# 项目中会用到的技术

### 5 . 插件化相关技术，热修补技术是怎样实现的，和插件化有什么区别

相同点:

都使用ClassLoader来实现的加载的新的功能类，都可以使用PathClassLoader与DexClassLoader

不同点：

热修复因为是为了修复Bug的，所以要将新的同名类替代同名的Bug类，要抢先加载新的类而不是Bug类，所以多做两件事：在原先的app打包的时候，阻止相关类去打上CLASS\_ISPREVERIFIED标志，还有在热修复时动态改变BaseDexClassLoader对象间接引用的dexElements，这样才能抢先代替Bug类，完成系统不加载旧的Bug类.

而插件化只是增肌新的功能类或者是资源文件，所以不涉及抢先加载旧的类这样的使命，就避过了阻止相关类去打上CLASS\_ISPREVERIFIED标志和还有在热修复时动态改变BaseDexClassLoader对象间接引用的dexElements.

所以插件化比热修复简单，热修复是在插件化的基础上在进行替旧的Bug类

##### 360截图20180402214254011

##### 一. 类的加载过程，Person person = new Person();为例进行说明。

1).因为new用到了Person.class，所以会先找到Person.class文件，并加载到内存中;

2).执行该类中的static代码块，如果有的话，给Person.class类进行初始化;

3).在堆内存中开辟空间分配内存地址;

4).在堆内存中建立对象的特有属性，并进行默认初始化;

5).对属性进行显示初始化;

6).对对象进行构造代码块初始化;

7).对对象进行与之对应的构造函数进行初始化;

8).将内存地址付给栈内存中的p变量

##### 双缓冲技术在绘图中的使用

文章来源：http://www.jianshu.com/u/4d74dd2268af

简单介绍：onDraw方法执行完成之后，才会把数据交给GPU去处理展示。这就是android绘图当中的第一道缓冲，即显示缓冲区。

而所谓的双缓冲，在android绘图中其实就是再创建一个Canvas和对应的Bitmap，然后在onDraw方法里默认的Canvas通过drawBitmap画刚才new的那个bitmap从而实现双缓冲。

##### Retrofit相关

生命周期抽取，通过代理模式：http://andydev.me/2017/07/03/retrofit-lifecycle-management/

##### SDK开发需要考虑的方向

借鉴文章：<http://blog.csdn.net/u010019468/article/details/75425742>

##### 安卓开发过程中的神器App

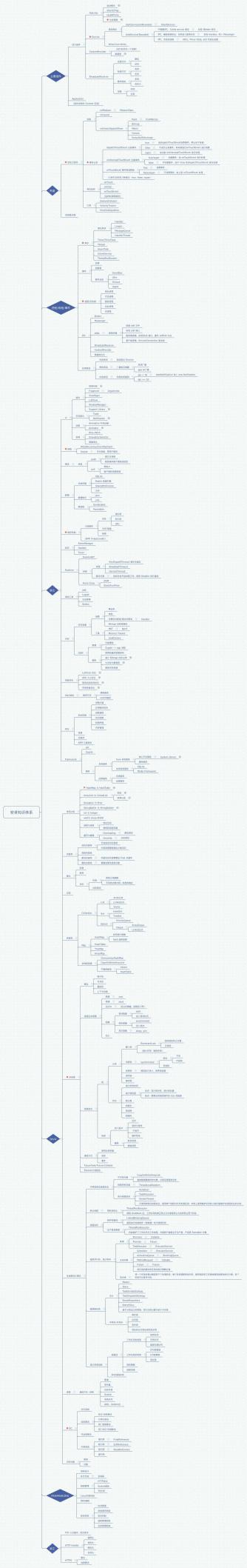
借鉴文章：<https://www.jianshu.com/p/1bdb98c3bf4c>

##### 事件总线

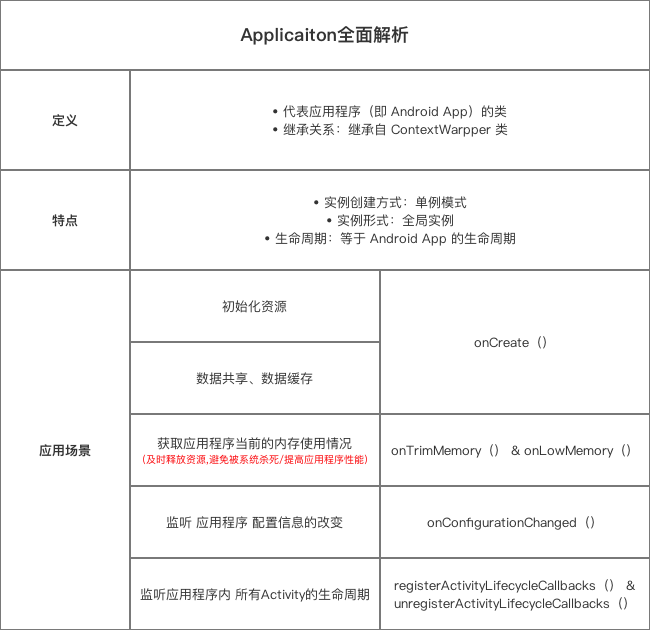
定义：针对事件提供统一订阅，发布以达到组件间通信的解决方案。

原理：观察者模式。发送者把消息发送到事件总线，然后事件总线在接收者中查找哪些方法注册了这个事件，如果某个方法注册了这个事件，就触发该方法。

##### 安卓知识导图

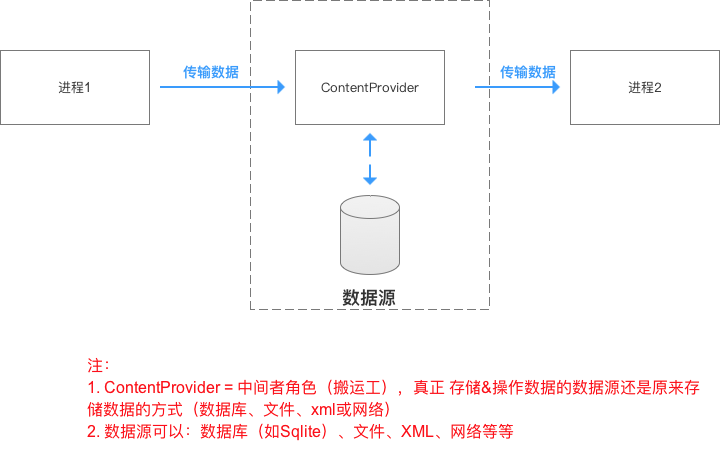


##### Application介绍



##### ContentProvider篇

作用：进程间，进行数据交互&共享，即跨进程通信



原理：Android的Binder机制

##### 时光轴的绘制



借鉴文章：<https://blog.csdn.net/carson_ho/article/details/75005994>

实现方式：通过RecyclerView的ItemDecoration，在偏移里设置装饰

public class TestDividerItemDecoration extends RecyclerView.ItemDecoration {

// 方法1：getItemOffsets（）

// 作用：设置ItemView的内嵌偏移长度（inset）

@Override

public void getItemOffsets(Rect outRect, View view, RecyclerView parent, RecyclerView.State state) {}

// 方法2：onDraw（）

// 作用：在子视图上设置绘制范围，并绘制内容，先于item的onDraw绘制

// 类似平时自定义View时写onDraw()一样

// 绘制图层在ItemView以下，所以如果绘制区域与ItemView区域相重叠，会被遮挡

@Override

public void onDraw(Canvas c, RecyclerView parent, RecyclerView.State state) {}

// 方法3：onDrawOver（）

// 作用：同样是绘制内容，但与onDraw（）的区别是：绘制在图层的最上层

@Override

public void onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state) {

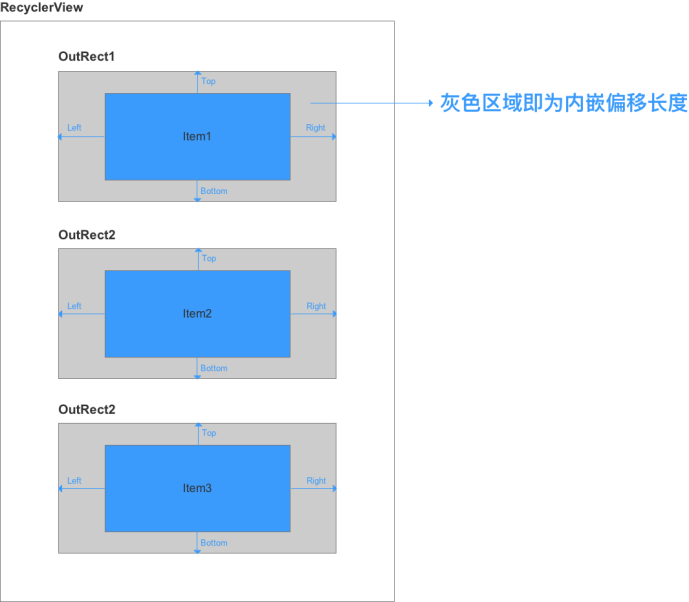
}

##### getItemOffsets

作用：设置ItemView的内嵌偏移长度（inset）

如图，其实RecyclerView 中的 ItemView 外面会包裹着一个矩形（outRect）

内嵌偏移长度是指：该矩形（outRect）与 ItemView的间隔



# Java设计模式篇

### 设计模式初识

**1.1 什么是设计模式**

设计模式(Design Pattern)是一套被反复使用、多数人知晓、经过分类编目的优秀代码设计经验的总结。

**1.2 设计模式的作用**

使用设计模式是为了提高代码的重用性，使代码更易理解并保证代码的可靠性，设计模式使代码编制真正工程化，是软件工程的基石。

**1.3 设计模式原理**

Java设计模式贯彻的原理是：面向接口编程

**1.4 设计模式的目标**

降低耦合，增强灵活性，设计模式告诉我们编程并不是API的简单拼凑和堆砌

### 设计模式分类

常用的设计模式可以概括为23种，按照特点可以将其分为三个类型，分别是创建型、结构性、行为型

* 1. **创建型**

作用：创建对象

方式：抽象实例化过程，帮助一个系统独立于其关联对象的创建、组合和表示方式

功能：

* 封装系统所用具体类的信息
* 隐藏类的实例是如何被创建和组合的，提供对象共同接口

分类：

常见的创建型设计模式有下面几种：

* 单例模式
* 工厂方法模式
* 抽象工厂模式
* 建造者模式
* 原型模式
  1. **结构型**

作用：对类和对象的结构进行探讨

分类：

* 类结构型模式：通过继承机制来组合接口或实现
* 对象结构型模式：通过组合一些对象实现新的功能

常见的结构型模式：

* 代理模式
* 装饰模式
* 适配器模式
* 组合模式
* 桥梁模式
* 外观模式
* 享元模式

**2.3 行为型**

介绍：行为型设计模式关注的是对象的行为，用来解决对象之间的联系问题

常见的行为型设计模式：

* 模板方法模式
* 命令模式
* 责任链模式
* 策略模式
* 迭代器模式
* 中介者模式
* 观察者模式
* 备忘录模式
* 访问者模式
* 状态模式
* 解释其模式

### 创建型设计模式

### 一 单例模式

确保一个类只有一个实例，而且自行实例化并向整个系统提供这个实例

特点：类的构造函数是私有有，从而避免外界利用构造函数创建出任意多的实例

两种表现形式：

* 饿汉式单例类：类加载时，就进行对象实例化
* 懒汉式单例类：第一次引用类时，才进行对象实例化

单例模式的使用场景：

* 要求生成唯一序列号的场景
* 在整个项目中需要一个共享访问点或共享数据
* 创建一个对象需要的资源过多，如访问IO和数据库等资源
* 需要大量的静态常量和静态方法(如工具类)的环境，也可以采用直接声明为static的方式

如spring框架中，每个bean默认是单例的

优点：

* 由于单例模式在内存中只有一个实例，减少了内存的开支，特别是当一个对象需要频繁地创建、销毁而且创建或销毁的性能又无法优化时。
* 减少了系统的性能开销，当一个对象的产生需要比较多的资源时，可以在启用时直接产生一个单例对象，然后用永久驻留内存的方式来解决
* 单例模式可以避免对资源的多重占用。如同一个资源文件避免同时被读写
* 单例模式可以在系统设置全局访问点，优化和共享资源文件

缺点：

* 单例模式无法创建子类，扩展困难
* 单例模式对测试不利，在并行开发环境中，如果采用单例模式的类没有完成，程序是不能进行测试的
* 单例模式和单一职责原则有冲突

### 二 工厂模式

工厂模式的分类：

* 简单工厂模式(静态工厂模式)
* 工厂方法模式
* 抽象工厂模式

**简单工厂模式**

定义：由一个工厂对象决定创建出哪一种产品类的实例，工厂类知道每一个产品

优点：允许客户端独立于产品的创建过程，并且在系统引入新产品时无需修改客户端，在某种程度上支持”开-闭”原则

缺点：对开闭原则的支持力度不够，一旦有新产品加入到系统中，则需要修改工厂类，将必要的逻辑加入到工厂类中

**工厂方法模式**

定义：定义一个用于创建对象的接口，让子类决定实例化哪个类。工厂方法使一个类的实例化延迟到其子类

对象：一种产品

优点：

* 核心的工厂类不再负责所有产品的创建，而是将具体创建的工作交给子类去实现
* 系统在不修改具体工厂角色的情况下引进新的产品，
* 子类提供挂钩。基类为工厂方法提供缺省实现，子类可以重写新的实现，也可以继承父类的实现。-- 加一层间接性，增加了灵活性
* 屏蔽产品类。产品类的实现如何变化，调用者都不需要关心，只需关心产品的接口，只要接口保持不变，系统中的上层模块就不会发生变化。
* 典型的解耦框架。高层模块只需要知道产品的抽象类，其他的实现类都不需要关心，符合迪米特法则，符合依赖倒置原则，符合里氏替换原则。
* 多态性：客户代码可以做到与特定应用无关，适用于任何实体类。

**抽象工厂模式**

定义：为创建一组相关或相互依赖的对象提供一个接口，而且无需指定他们的具体类

特点：工厂方法模式的升级版本，当多个业务品种需要进行分类时使用

对象：一族产品

优点：

* 包含工厂方法模式的优点
* 增加新的产品线非常容易，只需要实现产品组中所有产品接口并建立新的工厂类即可

缺点：

产品族本身的扩展非常困难，如果在产品族中增加一个新的产品类型时，需要修改多个接口，并且会影响已有的工厂类

使用场景：

当一个对象族，或是一组没有任何关系的对象，都有相同的约束时，可以使用抽象工厂模式

### 三 建造者模式

##### 将一个复杂对象的构建与它的表示分离，似的同样的构建过程可以创建不同的表示