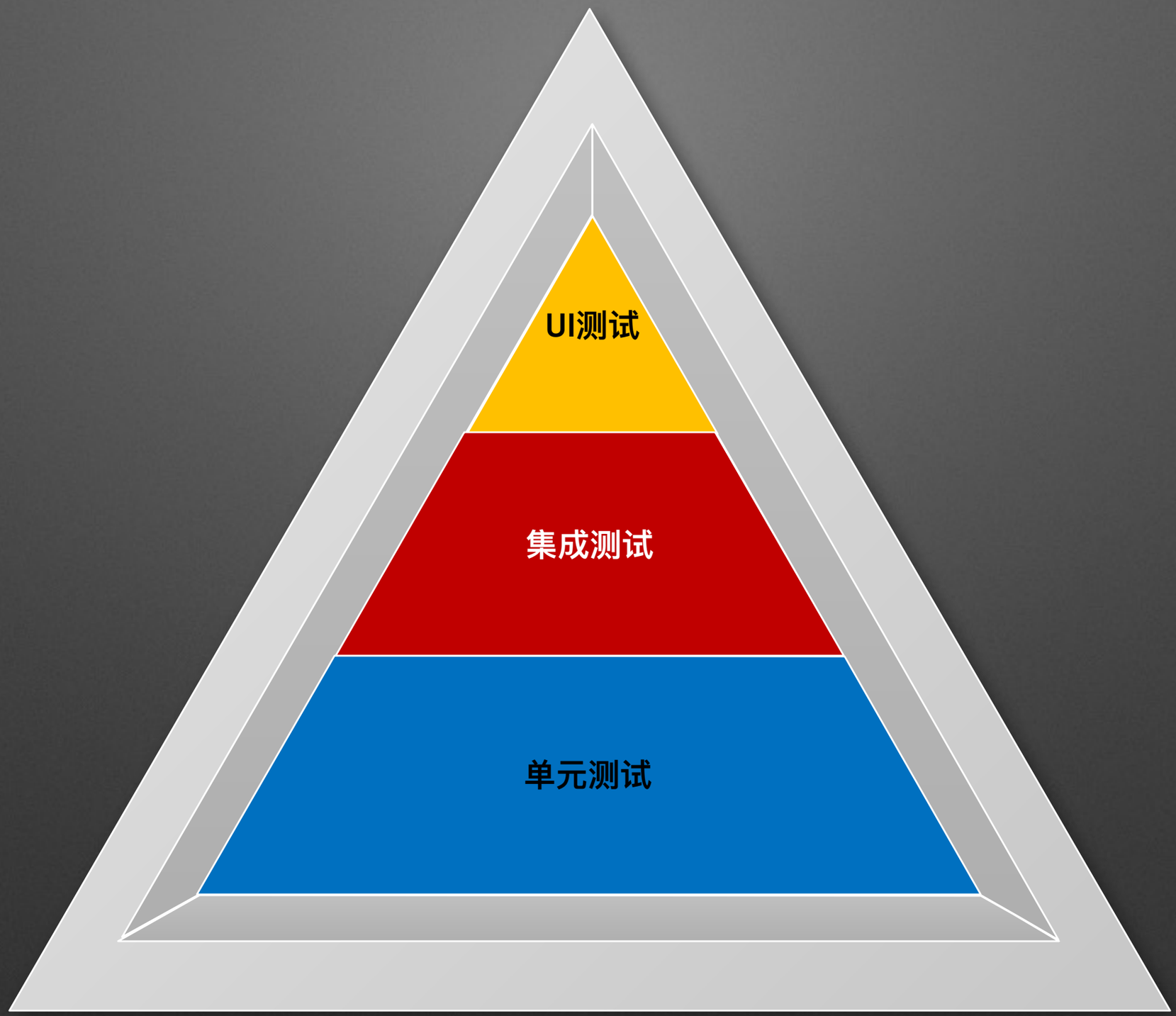
About Me

- Gemini 奶爸
- SegmentFault 移动端负责人

当我们讲测试的时候
我们在讲什么....

测试的颗粒度

- 单元测试（Unit Test）
- UI 测试
- 集成测试


UI测试
集成测试
单元测试

单元测试

- 是针对程序模块（软件设计的最小单位）来进行正确性检验的测试工作
- 检查输入与输出
- 是一切测试的基础
- 尽可能覆盖，但是不要期望覆盖率100%

集成测试

- 集成测试（也叫组装测试，联合测试）是单元测试的逻辑扩展。它最简单的形式是：把两个已经测试过的单元组合成一个组件，测试它们之间的接口。

UI测试

- 检查设计的还原度
- 检查组件之间的交互
- 检查业务流程
- 基于测试用例，做更完备的集成测试

Android Testing Framework

Espresso

```
@Test
public void greeterSaysHello() {
  onView(withId(R.id.name_field))
    .perform(typeText("Steve"));

  onView(withId(R.id.greet_button))
    .perform(click());

  onView(withText("Hello Steve!"))
    .check(matches(isDisplayed()));
}
```

UI测试思考的问题

- UI测试依据什么来写?
- UI测试的代码该写点什么?
- 需要注意的一些技巧是什么?

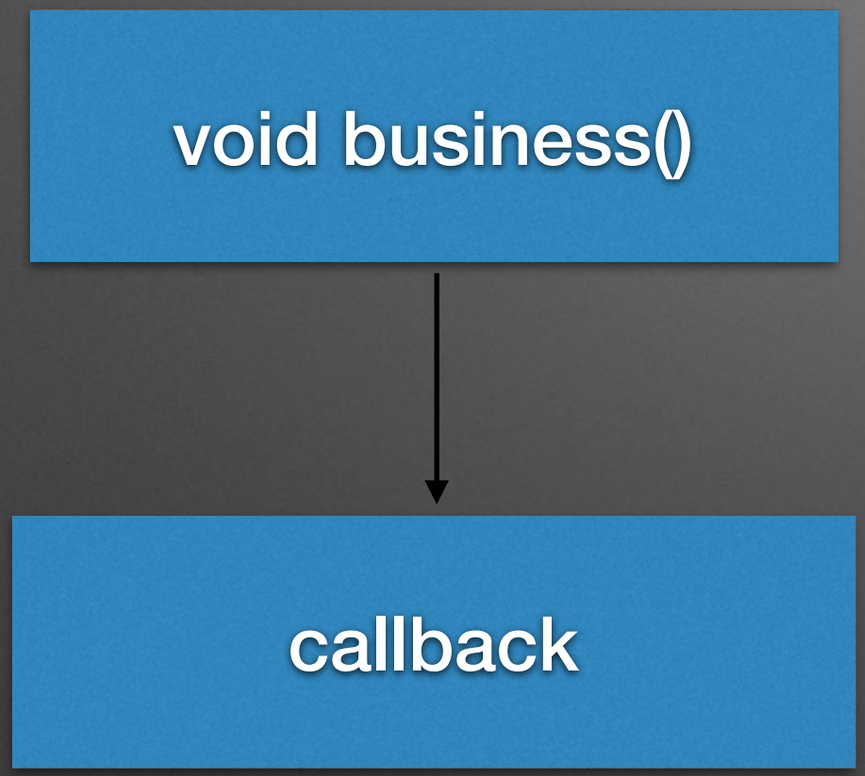
回归测试用例

B	C	D	E	F	G
优先级	模块	前置条件	用例	条件	期望结果
100	登录	无	输入正确的邮箱和密码		登录成功
100	登录		输入错误的邮箱或错误的密码		登录失败，提示用户名或密码错误
100	登录		不输入邮箱，输入密码		登录失败，提示邮箱没有输入
100	登录		不输入邮箱，不输入密码		登录失败，提示邮箱没有输入
100	登录		输入邮箱，不输入密码		登录失败，提示密码没有输入
100	注册		输入不存在的用户名，不存在的邮箱，和密码		注册成功
100	注册		输入已存在的用户名，不存在的邮箱，和密码		注册失败，提示用户名已存在
			输入已存在的用户名，已存在的邮箱，和密码		注册失败，提示用户名已存在
100	注册		输入不存在的用户名，已存在的邮箱，和密码		注册失败，提示邮箱已存在
100	注册		某一项不输入		布局优先级从上到下，前端提示出没有输入的那一项
100	用户详情	登录状态	点击修改用户信息按钮，进入修改个人信息页面	网络正常	应该能获取用户线上资料，展示在UI上

```java
@Test
public void testLoginSuccess() throws Exception {
    onView(withId(R.id.et_mail))
            .perform(clearText(), replaceText("geminiwen@aliyun.com"));
    onView(withId(R.id.et_password))
            .perform(clearText(), replaceText("123456"));
    onView(withId(R.id.btn_login))
            .perform(click());
    Thread.sleep(200);
    onView(withId(R.id.et_mail))
            .check(doesNotExist());
}

@Test
public void testLoginFailed() throws Exception {
    onView(withId(R.id.et_mail))
            .perform(clearText(), replaceText("geminiwen@aliyun.com"));
    onView(withId(R.id.et_password))
            .perform(clearText(), replaceText("99999"));
    onView(withId(R.id.btn_login))
            .perform(click());
    Thread.sleep(200);

    onView(withText("用户名或密码错误"))
            .inRoot(toast())
            .check(matches(isDisplayed()));
}
```

```java
@Test
public void testInputEmpty() throws Exception {
    onView(withId(R.id.et_mail))
            .perform(clearText());
    onView(withId(R.id.btn_login))
            .perform(click());
    onView(withText("邮箱不能为空"))
            .check(matches(isDisplayed()));

    Thread.sleep(1000);

    onView(withId(R.id.et_mail))
            .perform(clearText(), replaceText("geminiwen@aliyun.com"));
    onView(withId(R.id.et_password))
            .perform(clearText());
    onView(withId(R.id.btn_login))
            .perform(click());
    onView(withText("密码不能为空"))
            .check(matches(isDisplayed()));
}
```

业务异步?



IdleResource
By
Espresso

Mock

绝对不可以在测试的时候和服务器交互

- Mock Object
- Mock Http

Mockito

- 伪造model提供给UI渲染
- 判断伪造model的接口是否被调用

OkHttp MockWebServer

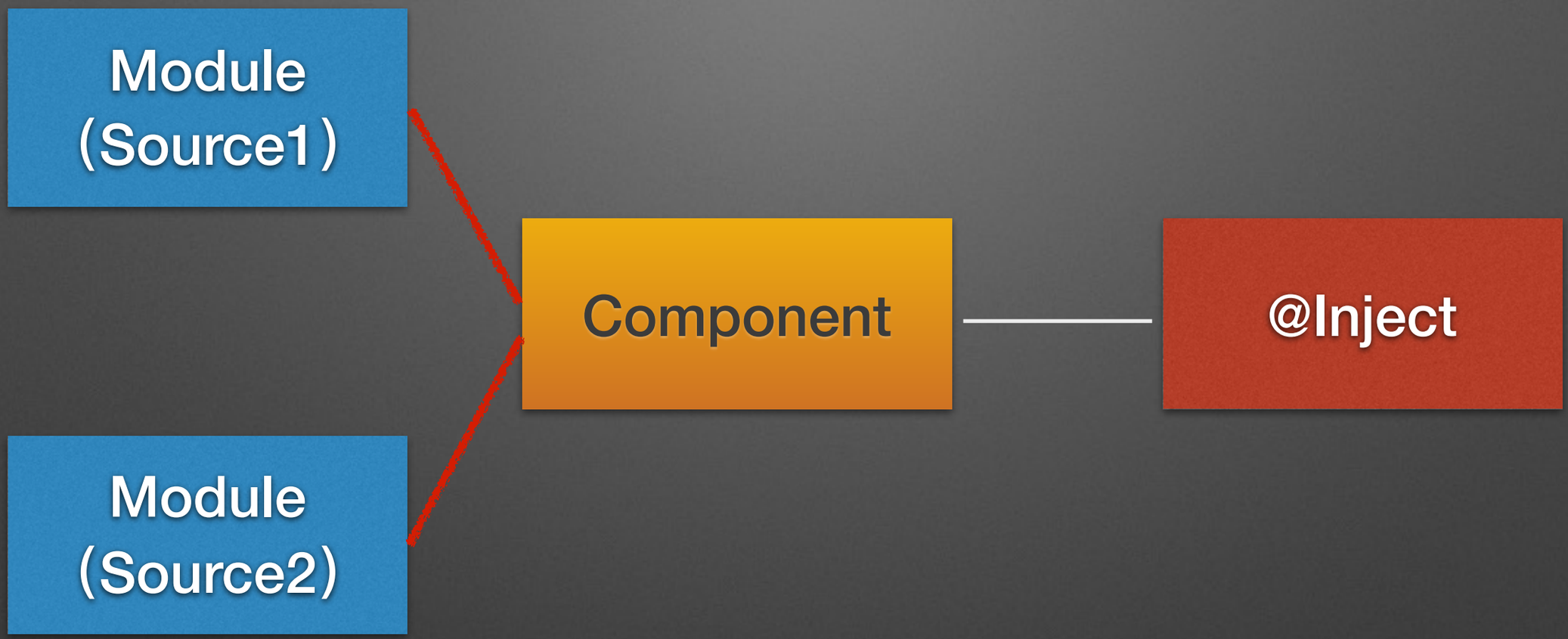
```
private static Dispatcher sServerDispatcher = new Dispatcher() {
    @Override
    public MockResponse dispatch(RecordedRequest request) throws InterruptedException {
        String url = request.getPath();
        MockResponse response = new MockResponse();
        if (url.equals("/user/login")) {
            Buffer buffer = request.getBody();
            String requestBody = buffer.readString(Charset.defaultCharset());
            if (requestBody.contains("password=123456")) {
                response.setBody("{\"status\":0,\"data\":{\"user\":{\"name\":\"Gemini\",\"mail\":\
                response.setResponseCode(200);
            } else {
                response.setBody("{\"status\":1,\"code\":\"01390902\",\"message\":\"\\u7528\\u623
                response.setResponseCode(200);
            }
        } else {
            response.setResponseCode(500);
        }
        return response;
    }
};
```

如何注入Mock Object?

- Dagger2 —— 静态依赖注入框架

Dagger Components

- Module
- Component
- @Inject


Module
(Source1)
Module
(Source2)
Component
@Inject

```java
public class App extends MultiDexApplication implements Application

    private final static String APP_NAME = "SegmentFault";
    private final static String MIPUSH_TAG = "SegmentFault_MiPush";

    @Inject AppComponent mAppComponent;
    @Inject UserStore mUserStore;
```

```java
App app = (App)context.getApplicationContext();
AppComponent appComponent = DaggerAppComponent.builder()
        .appModule(new TestAppModule(context, url))
        .build();
appComponent.inject(app);
```

```java
public class TestAppModule extends AppModule {

    private String mBaseUrl;

    public TestAppModule(Context context, String baseUrl) {
        super(context);
        this.mBaseUrl = baseUrl;
    }

    @Override
    public Service provideService(Context context, OkHttpClient okHttpClient) {
        return new Service(context, okHttpClient, mBaseUrl);
    }
}
```