armsMVP demo 文档

- 主要修改了arms框架的dagger注入部分，改为了dagger-android的方式
  - 这样就不用每次都调用如下方法注入依赖

```
DaggerMainComponent.builder()
     .appComponent(appComponent)
     .mainModule(new MainModule(this))
     .build().inject(this);
```

  而是在activity的oncreate

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
  AndroidInjection.inject(this);
  super.onCreate(savedInstanceState);
}
```

  或者fragment的 onattach 注入（注入已经放到arms库基类里面）

```
@Override
public void onAttach(Context context) {
    AndroidSupportInjection.inject(this);
    super.onAttach(context);
}
```

  其他三大组件如果需要依赖于app同理

- **此demo有一个MainActivity，其下有dayli weekly monthly三个fragment，weeklyFragment里请求网络获取了一个七天的城市天气。**
- dagger-android构造了组件之间的依赖关系

  appComponent -> activityComponent -> fragmentComponent

  demo则是 AppDelegate -> MainActivity -> 三个tab fragment

  AppDelegate继承于arms里的BaseAppDelegate，在它的实现方法里可以拿到arms提供给我们的BaseComponent， 然后我们可以创建自己的Appcomponent并声明它为BaseComponent的子Component，这样就可以使用arms提供给我们的依赖

```
@Override
  public void bindSubComponent(BaseComponent baseComponent) {
      DaggerAppComponent.builder()
              .baseComponent(baseComponent)
              .build().inject(this);
  }
```

- AppComponent管理ActivityComponent

  创建一个ActivityModule用于注册所有ActivityComponent，@ContributesAndroidInjector会帮助生成一个隐式的component， 然后这里相当于一个MainActivityComponent作为AppComponent的子组件，它又拥有一个MainFragmentModule，用来管理它的三个 fragmentComponent，类似ActivityModule

```
@Module
public abstract class ActivityModule {

    /**
     * {@link ContributesAndroidInjector} 相当于创建了了一个用于注入MainActivity的AppComponent的子Component,
     * modules：此component所依赖的modules。
     * 提供activity依赖  所生成代码：build/下： ActivityModule_ForecastListActivityInjector
     * @return
     */
    @ContributesAndroidInjector(modules = {
            MainModule.class,
            MainFragmentModule.class,
    })
    @ActivityScope
    abstract MainActivity provideForecastListActivity();
}
```

- **创建一个新的Activity步骤**

- 创建Contract (View 和 Presenter的业务接口)

```
public interface MainContract {
  interface View extends IView {

  }

  interface Presenter {

  }
}
```

- 创建Model的业务接口，也可以不创建接口直接创建实现类
- 创建Model实现类，如果model需要RepositoryManager(提供了retrofit和rxCache实例)就继承BaseModel，否则实现IModel

```
@ActivityScope
public class MainModel extends BaseModel implements IMainModel {
    @Inject
    public MainModel() {

    }
}
```

  - 创建Presenter实现类，必须继承基类，Model泛型要为model实现类类型，View泛型要为抽象类（避免在presenter里面直接操作context）

```
@ActivityScope
public class MainPresenter extends BasePresenter<MainModel, MainContract.View> implements MainContract.Presenter {
    @Inject
    public MainPresenter() {

    }
}
```

  - 创建Activity，Presenter泛型要为实现类类型，如果需要fragment则需要实现HasSupportFragmentInjector，添加如下固定写法， 并且创建此act的fragmentModule

```
public class MainActivity extends BaseActivity<MainPresenter> implements MainContract.View, HasSupportFragmentInjector {
    @Inject
    DispatchingAndroidInjector<Fragment> fragmentInjector;

    @Override
    public AndroidInjector<Fragment> supportFragmentInjector() {
        return fragmentInjector;
    }
}
```

    - 如果activity不需要presenter，可以不提供泛型，同时在appModule里面添加一个空实例, 不需要model同理

```
@Provides
@Nullable
public static IPresenter providerPresenter(){
    return null;
}
```

  - 创建MainActivityModule，提供View给Presenter。还可以添加一些其他的实例依赖

```
@Module
public abstract class MainModule {

    @ActivityScope
    @Binds
    abstract MainContract.View provideMainView(MainActivity activity);
}
```

  - 在ActivityModule注册此activity

```
  @ContributesAndroidInjector(modules = {
       MainModule.class,
       MainFragmentModule.class,
  })
  @ActivityScope
  abstract MainActivity provideForecastListActivity();
```