热更新API接口

安装Tinker(无参)

```
Beta.installTinker();
```

> 此接口仅用于反射Application方式接入。

安装Tinker

```
Beta.installTinker(this);
```

> 此接口仅用于改造Application方式接入，参数为ApplicationLike对象。

用户自定义扩展Tinker接口

```
// or you can just use DefaultLoadReporter
LoadReporter loadReporter = new SampleLoadReporter(getApplication());
// or you can just use DefaultPatchReporter
PatchReporter patchReporter = new SamplePatchReporter(getApplication());
// or you can just use DefaultPatchListener
PatchListener patchListener = new SamplePatchListener(getApplication());
// you can set your own upgrade patch if you need
AbstractPatch upgradePatchProcessor = new UpgradePatch();
// you can set your own repair patch if you need
AbstractPatch repairPatchProcessor = new RepairPatch();
TinkerManager.TinkerPatchResultListener patchResultListener = new TinkerM
anager.TinkerPatchResultListener() {
        @Override
      public void onPatchResult(PatchResult result) {
      // you can get the patch result
      }
};

Beta.installTinker(this, loadReporter, patchReporter, patchListener, patc
hResultListener, upgradePatchProcessor, repairPatchProcessor);
```

> 此接口仅用于改造Application方式接入。

加载指定路径的补丁

```
Beta.applyTinkerPatch(getApplicationContext(), Environment.getExternalStorageDirectory().getAbsolutePath() + "/patch_signed_7zip.apk");
```

加载Arm架构so库

```
Beta.loadArmLibrary(getApplicationContext(), "mylib");
```

> 只能用于加载armabi架构的so文件。

加载Arm-v7架构so库

```
Beta.loadArmV7Library(getApplicationContext(), "mylib");
```

> 只能用于加载armabi-v7a架构的so文件。

通用加载so库接口

```
Beta.loadLibrary("mylib");
```

> 通用加载so接口，我们自动帮你加载指定架构的so。

清除补丁

```
Beta.cleanTinkerPatch();
```

> 注：清除补丁之后，就会回退基线版本状态。

主动检查更新

```
Beta.checkUpgrade();
```

> 用于主动检查补丁策略的接口。

设置是否允许自动下载补丁

```
Beta.canAutoDownloadPatch = true;
```

> 默认为true，如果想选择下载补丁的时机，设置为false即可。

设置是否允许自动合成补丁

```
Beta.canAutoPatch = true;
```

> 默认为true，如果想选择合成补丁的时机，设置为false即可。

设置是否显示弹窗提示用户重启

```
Beta.canNotifyUserRestart = false
```

> 默认为false，如果想弹窗提示用户重启，设置为true即可。

补丁回调接口

```
    Beta.betaPatchListener = new BetaPatchListener() {
            @Override
            public void onPatchReceived(String patchFile) {
                Toast.makeText(getApplicationContext(), "补丁下载地址" + pa
tchFile , Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onDownloadReceived(long savedLength, long totalLe
ngth) {
                Toast.makeText(getApplicationContext(), String.format(Loc
ale.getDefault(),
                        "%s %d%%",
                        Beta.strNotificationDownloading,
                        (int) (totalLength == 0 ? 0 : savedLength * 100 /
 totalLength)), Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onDownloadSuccess(String msg) {
                Toast.makeText(getApplicationContext(),"补丁下载成功",Toas
t.LENGTH_SHORT).show();
            }

            @Override
            public void onDownloadFailure(String msg) {
                Toast.makeText(getApplicationContext(),"补丁下载成功",Toas
t.LENGTH_SHORT).show();

            }

            @Override
            public void onApplySuccess(String msg) {
                Toast.makeText(getApplicationContext(),"补丁应用成功",Toas
t.LENGTH_SHORT).show();
            }

            @Override
            public void onApplyFailure(String msg) {
                Toast.makeText(getApplicationContext(),"补丁应用失败",Toas
t.LENGTH_SHORT).show();
            }
        };
```

> 我们提供了补丁下载、合成各个时机的回调接口，方便开发者控制补丁的生命周期。