在工程项目中，为什么vendor目录下的platformconfigs.xml文件可以对部分手机的属性特性进行设置?

答：在framework目录下，android.provider.Settings文件为这种方式提供了可能。通过官方文档对其描述： “The Settings provider contains global system-level device preferences.” 。即android.provider.Setttings类是为设备系统提供全局的、系统级别的设备属性。

在Settings类中，包含五个外部可以访问的内部类：
Setttings.Global: 为设备系统提供全局的系统属性，该属性特性对设备上的任何用户都是无差别的。
Settings.NameValueTable: 根据key-value的组合，作为数据库中表的公共基础。

```
public static class NameValueTable implements BaseColumns {
    public static final String NAME = "name";
    public static final String VALUE = "value";

    protected static boolean putString(ContentResolver resolver, Uri uri,
            String name, String value) {
        // The database will take care of replacing duplicates.
        try {
            ContentValues values = new ContentValues();
            values.put(NAME, name);
            values.put(VALUE, value);
            resolver.insert(uri, values);
            return true;
        } catch (SQLException e) {
            Log.w(TAG, "Can't set key " + name + " in " + uri, e);
            return false;
        }
    }

    public static Uri getUriFor(Uri uri, String name) {
        return Uri.withAppendedPath(uri, name);
    }
}
```

Settings:Secure: 仅仅只有读权限，而没有写权限的系统属性。 Settings.SettingNotFoundException: 未找到系统属性的异常处理， Settings.System: 各类的系统系统属性。

如果想添加一个系统属性，既可以在该文件中进行添加相应的属性内容。比如：希望添加一个全局性的属性，即可以在Settings.Global类进行处理:

```
public static final String AIRPLANE_MODE_ON = "airplane_mode_on";

public static final String[] SETTINGS_TO_BACKUP = {
            ~~~~~
            AIRPLANE_MODE_ON,
            ~~~~~
```

可以通过

```
Settings.Global.getInt(getContentResolver(), Settings.Global.AIRPLANE_MODE_ON,0);
Settings.Global.putInt(getContentResolver(), Settings.Global.AIRPLANE_MODE_ON,0);
```

去得到和设置“AIRPLANE_MODE_ON”的属性值。

那么，PlatformConfigs.xml文件中的属性值，是如何写入到系统级文件build.prop文件中的呢?
答：在build/core/Makefile中，可以发现有这样一段内容：

```
# platform properties, which will be writen to build.prop
PLATFORM_PROPERTIES_SH :=
$(PWD)/vendor/tinno/platform/configs/$(TARGET_PRODUCT)/$(PROJECT_NAME)/PlatformConfigs
```

```
    # add for platform configs
    @if [ -f "$(PLATFORM_PROPERTIES_SH)" ]; then \
            echo 'PLATFORM_PROPERTIES_SH exits'; \
            echo >> $@; \
      echo "#" >> $@; \
      echo "# PLATFORM_PROPERTIES START" >> $@; \
      echo "#" >> $@; \
            while read word; do \
                    if [ -n "$$word" ]; then \
                        echo "the word = $$word"; \
                        key=`echo "$$word" | cut -d = -f 1`; \
                        echo "the key = $$key"; \
                        value=`echo "$$word" | cut -d = -f 2`; \
                        echo "the value = $$value"; \
                        sed -i "/^$$key=/d" $@; \
                        echo "$$word" >> $@;    \
                    fi; \
            done < $(PLATFORM_PROPERTIES_SH); \
      echo "#" >> $@; \
      echo "# PLATFORM_PROPERTIES END" >> $@; \
      echo "#" >> $@; \
    fi;
```

从这段脚本可以发现，其将PlatformConfig.xml的属性值写到build.prop文件上。

原文路径:

1. Android里拥有很多属性，每一个属性都有一个名称和值，类似于Map的方式。这些属性可以在开机启动时预先设定，亦可以是动态加载的。 系统启动时，将分别以相面的次序加载预先设定的Android属性。如果出现了重名属性，那么后者可以覆盖前者的属性值，即后者的优先级高于前者。

```
/default.prop ->
/system/build.prop ->
/system/default.prop ->
/data/local.prop ->
/data/property/*
```

persist.* : 以persist开始的属性会在 /data/property 存一个副本。也就是说，如果程序调用property_set设以persist为前缀的属性，系统会在 /data/property/* 记录下这个属性的副本，重启以后，这个属性依然存在。有点类似于持久化技术。如果调用property_set以非 persist 为前缀的属性，这个属性仅仅存在于手机运存中，在手机重启后，这个属性就初始化了。(pt)

ro.* : 不能修改，即表示系统属性。当然手机开发者可以对源代码进行修改。(read only)

2. 在应用程序属性使用方法

```
import android.os.SystemProperties;
SystemProperties.set("persist.sys.country"," china" );
```

在java里取得属性：

```
String vmHeapSize = SystemProperties.get("dalvik.vm.heapgrowthlimit", "24m");
```

也可以用SystemProperties.getBoolean，getInt等

3. 属性初始化的入口点是property_init ，在system/core/init/property_service.c中定义。它的主要工作是申请32k共享内存，其中前1k是属性区的头，后面31k可以存247个属性（受前1k头的限制）。property_init初始化完property以后，加载/default.prop的属性定义。

其它的系统属性（build.prop, local.prop,...）在start_property_service中加载。加载完属性服务创建一个socket和其他 进程通信（设置或读取属性）。

> Init进程poll属性的socket，等待和处理属性请求。如果有请求到来，则调用handle_property_set_fd来处理这个请求。在这个函数里，首先检查请求者的uid/gid看看是否有权限，如果有权限则调property_service.c中的property_set函数。

> 在property_set函数中，它先查找就没有这个属性，如果找到，更改属性。如果找不到，则添加新属性。更改时还会判断是不是“ro”属性，如果是，则不能更改。如果是persist的话还会写到/data/property/<name>中。

> 最后它会调property_changed，把事件挂到队列里，如果有人注册这个属性的话（比如init.rc中on property:ro.kernel.qemu=1），最终会调它的会调函数。

当然，对于在PlatformConfigs.xml文件中，有些属性可以使用**SystemProperties**（SystemProperties位于frameworks/base/core/java/android/os中）来进行管理。如果用户需要对某些属性进行全局的处理，即可以在PlatformConfigs.xml文件中添加相应的宏。示例： java代码：

```
    import android.os.SystemProperties;
    String value = SystemProperties.get("ro.pt.show_photos_name", "false");
```

PlatformConfigs.xml

```
<!---Add @ying-->
<!-- 设置，camera 显示标题 -->
<feature id="SYM_BD_SHOW_LAST" title="show not class CameraActivity">
<string name="ro.pt.show_photos_name" value="true"></string>
</feature>
<!--end @ying -->
```

再来看看源码：

```java
package android.os;
import java.util.ArrayList;

public class SystemProperties
{
    public static final int PROP_NAME_MAX = 31;
    public static final int PROP_VALUE_MAX = 91;

    private static final ArrayList<Runnable> sChangeCallbacks = new ArrayList<Runnable>();

    private static native String native_get(String key);
    private static native String native_get(String key, String def);
    private static native int native_get_int(String key, int def);
    private static native long native_get_long(String key, long def);
    private static native boolean native_get_boolean(String key, boolean def);
    private static native void native_set(String key, String def);
    private static native void native_add_change_callback();

    public static String get(String key) {
        if (key.length() > PROP_NAME_MAX) {
            throw new IllegalArgumentException("key.length > " + PROP_NAME_MAX);
        }
        return native_get(key);
    }

    public static String get(String key, String def) {
        if (key.length() > PROP_NAME_MAX) {
            throw new IllegalArgumentException("key.length > " + PROP_NAME_MAX);
        }
        return native_get(key, def);
    }

    public static boolean is_target(final String s) {
    return get("ro.target", "").equals(s);
    }

    public static boolean is_project(final String s) {
    return get("ro.project", "").equals(s);
    }

    /* qmb_pk */
    public static boolean is_qmb_pk() {
        return is_project("qmb_pk");
    }

    public static int getInt(String key, int def) {
        if (key.length() > PROP_NAME_MAX) {
            throw new IllegalArgumentException("key.length > " + PROP_NAME_MAX);
        }
        return native_get_int(key, def);
    }

    public static long getLong(String key, long def) {
        if (key.length() > PROP_NAME_MAX) {
            throw new IllegalArgumentException("key.length > " + PROP_NAME_MAX);
        }
        return native_get_long(key, def);
    }
```

```
    public static boolean getBoolean(String key, boolean def) {
        if (key.length() > PROP_NAME_MAX) {
            throw new IllegalArgumentException("key.length > " + PROP_NAME_MAX);
        }
        return native_get_boolean(key, def);
    }

    public static void set(String key, String val) {
        if (key.length() > PROP_NAME_MAX) {
            throw new IllegalArgumentException("key.length > " + PROP_NAME_MAX);
        }
        if (val != null && val.length() > PROP_VALUE_MAX) {
            throw new IllegalArgumentException("val.length > " +
                PROP_VALUE_MAX);
        }
        native_set(key, val);
    }

    public static void addChangeCallback(Runnable callback) {
        synchronized (sChangeCallbacks) {
            if (sChangeCallbacks.size() == 0) {
                native_add_change_callback();
            }
            sChangeCallbacks.add(callback);
        }
    }

    static void callChangeCallbacks() {
        synchronized (sChangeCallbacks) {
            //Log.i("foo", "Calling " + sChangeCallbacks.size() + " change callbacks!");
            if (sChangeCallbacks.size() == 0) {
                return;
            }
            ArrayList<Runnable> callbacks = new ArrayList<Runnable>(sChangeCallbacks);
            for (int i=0; i<callbacks.size(); i++) {
                callbacks.get(i).run();
            }
        }
    }
}
```

除了SystemProperties可以增加全局宏控外，还可以使用Context.getResource()方法，来处理相应模块的属性设置。该方法先在strings.xml、config.xml文件中定义各种字符类型，然后在java源码中调用。

```
<res>
<bool  name="test_about">boolean_value</bool>

<java>
boolean str= getResources().getBoolean(R.bool.test_about);
```