近乎完美的安卓 Model 层架构

Piasy 许建林 @ YOLO

自我介绍

- Piasy，许建林 @ YOLO
- 目前专注安卓开发
- 必备开源库源码导读
- Advanced RxJava 系列翻译
- APP 架构/TDD 的思考与实践
- https://github.com/Piasy
- blog：http://blog.piasy.com/
- 公众号：Piasy
- 微博：@Piasy

注意事项

- 需要一定的安卓开发基础
- 分享本套架构对比传统/其他方案的优势
- 需要 *结合* 很多库，重在优势与结合，不在介绍
- 代码都很 *碎片*，重点在对比出优势
- 需要自行 Google，体验细节

目录

初心

- 简单可依赖
- Immutable value type
- 发出 RESTful API 请求
- JSON 数据格式
- SQLite 数据库
- Reactive

需求：维护用户 following 列表

- HTTP 缓存
- 从服务器获取
- SQLite 数据库本地缓存
- 定义 Model 类型
- 先展示本地缓存，再更新为网络数据
- 多页面同步更新
- Activity/Fragment 传参
- 单元测试，集成测试

需求：维护用户 following 列表

- ***HTTP 缓存***
- 从服务器获取
- SQLite 数据库本地缓存
- 定义 Model 类型
- 先展示本地缓存，再更新为网络数据
- 多页面同步更新
- Activity/Fragment 传参
- 单元测试，集成测试

HTTP 缓存

- 实现存储的读写，磁盘缓存，缓存淘汰策略？
- 解析响应 header？
- 实现 HTTP 缓存逻辑？
- Socket？HttpURLConnection？
- OkHttp！
- http://blog.piasy.com/2016/07/11/Understand-OkHttp/

需求：维护用户 following 列表

- HTTP 缓存
- *从服务器获取*
- SQLite 数据库本地缓存
- 定义 Model 类型
- 先展示本地缓存，再更新为网络数据
- 多页面同步更新
- Activity/Fragment 传参
- 单元测试，集成测试

从服务器获取

- 利用 OkHttp 发起 HTTP 请求
- 构造请求?
- 数据转换?
- RxJava?

```
final Gson gson = new GsonBuilder().create();
Observable<List<GithubUser>> following = Observable.create(subscriber -> {
    OkHttpClient client = new OkHttpClient();
    Request request = new Request.Builder().url("https://api.github.com/Piasy/following")
            .get()
            .build();
    client.newCall(request).enqueue(new Callback() {
        @Override
        public void onFailure(Call call, IOException e) {
            subscriber.onError(e);
        }

        @Override
        public void onResponse(Call call, Response response) throws IOException {
            String body = response.body().string();
            List<GithubUser> users = gson.fromJson(body, new TypeToken<List<GithubUser>>() {
            }.getType());
            subscriber.onNext(users);
            subscriber.onCompleted();
        }
    });
});

following.subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(followings -> {
            // 拿到了 API 返回数据(可能是 HTTP 缓存)
        }, err -> {
            // 错误处理
        });
```

```
Retrofit retrofit = new Retrofit.Builder().baseUrl("https://api.github.com/")
        .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
        .addConverterFactory(GsonConverterFactory.create())
        .build();
GithubApi api = retrofit.create(GithubApi.class);
Observable<List<GithubUser>> following = api.following("Piasy");

following.subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(followings -> {
            // 拿到了 API 返回数据(可能是 HTTP 缓存)
        }, err -> {
            // 错误处理
        });
```

Retrofit

- 基于注解，减少 boilerplate code
- 类型安全
- 高度可扩展：converter，call adapter 随意配置
- http://blog.piasy.com/2016/06/25/Understand-Retrofit/

需求：维护用户 following 列表

- HTTP 缓存
- 从服务器获取
- ***SQLite 数据库本地缓存***
- 定义 Model 类型
- 先展示本地缓存，再更新为网络数据
- 多页面同步更新
- Activity/Fragment 传参
- 单元测试，集成测试

SQLite 数据库本地缓存

- 建表
- 增删改查
- transaction
- io 线程执行

```
DbOpenHelper helper = new DbOpenHelper(context);
SQLiteDatabase database = helper.getWritableDatabase();

database.beginTransaction();
try {
    for (int i = 0, size = users.size(); i < size; i++) {
        GithubUser githubUser = users.get(i);
        ContentValues contentValues = new ContentValues();
        contentValues.put(GithubUser.ID, githubUser.id());
        contentValues.put(GithubUser.LOGIN, githubUser.login());
        if (githubUser.created_at() != null) {
            contentValues.put(GithubUser.CREATED_AT,
                    formatter.format(githubUser.created_at()));
        }
        database.insert(GithubUser.TABLE_NAME, null, contentValues);
    }
    database.setTransactionSuccessful();
} finally {
    database.endTransaction();
}
```

```
BriteDatabase briteDb = SqlBrite.create().wrapDatabaseHelper(
    new DbOpenHelper(context), Schedulers.io());

BriteDatabase.Transaction transaction = briteDb.newTransaction();
try {
    for (int i = 0, size = users.size(); i < size; i++) {
        briteDb.insert(GithubUser.TABLE_NAME,
                GithubUser.FACTORY.marshal(users.get(i)).asContentValues(),
                SQLiteDatabase.CONFLICT_REPLACE);
    }
    transaction.markSuccessful();
} finally {
    transaction.end();
}
```

SqlBrite

- SQLiteOpenHelper 的轻量封装，提供更新提醒（RxJava Observable）
- 基础 SQLite 访问 API
  - insert
  - delete
  - update
  - query
  - execute
  - transaction
- 结合 SqlDelight，实现类型安全
- https://github.com/square/sqlbrite

需求：维护用户 following 列表

- HTTP 缓存
- 从服务器获取
- SQLite 数据库本地缓存
- ***定义 Model 类型***
- 先展示本地缓存，再更新为网络数据
- 多页面同步更新
- Activity/Fragment 传参
- 单元测试，集成测试

定义 Model 类型

- immutable?
- builder?
- JSON 转换?
- Server <=> Model <=> SQLite?

AutoValue!

```java
@AutoValue
public abstract class Payment {

  public static Payment create(
      long id, long amount, Currency currency, String note) {
    return new AutoValue_Payment(id, amount, currency, note);
  }

  public abstract long id();
  public abstract long amount();
  public abstract Currency currency();
  public abstract String note();
}
```

```java
final class AutoValue_Payment extends Payment {
  private final long id;
  private final long amount;
  private final Currency currency;
  private final String note;

  AutoValue_Payment(long id, long amount, Currency currency, String note) {
    this.id = id;
    this.amount = amount;
    this.currency = currency;
    this.note = note;
  }

  @Override public long id() {
    return id;
  }
  @Override public long amount() {
    return amount;
  }
  @Override public Currency currency() {
    return currency;
  }
  @Override public String note() {
    return note;
  }

  @Override public String toString() {
    return "Payment{" +
        "id=" + id +
        ", amount=" + amount +
        ", currency=" + currency +
        ", note='" + note + '\'' +
        '}';
  }

  @Override public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Payment)) return false;
    Payment other = (Payment) o;
    return id == other.id
        && amount == other.amount
        && currency.equals(other.currency)
        && note.equals(other.note);
  }

  @Override public int hashCode() {
    int result = (int) (id ^ (id >>> 32));
    result = 31 * result + (int) (amount ^ (amount >>> 32));
    result = 31 * result + currency.hashCode();
    result = 31 * result + note.hashCode();
    return result;
  }
}
```

JSON 转换

- 使用 Gson
- Gson 默认使用反射完成转换
- 支持自定义转换逻辑
- 通过测试，`toJson` 使用反射耗时增加：192.2% ~ 298%
- http://blog.piasy.com/2016/05/06/Perfect-Android-Model-Layer/#autogson-

```java
public static final class GsonTypeAdapter extends TypeAdapter<GithubUser> {
    private final TypeAdapter<Long> idAdapter;
    private final TypeAdapter<String> loginAdapter;
    private final TypeAdapter<ZonedDateTime> created_atAdapter;
    public GsonTypeAdapter(Gson gson) {
      this.idAdapter = gson.getAdapter(Long.class);
      this.loginAdapter = gson.getAdapter(String.class);
      this.created_atAdapter = gson.getAdapter(ZonedDateTime.class);
    }
    @Override
    public void write(JsonWriter jsonWriter, GithubUser object) throws IOException {
      jsonWriter.beginObject();
      if (object.id() != null) {
        jsonWriter.name("id");
        idAdapter.write(jsonWriter, object.id());
      }
      jsonWriter.name("login");
      loginAdapter.write(jsonWriter, object.login());
      if (object.created_at() != null) {
        jsonWriter.name("created_at");
        created_atAdapter.write(jsonWriter, object.created_at());
      }
      jsonWriter.endObject();
    }
    @Override
    public GithubUser read(JsonReader jsonReader) throws IOException {
      jsonReader.beginObject();
      Long id = null;
      String login = null;
      ZonedDateTime created_at = null;
      while (jsonReader.hasNext()) {
        String _name = jsonReader.nextName();
        if (jsonReader.peek() == JsonToken.NULL) {
          jsonReader.skipValue();
          continue;
        }
        switch (_name) {
          case "id": {
            id = idAdapter.read(jsonReader);
            break;
          }
          case "login": {
            login = loginAdapter.read(jsonReader);
            break;
          }
          case "created_at": {
            created_at = created_atAdapter.read(jsonReader);
            break;
          }
          default: {
            jsonReader.skipValue();
          }
        }
      }
      jsonReader.endObject();
      return new AutoValue_GithubUser(id, login, created_at);
    }
  }
```

```java
@AutoValue
public abstract class GithubUser {

    // 方法定义

    public static TypeAdapter<GithubUser> typeAdapter(final Gson gson) {
        return new AutoValue_GithubUser.GsonTypeAdapter(gson);
    }

}
```

AutoValue 及其扩展

- apt 生成代码，稳定可靠
- https://github.com/google/auto/blob/master/value/
- 支持非反射 Gson：https://github.com/rharter/auto-value-gson
- http://ryanharter.com/blog/2016/05/16/autovalue-extensions/

Server <=> Model <=> SQLite

- Server <=> Model：JSON，Gson，auto-value-gson
- Model => SQLite：ContentValue
- Model <= SQLite：Cursor

```
ContentValues contentValues = new ContentValues();
contentValues.put(GithubUser.ID, githubUser.id());
contentValues.put(GithubUser.LOGIN, githubUser.login());
if (githubUser.created_at() != null) {
    contentValues.put(GithubUser.CREATED_AT, formatter.format(githubUser.created_at()));
}
database.insert(GithubUser.TABLE_NAME, null, contentValues);

Cursor cursor = database.query(GithubUser.TABLE_NAME,
        new String[] { GithubUser.ID, GithubUser.LOGIN, GithubUser.CREATED_AT }, "id = ?",
        new String[] { "1" }, null, null, null);
GithubUser read =
        new AutoValue_GithubUser(cursor.getLong(cursor.getColumnIndex(GithubUser.ID)),
                cursor.getString(cursor.getColumnIndex(GithubUser.LOGIN)), formatter.parse(
                cursor.getString(cursor.getColumnIndex(GithubUser.CREATED_AT)),
                ZonedDateTime.FROM));
cursor.close();
```

```
briteDb.insert(GithubUser.TABLE_NAME,
        GithubUser.FACTORY.marshal(githubUser).asContentValues(),
        SQLiteDatabase.CONFLICT_REPLACE);

Cursor cursor = briteDb.query(GithubUser.TABLE_NAME, GithubUser.GET_BY_ID, "1");
GithubUser read = GithubUser.MAPPER.map(cursor);
cursor.close();
```

SqlDelight

- 根据建表 SQL 语句生成 model interface
- 兼容 AutoValue
- 生成 DB 读写类型安全转换代码
- 支持自定义类型
- https://github.com/square/sqldelight

需求：维护用户 following 列表

- HTTP 缓存
- 从服务器获取
- SQLite 数据库本地缓存
- 定义 Model 类型
- *先展示本地缓存，再更新为网络数据*
- 多页面同步更新
- Activity/Fragment 传参
- 单元测试，集成测试

先展示本地缓存，再更新为网络数据

- 两个数据来源，如何结合？
- 异步？
- 错误处理？
- 如果缓存命中，就不请求网络？

RxJava

- 强大的事件流处理能力：操作符
- 简单的异步 API：`subscribeOn`，`observeOn`，`scheduler`
- 集中错误处理
- http://blog.csdn.net/theone10211024/article/details/50435325

```
Observable.concat(local, network)
        .first()
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(followings -> {
            // 缓存命中就是缓存，且不会发起网络请求，否则是网络数据
        }, err -> {
            // 集中错误处理
        });
```

需求：维护用户 following 列表

- HTTP 缓存
- 从服务器获取
- SQLite 数据库本地缓存
- 定义 Model 类型
- 先展示本地缓存，再更新为网络数据
- *多页面同步更新*
- Activity/Fragment 传参
- 单元测试，集成测试

多页面同步更新

- 本质：数据源更新，要通知所有感兴趣的使用者（Observer）
- Java 内置的 Observer API
- EventBus
- RxJava Observable！（涉及 DB 时 SqlBrite 已有支持）

需求：维护用户 following 列表

- HTTP 缓存
- 从服务器获取
- SQLite 数据库本地缓存
- 定义 Model 类型
- 先展示本地缓存，再更新为网络数据
- 多页面同步更新
- ***Activity/Fragment 传参***
- 单元测试，集成测试

Activity/Fragment 传参

- 构造函数？Activity 由 framework 利用反射构造！
- setter？Activity/Fragment 可能会被系统销毁然后恢复！
- Activity 设置在 `Intent` 中，`getIntent()` 读取
- Fragment 利用 `setArguments()` 设置，`getArguments()` 读取
- `Bundle`，`Parcelable`
- auto-value-parcel：https://github.com/rharter/auto-value-parcel
- IntentBuilder，FragmentArgs，AutoBundle
- AutoBundle：https://github.com/yatatsu/AutoBundle

```java
@NonNull
private static Intent newInstance(Context context, Group group, int type) {
    Intent intent = new Intent(context, RoomActivity.class);
    intent.putExtra(INTENT_GROUP_KEY, gson.toJson(group));
    intent.putExtra(INTENT_ROOM_TYPE_KEY, type);
    return intent;
}

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

    Intent intent = getIntent();
    mRoomType = intent.getIntExtra(INTENT_ROOM_TYPE_KEY, ROOM_TYPE_CHAT);
    String groupStr = getIntent().getStringExtra(INTENT_GROUP_KEY);
    Group group = mGson.fromJson(groupStr, Group.class);
    // ...
}
```

```java
public static void setArgs(Bundle args, long roomId, long groupId,
        long bcid, String fromUserAvatar, String fromUsername,
        long fromUid, String toUserAvatar, String toUsername, long toUid) {
    args.putLong(ARGS_KEY_ROOM_ID, roomId);
    args.putLong(ARGS_KEY_GROUP_ID, groupId);
    args.putLong(ARGS_KEY_BCID, bcid);
    args.putString(ARGS_KEY_FROM_USER_AVATAR, fromUserAvatar);
    args.putString(ARGS_KEY_FROM_USER_NAME, fromUsername);
    args.putLong(ARGS_KEY_FROM_UID, fromUid);
    args.putString(ARGS_KEY_TO_USER_AVATAR, toUserAvatar);
    args.putString(ARGS_KEY_TO_USER_NAME, toUsername);
    args.putLong(ARGS_KEY_TO_UID, toUid);
}

private void getArgs() {
    mBcid = getArguments().getLong(ARGS_KEY_BCID);
    mRoomId = getArguments().getLong(ARGS_KEY_ROOM_ID);
    mGroupId = getArguments().getLong(ARGS_KEY_GROUP_ID);
    mFromUserAvatar = getArguments().getString(ARGS_KEY_FROM_USER_AVATAR);
    mFromUsername = getArguments().getString(ARGS_KEY_FROM_USER_NAME);
    mFromUid = getArguments().getLong(ARGS_KEY_FROM_UID);
    mToUserAvatar = getArguments().getString(ARGS_KEY_TO_USER_AVATAR);
    mToUsername = getArguments().getString(ARGS_KEY_TO_USER_NAME);
    mToUid = getArguments().getLong(ARGS_KEY_TO_UID);
}
```

```java
public static final Parcelable.Creator<AutoValue_GithubUser> CREATOR
    = new Parcelable.Creator<AutoValue_GithubUser>() {
  @Override
  public AutoValue_GithubUser createFromParcel(Parcel in) {
    return new AutoValue_GithubUser(
        in.readInt() == 0 ? in.readLong() : null,
        in.readString(),
        in.readInt() == 0 ? (ZonedDateTime) in.readSerializable() : null
    );
  }
  @Override
  public AutoValue_GithubUser[] newArray(int size) {
    return new AutoValue_GithubUser[size];
  }
};

@Override
public void writeToParcel(Parcel dest, int flags) {
  if (id() == null) {
    dest.writeInt(1);
  } else {
    dest.writeInt(0);
    dest.writeLong(id());
  }
  dest.writeString(login());
  if (created_at() == null) {
    dest.writeInt(1);
  } else {
    dest.writeInt(0);
    dest.writeSerializable(created_at());
  }
}

@Override
public int describeContents() {
  return 0;
}
```

```java
@AutoValue
public abstract class GithubUser implements Parcelable {
  // 其他成员方法
}
```

```java
public class ProfileActivity extends AppCompatActivity {

    @AutoBundleField
    GithubUser mUser;

    @Override
    protected void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        AutoBundle.bind(this);
        // 使用 mUser
    }
}

startActivity(ProfileActivityAutoBundle.createIntentBuilder(user).build(getContext()));
```

需求：维护用户 following 列表

- HTTP 缓存
- 从服务器获取
- SQLite 数据库本地缓存
- 定义 Model 类型
- 先展示本地缓存，再更新为网络数据
- 多页面同步更新
- Activity/Fragment 传参
- *单元测试，集成测试*

单元测试

- 把对 framework 的依赖都通过 delegate 接口隔离，实现解耦
- 接口可以随意 mock，JUnit 测试即可
- 类似还有 MVP 模式中 V 引入接口
- http://www.philosophicalhacker.com/2015/05/01/how-to-make-our-android-apps-unit-testable-pt-1/

UnMock Plugin

单元测试：`*** not mocked`?

```
apply plugin: 'de.mobilej.unmock'

unMock {
    downloadFrom 'https://oss.sonatype.org/...'

    keep "android.os.Looper"
    keep "android.content.ContentValues"
    keepStartingWith "android.util."
}
```

https://github.com/bjoernQ/unmock-plugin

集成测试

- 测试中不要发起实际网络请求！
- mock 哪一层？
- 单元测试，mock 的越多越好
- 集成测试，mock 的越少越好

```
new Retrofit.Builder()
    .baseUrl("http://localhost:" + PORT + "/")
    .build();

mMockWebServer = new MockWebServer();
mMockWebServer.start(PORT);
mMockWebServer.setDispatcher(new Dispatcher() {
    @Override
    public MockResponse dispatch(final RecordedRequest request)
            throws InterruptedException {
        final String path = request.getPath();
        if (path.startsWith("/search/users?")) {
            return new MockResponse().setBody(someString);
        } else {
            return new MockResponse();
        }
    }
});
```

```
new Retrofit.Builder()
    .baseUrl(RESTMockServer.getUrl())
    .build();

RESTMockServer.whenGET(pathStartsWith("/search/users?"))
        .thenReturnString(200, someString);
```

- RequestMatcher
- RequestsVerifier
- https://github.com/andrzejchm/RESTMock

Config Injection

- 把业务相关的配置注入到业务无关的 model 架构中
- Retrofit，EventBus，SqlBriteDatabase 等对象的创建都在 base module 中
- debug，base url 等参数却和具体业务相关，甚至和 build 相关
- 依赖倒置，base 不能依赖 model/app
- injection! 类似有些框架中的配置文件

ZonedDateTime

- 用什么表示时间？
- 获得指定日期时间的对象？获得加/减指定时间段之后的对象？计算两个时间的（日期）时间差？
- `Date` + `Calendar`？ `ZonedDateTime` ！
- Java 8 引入，解决老的 Date API 的问题
- ThreeTenBP & ThreeTenABP
- https://github.com/JakeWharton/ThreeTenABP

Date API 的问题

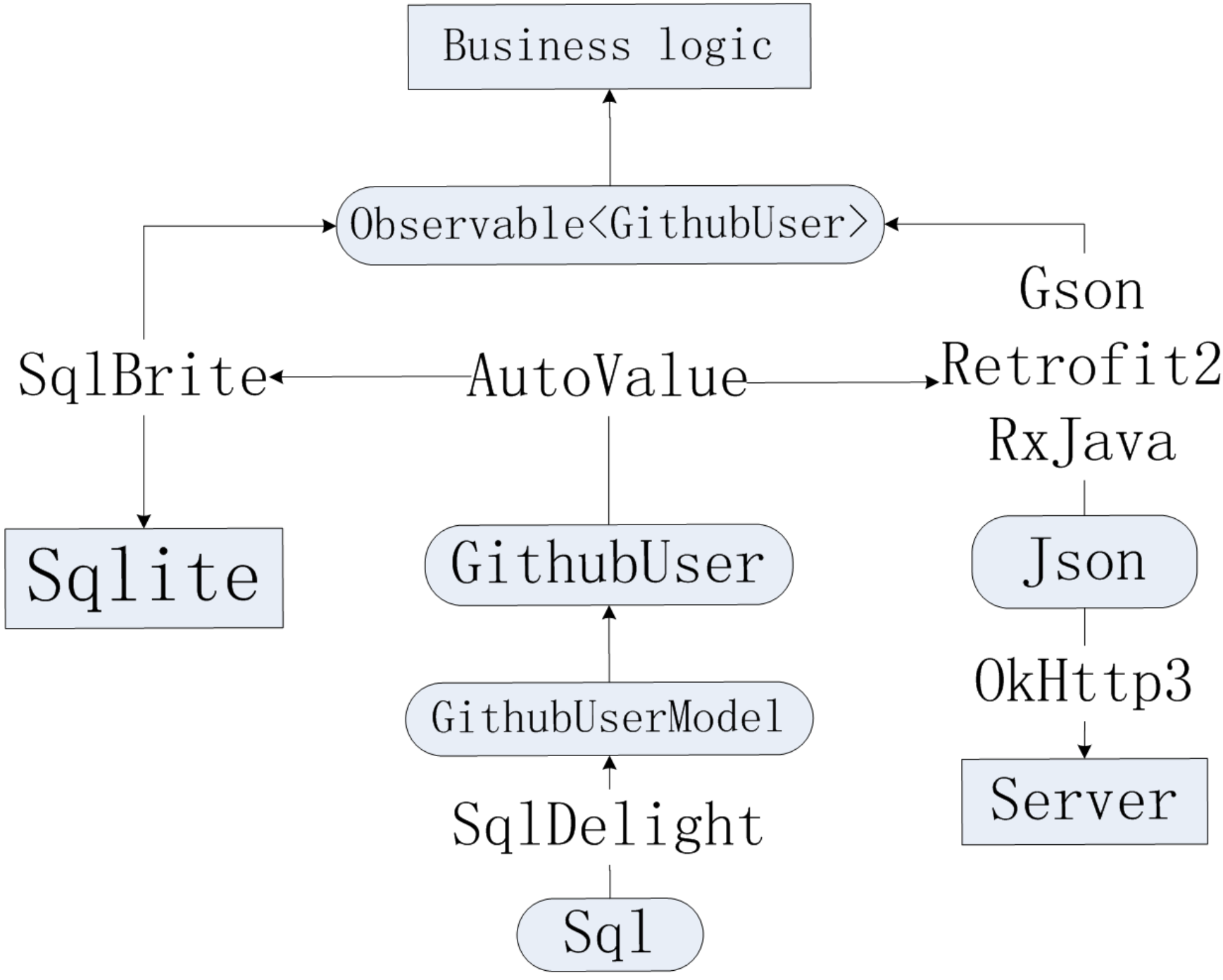
- mutable，非线程安全
- 1 月的值是 0，转化为字符串时却显示 1
- 没有和时区关联
- 日期运算非常不方便
- ...
- http://stackoverflow.com/a/1969651/3077508

```java
Calendar calendar = Calendar.getInstance();
calendar.set(2016, 6, 10, 18, 29, 0);
Date date = calendar.getTime();

Date after = new Date(date.getTime() + 2 * 24 * 60 * 60 * 1000
    + 3 * 60 * 60 * 1000);
long diff = after.getTime() - date.getTime();
long diffHour = diff / (60 * 60 * 1000);

System.out.println(format.format(date) + ", "
    + format.format(after) + ", " + diffHour);
// 2016/07/10 18:29:00, 2016/07/12 21:29:00, 51
```

```
ZonedDateTime time = ZonedDateTime.of(2016, 6, 10, 18, 29, 0, 0,
        ZoneId.systemDefault());
ZonedDateTime after = time.plusDays(2).plusHours(3);
Duration duration = Duration.between(time, after);
System.out.println(time.format(formatter) + ", " + after.format(formatter)
        + ", " + duration.toHours());
// 2016/06/10 18:29:00, 2016/06/12 21:29:00, 51
```

总结

- OkHttp + Retrofit：调用 RESTful API
- SqlBrite：数据库访问 && Rx
- SqlDelight：SQL -> model interface，DB 安全读写
- AutoValue(ext)：model interface -> model class
- Gson，auto-value-gson：简洁高效 JSON 转换
- RxJava：异步 && 事件流
- delegate 接口层隔离安卓系统
- Config injection
- ZonedDateTime：安全，简单


Business logic
Observable<GithubUser>
SqlBrite
AutoValue
Gson
Retrofit2
RxJava
Sqlite
GithubUser
Json
GithubUserModel
OkHttp3
SqlDelight
Server
Sql

不足

- 东西较多，有一定复杂度，但我们想要的很多
- 重度依赖 apt 代码生成

广告一则

- YOLO：朋友间的视频直播 APP
- 上线1年多，A轮融资
- 招聘安卓、iOS、流媒体、PHP工程师
- 简历发至 xujianlin@yoloyolo.tv

谢谢！

Q&A

『完美的安卓model层架构』分享Q&A

该二维码7天内(8月7日前)有效，重新进入将更新