升级SDK高级配置

Bugly文档资料

我们提供**Beta类**作为Bugly的初始化扩展，通过Beta类可以修改升级的检测时机，界面元素以及自定义升级行为，可以参考BetaSDKDemo的相关设置。

自动初始化开关

```
Beta.autoInit = true;
```

> true表示app启动自动初始化升级模块;
> false不会自动初始化;
> 开发者如果担心sdk初始化影响app启动速度，可以设置为false，在后面某个时刻手动调用Beta.init(getApplicationContext(), false);

自动检查更新开关

```
Beta.autoCheckUpgrade = true;
```

> true表示初始化时自动检查升级;
> false表示不会自动检查升级,需要手动调用Beta.checkUpgrade()方法;

升级检查周期设置

```
Beta.upgradeCheckPeriod = 60 * 1000;
```

> 设置升级检查周期为60s(默认检查周期为0s)，60s内SDK不重复向后台请求策略);

延迟初始化

```
Beta.initDelay = 1 * 1000;
```

> 设置启动延时为1s（默认延时3s），APP启动1s后初始化SDK，避免影响APP启动速度;

设置通知栏大图标

```
 Beta.largeIconId = R.drawable.ic_launcher;
```

> largeIconId为项目中的图片资源;

设置状态栏小图标

```
 Beta.smallIconId = R.drawable.ic_launcher;
```

> smallIconId为项目中的图片资源id

设置更新弹窗默认展示的banner

```
 Beta.defaultBannerId = R.drawable.ic_launcher;
```

> defaultBannerId为项目中的图片资源Id;
> 当后台配置的banner拉取失败时显示此banner，默认不设置则展示“loading…“;

设置sd卡的Download为更新资源存储目录

```
 Beta.storageDir = Environment.getExternalStoragePublicDirectory(Environm
ent.DIRECTORY_DOWNLOADS);
```

> 后续更新资源会保存在此目录，需要在manifest中添加WRITE_EXTERNAL_STORAGE权限;

设置开启显示打断策略

```
Beta.showInterruptedStrategy = true;
```

> 设置点击过确认的弹窗在App下次启动自动检查更新时会再次显示。

添加可显示弹窗的Activity

```
Beta.canShowUpgradeActs.add(MainActivity.class);
```

例如，只允许在MainActivity上显示更新弹窗，其他activity上不显示弹窗; 如果不设置默认所有activity都可以显示弹窗。 注意：自定义Activity不适用于这个场景。

设置自定义升级对话框UI布局

```
Beta.upgradeDialogLayoutId = R.layout.upgrade_dialog;
```

upgrade_dialog为项目的布局资源。
注意：因为要保持接口统一，需要用户在指定控件按照以下方式设置tag，否则会影响您的正常使用：

- 标题：beta_title，如：android:tag=”beta_title”
- 升级信息：beta_upgrade_info 如： android:tag=”beta_upgrade_info”
- 更新属性：beta_upgrade_feature 如： android:tag=”beta_upgrade_feature”
- 取消按钮：beta_cancel_button 如：android:tag=”beta_cancel_button”
- 确定按钮：beta_confirm_button 如：android:tag=”beta_confirm_button”

设置自定义tip弹窗UI布局

```
Beta.tipsDialogLayoutId = R.layout.tips_dialog;
```

注意：因为要保持接口统一，需要用户在指定控件按照以下方式设置tag，否则会影响您的正常使用：

- 标题：beta_title，如：android:tag=”beta_title”
- 提示信息：beta_tip_message 如： android:tag=”beta_tip_message”
- 取消按钮：beta_cancel_button 如：android:tag=”beta_cancel_button”
- 确定按钮：beta_confirm_button 如：android:tag=”beta_confirm_button”

设置升级对话框生命周期回调接口

```
Beta.upgradeDialogLifecycleListener = new UILifecycleListener<UpgradeInfo
>() {
            @Override
            public void onCreate(Context context, View view, UpgradeInfo
 upgradeInfo) {
                Log.d(TAG, "onCreate");
                // 注：可通过这个回调方式获取布局的控件，如果设置了id，可通过findVi
ewById方式获取，如果设置了tag，可以通过findViewWithTag，具体参考下面例子：

                // 通过id方式获取控件，并更改imageview图片
                ImageView imageView = (ImageView) view.findViewById(R.id.
icon);
                imageView.setImageResource(R.mipmap.ic_launcher);

                // 通过tag方式获取控件，并更改布局内容
                TextView textView = (TextView) view.findViewWithTag("text
view");
                textView.setText("my custom text");

                // 更多的操作：比如设置控件的点击事件
                imageView.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {我们提供**Beta类**作为Bugly
的初始化扩展，通过Beta类可以修改升级的检测时机，界面元素以及自定义升级行为，可以参考[Be
taSDKDemo](https://github.com/vell001/BetaSDKDemo)的相关设置。
```

自动初始化开关

```
 Beta.autoInit = true;
```

> true表示app启动自动初始化升级模块;
> false不会自动初始化;
> 开发者如果担心sdk初始化影响app启动速度，可以设置为false，在后面某个时刻手动调用Beta.init(getApplicationContext(),false);

自动检查更新开关

```
Beta.autoCheckUpgrade = true;
```

> true表示初始化时自动检查升级;
> false表示不会自动检查升级,需要手动调用Beta.checkUpgrade()方法;

升级检查周期设置

```
Beta.upgradeCheckPeriod = 60 * 1000;
```

> 设置升级检查周期为60s(默认检查周期为0s)，60s内SDK不重复向后台请求策略);

延迟初始化

```
Beta.initDelay = 1 * 1000;
```

> 设置启动延时为1s（默认延时3s），APP启动1s后初始化SDK，避免影响APP启动速度;

设置通知栏大图标

```
Beta.largeIconId = R.drawable.ic_launcher;
```

> largeIconId为项目中的图片资源;

设置状态栏小图标

```
Beta.smallIconId = R.drawable.ic_launcher;
```

> smallIconId为项目中的图片资源id

设置更新弹窗默认展示的banner

```
Beta.defaultBannerId = R.drawable.ic_launcher;
```

> defaultBannerId为项目中的图片资源Id;
> 当后台配置的banner拉取失败时显示此banner，默认不设置则展示“loading…“;

设置sd卡的Download为更新资源存储目录

```
Beta.storageDir = Environment.getExternalStoragePublicDirectory(Environm
ent.DIRECTORY_DOWNLOADS);
```

> 后续更新资源会保存在此目录，需要在manifest中添加WRITE_EXTERNAL_STORAGE权限;

设置开启显示打断策略

```
Beta.showInterruptedStrategy = true;
```

> 设置点击过确认的弹窗在App下次启动自动检查更新时会再次显示。

添加可显示弹窗的Activity

```
Beta.canShowUpgradeActs.add(MainActivity.class);
```

> 例如，只允许在MainActivity上显示更新弹窗，其他activity上不显示弹窗; 如果不设置默认所有activity都可以显示弹窗。

设置自定义升级对话框UI布局

```
 Beta.upgradeDialogLayoutId = R.layout.upgrade_dialog;
```

> upgrade_dialog为项目的布局资源。
> 注意：因为要保持接口统一，需要用户在指定控件按照以下方式设置tag，否则会影响您的正常使用：
>
> - 特性图片：beta_upgrade_banner，如：android:tag=”beta_upgrade_banner”
> - 标题：beta_title，如：android:tag=”beta_title”
> - 升级信息：beta_upgrade_info 如： android:tag=”beta_upgrade_info”
> - 更新属性：beta_upgrade_feature 如： android:tag=”beta_upgrade_feature”
> - 取消按钮：beta_cancel_button 如：android:tag=”beta_cancel_button”
> - 确定按钮：beta_confirm_button 如：android:tag=”beta_confirm_button”

设置自定义tip弹窗UI布局

```
Beta.tipsDialogLayoutId = R.layout.tips_dialog;
```

> 注意：因为要保持接口统一，需要用户在指定控件按照以下方式设置tag，否则会影响您的正常使用：
>
> - 标题：beta_title，如：android:tag=”beta_title”
> - 提示信息：beta_tip_message 如： android:tag=”beta_tip_message”
> - 取消按钮：beta_cancel_button 如：android:tag=”beta_cancel_button”
> - 确定按钮：beta_confirm_button 如：android:tag=”beta_confirm_button”

设置升级对话框生命周期回调接口

```
Beta.upgradeDialogLifecycleListener = new UILifecycleListener<UpgradeInfo
>() {
            @Override
            public void onCreate(Context context, View view, UpgradeInfo
 upgradeInfo) {
                Log.d(TAG, "onCreate");

            }

            @Override
            public void onStart(Context context, View view, UpgradeInfo u
pgradeInfo) {
                Log.d(TAG, "onStart");
            }

            @Override
            public void onResume(Context context, View view, UpgradeInfo
 upgradeInfo) {
                Log.d(TAG, "onResume");
                // 注：可通过这个回调方式获取布局的控件，如果设置了id，可通过findVi
ewById方式获取，如果设置了tag，可以通过findViewWithTag，具体参考下面例子：

                // 通过id方式获取控件，并更改imageview图片
                ImageView imageView = (ImageView) view.findViewById(R.id.
icon);
                imageView.setImageResource(R.mipmap.ic_launcher);

                // 通过tag方式获取控件，并更改布局内容
                TextView textView = (TextView) view.findViewWithTag("text
view");
                textView.setText("my custom text");

                // 更多的操作：比如设置控件的点击事件
                imageView.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        Intent intent = new Intent(getApplicationContext
(), OtherActivity.class);
                        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        startActivity(intent);
                    }
                });
            }

            @Override
            public void onPause(Context context, View view, UpgradeInfo u
pgradeInfo) {
                Log.d(TAG, "onPause");
            }

            @Override
```

```
            public void onStop(Context context, View view, UpgradeInfo up
gradeInfo) {
                Log.d(TAG, "onStop");
            }

            @Override
            public void onDestroy(Context context, View view, UpgradeInfo
 upgradeInfo) {
                Log.d(TAG, "onDestory");
            }

        };
```

如果想监听升级对话框的生命周期事件，可以通过设置OnUILifecycleListener接口
回调参数解释：

- context - 当前弹窗上下文对象
- view - 升级对话框的根布局视图，可通过这个对象查找指定view控件
- upgradeInfo - 升级信息

初始化统一接口

```
Bugly.init(this, APP_ID, false);
```

接口说明

更新功能主要API

```
    /**
     * 手动检查更新（用于设置页面中检测更新按钮的点击事件）
     */
    public static synchronized void checkUpgrade()

    /**
     * 获取本地已有升级策略（非实时，可用于界面红点展示）
     *
     * @return
     */
    public static synchronized UpgradeInfo getUpgradeInfo()

    /**
     * @param isManual  用户手动点击检查，非用户点击操作请传false
     * @param isSilence 是否显示弹窗等交互，[true:没有弹窗和toast] [false:有弹窗或toast]
     */
    public static synchronized void checkUpgrade(boolean isManual, boolean isSilence)
```

示例

```
public class MainActivity extends Activity {
        Button checkUpgradeBtn;
        Button refreshBtn;
        TextView upgradeInfoTv;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            checkUpgradeBtn = $(R.id.check_upgrade);
            refreshBtn = $(R.id.refresh_info);
            upgradeInfoTv = $(R.id.upgrade_info);

            checkUpgradeBtn.setOnClickListener(new OnClickListener() {
            @Override
                public void onClick(View v) {

                    /***** 检查更新 *****/
                    Beta.checkUpgrade();
                }
            });

            refreshBtn.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {
                    loadUpgradeInfo();
                }
            });
        }

        private void loadUpgradeInfo() {
            if (upgradeInfoTv == null)
                return;

            /***** 获取升级信息 *****/
            UpgradeInfo upgradeInfo = Beta.getUpgradeInfo();

            if (upgradeInfo == null) {
                upgradeInfoTv.setText("无升级信息");
                return;
            }
            StringBuilder info = new StringBuilder();
            info.append("id: ").append(upgradeInfo.id).append("\n");
            info.append("标题: ").append(upgradeInfo.title).append("\n");
            info.append("升级说明: ").append(upgradeInfo.newFeature).append
("\n");
            info.append("versionCode: ").append(upgradeInfo.versionCode).
append("\n");
            info.append("versionName: ").append(upgradeInfo.versionName).
append("\n");
```

```
            info.append("发布时间: ").append(upgradeInfo.publishTime).appen
d("\n");
            info.append("安装包Md5: ").append(upgradeInfo.apkMd5).append("\n");
            info.append("安装包下载地址: ").append(upgradeInfo.apkUrl).appen
d("\n");
            info.append("安装包大小: ").append(upgradeInfo.fileSize).append
("\n");
            info.append("弹窗间隔（ms）: ").append(upgradeInfo.popInterva
l).append("\n");
            info.append("弹窗次数: ").append(upgradeInfo.popTimes).append("\n");
            info.append("发布类型（0:测试 1:正式）: ").append(upgradeInfo.pub
lishType).append("\n");
            info.append("弹窗类型（1:建议 2:强制 3:手工）: ").append(upgradeI
nfo.upgradeType).append("\n");
            info.append("图片地址：").append(upgradeInfo.imageUrl);


            upgradeInfoTv.setText(info);
        }
    }
```

UpgradeInfo内容如下

```
    public String id = "";//唯一标识
    public String title = "";//升级提示标题
    public String newFeature = "";//升级特性描述
    public long publishTime = 0;//升级发布时间,ms
    public int publishType = 0;//升级类型 0测试 1正式
    public int upgradeType = 1;//升级策略 1建议 2强制 3手工
    public int popTimes = 0;//提醒次数
    public long popInterval = 0;//提醒间隔
    public int versionCode;
    public String versionName = "";
    public String apkMd5;//包md5值
    public String apkUrl;//APK的CDN外网下载地址
    public long fileSize;//APK文件的大小
    pubilc String imageUrl; // 图片url
```

自定义UI

这里我们提供两种自定义UI的方式：固定控件id方式和自定义activity两种方式；

1.固定控件id

使用固定控件id方式，弹窗的生命周期仍然由更新sdk管理，用户可以自定义升级弹窗的布局和网络弹窗布局，仅需要将自定义布局中的控件tag设置为指定tag，并在在初始化时设置自定义布局的资源id与回调。

> 接口说明

```
/*控件固定tag如下*/
public static final String TAG_IMG_BANNER = "beta_upgrade_banner";
public static final String TAG_TITLE = "beta_title";
public static final String TAG_UPGRADE_INFO = "beta_upgrade_info";
public static final String TAG_UPGRADE_FEATURE = "beta_upgrade_feature";
public static final String TAG_CANCEL_BUTTON = "beta_cancel_button";
public static final String TAG_CONFIRM_BUTTON = "beta_confirm_button";
public static final String TAG_TIP_MESSAGE = "beta_tip_message";
```

> tag与界面控件对应关系如下:

```
    /*自定义文案，修改后立即生效，默认控件文案如下*/
    public static String strToastYourAreTheLatestVersion = "你已经是最新版了";
    public static String strToastCheckUpgradeError = "检查新版本失败，请稍后重试";
    public static String strToastCheckingUpgrade = "正在检查，请稍候...";
    public static String strNotificationDownloading = "正在下载";
    public static String strNotificationClickToView = "点击查看";
    public static String strNotificationClickToInstall = "点击安装";
    public static String strNotificationClickToRetry = "点击重试";
    public static String strNotificationDownloadSucc = "下载完成";
    public static String strNotificationDownloadError = "下载失败";
    public static String strNotificationHaveNewVersion = "有新版本";
    public static String strNetworkTipsMessage = "你已切换到移动网络，是否继续当前下载？";
    public static String strNetworkTipsTitle = "网络提示";
    public static String strNetworkTipsConfirmBtn = "继续下载";
    public static String strNetworkTipsCancelBtn = "取消";
    public static String strUpgradeDialogVersionLabel = "版本";
    public static String strUpgradeDialogFileSizeLabel = "包大小";
    public static String strUpgradeDialogUpdateTimeLabel = "更新时间";
    public static String strUpgradeDialogFeatureLabel = "更新说明";
    public static String strUpgradeDialogUpgradeBtn = "立即更新";
    public static String strUpgradeDialogInstallBtn = "安装";
    public static String strUpgradeDialogRetryBtn = "重试";
    public static String strUpgradeDialogContinueBtn = "继续";
    public static String strUpgradeDialogCancelBtn = "下次再说";
```

```
  /**
     * 升级弹窗生命周期回调;
     * 用户可以在在弹窗的各个生命周期中回调，
     * 并在回调中修改控件内容
     */
    public interface UILifecycleListener<T> {
        /**
        * @param context    上下文
        * @param rootView   界面根节点，通过它获取子控件，并进行设置
        * @param t          自定义泛型参数，用于回调时传递其他信息
        */
        void onCreate(Context context, View rootView, T t);
        void onStart(Context context, View rootView, T t);
        void onResume(Context context, View rootView, T t);
        void onPause(Context context, View rootView, T t);
        void onStop(Context context, View rootView, T t);
        void onDestroy(Context context, View rootView, T t);
    }
```

> 示例

```
 /**
          *  设置自定义升级对话框UI布局
          *  注意：因为要保持接口统一，需要用户在指定控件按照以下方式设置tag，否则会影响您的正常使用：
          *  标题：beta_title，如：android:tag="beta_title"
          *  升级信息：beta_upgrade_info  如：  android:tag="beta_upgrade_info"
          *  更新属性：beta_upgrade_feature 如：  android:tag="beta_upgrade_feature"
          *  取消按钮：beta_cancel_button 如：android:tag="beta_cancel_button"
          *  确定按钮：beta_confirm_button 如：android:tag="beta_confirm_button"
          *  详见layout/upgrade_dialog.xml
          */
         Beta.upgradeDialogLayoutId = R.layout.upgrade_dialog;

         /**
          * 设置自定义tip弹窗UI布局
          * 注意：因为要保持接口统一，需要用户在指定控件按照以下方式设置tag，否则会影响您的正常使用：
          *  标题：beta_title，如：android:tag="beta_title"
          *  提示信息：beta_tip_message 如：  android:tag="beta_tip_message"
          *  取消按钮：beta_cancel_button 如：android:tag="beta_cancel_button"
          *  确定按钮：beta_confirm_button 如：android:tag="beta_confirm_button"
          *  详见layout/tips_dialog.xml
          */
         Beta.tipsDialogLayoutId = R.layout.tips_dialog;

         /**
          *  如果想监听升级对话框的生命周期事件，可以通过设置OnUILifecycleListener接口
          *  回调参数解释：
          *  context - 当前弹窗上下文对象
          *  view - 升级对话框的根布局视图，可通过这个对象查找指定view控件
          *  upgradeInfo - 升级信息
          */
        Beta.upgradeDialogLifecycleListener = new UILifecycleListener<UpgradeInfo>() {
             @Override
             public void onCreate(Context context, View view, UpgradeInfo upgradeInfo) {
                 Log.d(TAG, "onCreate");
                 // 注：可通过这个回调方式获取布局的控件，如果设置了id，可通过findViewById方式获取，如果设置了tag，可以通过findViewWithTag，具体参考下面例子：

                 // 通过id方式获取控件，并更改imageview图片
                 ImageView imageView = (ImageView) view.findViewById(R.id.imageview);
```

```
                imageView.setImageResource(R.mipmap.ic_launcher);

                // 通过tag方式获取控件，并更改布局内容
                TextView textView = (TextView) view.findViewWithTag("text
view");
                textView.setText("my custom text");

                // 更多的操作：比如设置控件的点击事件
                imageView.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        Intent intent = new Intent(getApplicationContext
(), OtherActivity.class);
                        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        startActivity(intent);
                    }
                });
            }

            @Override
            public void onStart(Context context, View view, UpgradeInfo u
pgradeInfo) {
                Log.d(TAG, "onStart");
            }

            @Override
            public void onResume(Context context, View view, UpgradeInfo
 upgradeInfo) {
                Log.d(TAG, "onResume");
            }

            @Override
            public void onPause(Context context, View view, UpgradeInfo u
pgradeInfo) {
                Log.d(TAG, "onPause");
            }

            @Override
            public void onStop(Context context, View view, UpgradeInfo up
gradeInfo) {
                Log.d(TAG, "onStop");
            }

            @Override
            public void onDestroy(Context context, View view, UpgradeInfo
 upgradeInfo) {
                Log.d(TAG, "onDestory");
            }
        };
```

2.自定义activity

使用自定义activity的方式，弹窗界面的绘制与生命周期均由用户自己维护，sdk负责在收到策略,下载时回调与事件上报，并提供相关接口控制任务下载，获取任务状态。

> 接口说明

```
    /**
     * 监听更新的各个状态，可以替换SDK内置的toast提示
     */
     public interface UpgradeStateListener {
         /**
          * 更新失败
          * @param isManual  true:手动检查 false:自动检查
          */
         void  onUpgradeFailed(boolean isManual);

         /**
          * 更新成功
          * @param isManual
          */
         void  onUpgradeSuccess(boolean isManual);

         /**
          * 没有更新
          * @param isManual
          */
         void  onUpgradeNoVersion(boolean isManual);

         /**
          * 正在更新
          * @param isManual
          */
         void  onUpgrading(boolean isManual);
     }

     /**
     * 更新监听，收到策略时回调
     */
     public interface UpgradeListener {
         /**
          * 接收到更新策略
          * @param ret  0:正常 -1:请求失败
          * @param strategy 更新策略
          * @param isManual true:手动请求 false:自动请求
          * @param isSilence true:不弹窗 false:弹窗
          * @return 是否放弃SDK处理此策略，true:SDK将不会弹窗，策略交由app自己处理
          */
         void onUpgrade(int ret,UpgradeInfo strategy,boolean isManual, boo
lean isSilence);
     }
     /**
     * 下载监听，下载时回调
     */
     public interface DownloadListener {
         /**
```

```
         * @param task 下载任务
         */
        void onReceive(DownloadTask task);
        /**
         * @param task 下载任务
         */
        void onCompleted(DownloadTask task);
        /**
         * 下载失败
         *
         * @param task 下载任务
         * @param code 错误码
         * @param extMsg 错误信息
         */
        void onFailed(DownloadTask task, int code, String extMsg);
    }
    /**
     * 注册下载监听
     * 在弹窗activity的oncreate中调用
     * @param dl 用户自己实现的下载接口
     */
    public static void registerDownloadListener(DownloadListener dl)
    /**
     * 注销下载监听
     * 在弹窗activity的ondestroy中调用，不使用会导致内存泄漏
     */
    public static void unregisterDownloadListener()
    /**
     * 控制下载
     * 控制任务的下载暂停等，用于下载按钮的监听
     * @return 返回下载任务
     */
    public static DownloadTask startDownload()
    /**
     * 取消下载
     * 用于取消按钮的监听
     */
    public static void cancelDownload()
    /**
     * 获取下载任务信息
     * 初始化弹窗时获取下载任务信息
     * @return 下载任务
     */
    public static DownloadTask getStrategyTask()
    /**
     * 获取本地已有升级策略
     *
     * @return
     */
    public static synchronized UpgradeInfo getUpgradeInfo()
```

demo

```
    /*在application中初始化时设置监听，监听策略的收取*/
        Beta.upgradeListener = new UpgradeListener() {
            @Override
            public void onUpgrade(int ret,UpgradeInfo strategy, boolean i
sManual, boolean isSilence) {
                if (strategy != null) {
                    Intent i = new Intent();
                    i.setClass(getApplicationContext(), UpgradeActivity.c
lass);
                    i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                    startActivity(i);
                } else {
                    Toast.makeText(BaseApplication.this, "没有更新", Toast.
LENGTH_LONG).show();
                }
            }
        };

/* 设置更新状态回调接口 */
    Beta.upgradeStateListener = new UpgradeStateListener() {
            @Override
            public void onUpgradeSuccess(boolean isManual) {
                Toast.makeText(getApplicationContext(),"UPGRADE_SUCCESS",
Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onUpgradeFailed(boolean isManual) {
                Toast.makeText(getApplicationContext(),"UPGRADE_FAILED",T
oast.LENGTH_SHORT).show();
            }

            @Override
            public void onUpgrading(boolean isManual) {

Toast.makeText(getApplicationContext(),"UPGRADE_CHECKING",Toast.LENGTH_SH
ORT).show();
            }

            @Override
            public void onUpgradeNoVersion(boolean isManual) {
                Toast.makeText(getApplicationContext(),"UPGRADE_NO_VERSIO
N",Toast.LENGTH_SHORT).show();
            }
        };
```

```
/**
 * 更新弹窗demo
 */
public class UpgradeActivity extends Activity {
    private TextView tv;
    private TextView title;
    private TextView version;
    private TextView size;
    private TextView time;
    private TextView content;
    private Button cancel;
    private Button start;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_upgrade);
        tv = getView(R.id.tv);
        title = getView(R.id.title);
        version = getView(R.id.version);
        size = getView(R.id.size);
        time = getView(R.id.time);
        content = getView(R.id.content);
        cancel = getView(R.id.cancel);
        start = getView(R.id.start);

        /*获取下载任务，初始化界面信息*/
        updateBtn(Beta.getStrategyTask());
        tv.setText(tv.getText().toString() + Beta.getStrategyTask().g
etSavedLength() + "");

        /*获取策略信息，初始化界面信息*/
        title.setText(title.getText().toString() + Beta.getUpgradeInf
o().title);
        version.setText(version.getText().toString() + Beta.getUpgrad
eInfo().versionName);
        size.setText(size.getText().toString() + Beta.getUpgradeInfo
().fileSize + "");
        time.setText(time.getText().toString() + Beta.getUpgradeInfo
().publishTime + "");
        content.setText(Beta.getUpgradeInfo().newFeature);

        /*为下载按钮设置监听*/
        start.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                DownloadTask task = Beta.startDownload();
                updateBtn(task);
                if (task.getStatus() == DownloadTask.DOWNLOADING) {
                    finish();
                }
```

```
                }
            });

            /*为取消按钮设置监听*/
            cancel.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Beta.cancelDownload();
                    finish();
                }
            });

            /*注册下载监听，监听下载事件*/
            Beta.registerDownloadListener(new DownloadListener() {
                @Override
                public void onReceive(DownloadTask task) {
                    updateBtn(task);
                    tv.setText(task.getSavedLength() + "");
                }

                @Override
                public void onCompleted(DownloadTask task) {
                    updateBtn(task);
                    tv.setText(task.getSavedLength() + "");
                }

                @Override
                public void onFailed(DownloadTask task, int code, String
extMsg) {
                    updateBtn(task);
                    tv.setText("failed");

                }
            });
        }

        @Override
        protected void onResume() {
            super.onResume();
        }

        @Override
        protected void onStop() {
            super.onStop();
        }

        @Override
        protected void onDestroy() {
            super.onDestroy();

            /*注销下载监听*/
            Beta.unregisterDownloadListener();
```

```
        }


        public void updateBtn(DownloadTask task) {

            /*根据下载任务状态设置按钮*/
            switch (task.getStatus()) {
                case DownloadTask.INIT:
                case DownloadTask.DELETED:
                case DownloadTask.FAILED: {
                    start.setText("开始下载");
                }
                break;
                case DownloadTask.COMPLETE: {
                    start.setText("安装");
                }
                break;
                case DownloadTask.DOWNLOADING: {
                    start.setText("暂停");
                }
                break;
                case DownloadTask.PAUSED: {
                    start.setText("继续下载");
                }
                break;
            }
        }

        public <T extends View> T getView(int id) {
            return (T) findViewById(id);
        }
    }
```