目录

1. PendingIntent 不可变标志 (Pending Intent Mutability)

变更内容: Android 12 要求明确指定 PendingIntent 是否可变，FLAG_IMMUTABLE 或 FLAG_MUTABLE 必须设置

参考链接: https://developer.android.com/about/versions/12/behavior-changes-12#pending-intent-mutability

代码示例:

```
// Create PendingIntent with explicit immutability specification
private PendingIntent createAlarmPendingIntent() {
    Intent intent = new Intent(this, AlarmReceiver.class);
    intent.setAction("com.example.app.ALARM_ACTION");
```

```
    // Explicitly specify FLAG_IMMUTABLE, indicating this PendingIntent cannot be modified
    return PendingIntent.getBroadcast(
            this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE);
}

// For PendingIntent that needs to be mutable (e.g., geofencing), use FLAG_MUTABLE
private PendingIntent createGeofencePendingIntent() {
    Intent intent = new Intent(this, GeofenceTransitionsIntentService.class);

    // Explicitly specify FLAG_MUTABLE, allowing system to modify data in this Intent
    return PendingIntent.getService(
            this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_MUTABLE);
}

// Create notification
private void showNotification() {
    Intent intent = new Intent(this, MainActivity.class);
    // Explicitly specify immutability, most notifications don't need mutability
    PendingIntent pendingIntent = PendingIntent.getActivity(
            this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE);

    NotificationCompat.Builder builder = new NotificationCompat.Builder(this, "channel_id")
            .setContentTitle("Notification Title")
            .setContentText("Notification Content")
            .setSmallIcon(R.drawable.ic_notification)
            .setContentIntent(pendingIntent);

    NotificationManager notificationManager =
            (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(
                "channel_id", "Channel Name", NotificationManager.IMPORTANCE_DEFAULT);
        notificationManager.createNotificationChannel(channel);
    }
```

```
    notificationManager.notify(1, builder.build());
}

// DIFFERENCE: For compatibility with different versions, use helper method
private int getPendingIntentImmutableFlag() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        return PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE;

// DIFFERENCE: Added FLAG_IMMUTABLE
    } else {
        return PendingIntent.FLAG_UPDATE_CURRENT; // DIFFERENCE: No FLAG_IMMUTABLE
    }
}
```

2.通知系统变更

变更点：通知模板和样式更新

Android 12 对通知系统进行了视觉和功能上的更新，使其与 Material You 设计语言保持一致。

```
// Notification implementation for Android 10 and Android 12
private void showNotification() {
    NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("Notification Title")
        .setContentText("Notification Content")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);

    // DIFFERENCE: Android 12 supports richer styles
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) { // Android 12 is API level 31
        builder.setStyle(new NotificationCompat.DecoratedCustomViewStyle());
        // On Android 12, notification visual style will automatically adapt to Material You design
    }
```

```
    NotificationManagerCompat notificationManager = NotificationManagerCompat.from(context);
    notificationManager.notify(notificationId, builder.build());
}
```

主要变化：

1. 视觉更新：通知 UI 更新，采用 Material You 设计语言
2. 更大的触摸目标：通知操作按钮更大，更容易点击
3. 自适应颜色：通知颜色可以适应系统主题
4. 改进的媒体控制：媒体通知获得了新的设计和功能
5. 通知动画：新的展开和折叠动画

3.系统 UI 标志弃用

变更内容

- SYSTEM_UI_FLAG_*常量已弃用
- 应改用 WindowInsetsController

参考链接：https://developer.android.com/sdk/api_diff/30/changes

旧 API (已弃用)

在 Android 11 之前，控制系统 UI 元素（如状态栏、导航栏）的可见性是通过 View.SYSTEM_UI_FLAG_*常量和 setSystemUiVisibility()方法实现的

```
// Method to manage system UI: hide bars, check visibility, set theme, and listen for changes
private void manageSystemUI() {
    View decorView = getWindow().getDecorView();
    WindowInsetsController controller = decorView.getWindowInsetsController();

    if (controller != null) {
        // DIFFERENCE: HIDE SYSTEM BARS
```

```
        controller.hide(WindowInsets.Type.systemBars());

        // DIFFERENCE: SET IMMERSIVE MODE BEHAVIOR
        controller.setSystemBarsBehavior(
            WindowInsetsController.BEHAVIOR_SHOW_TRANSIENT_BARS_BY_SWIPE
        );

        // DIFFERENCE: SET STATUS BAR TO LIGHT THEME
        controller.setSystemBarsAppearance(
            WindowInsetsController.APPEARANCE_LIGHT_STATUS_BARS,
            WindowInsetsController.APPEARANCE_LIGHT_STATUS_BARS
        );
    }

    // DIFFERENCE: CHECK SYSTEM UI VISIBILITY
    WindowInsets insets = decorView.getRootWindowInsets();
    boolean isStatusBarVisible = insets.isVisible(WindowInsets.Type.statusBars());
    boolean isNavBarVisible = insets.isVisible(WindowInsets.Type.navigationBars());

    // DIFFERENCE: LISTEN FOR SYSTEM UI VISIBILITY CHANGES
    decorView.setOnApplyWindowInsetsListener(new View.OnApplyWindowInsetsListener() {
        @Override
        public WindowInsets onApplyWindowInsets(View v, WindowInsets insets) {
            boolean isStatusBarVisible = insets.isVisible(WindowInsets.Type.statusBars());
            // Handle visibility changes
            return v.onApplyWindowInsets(insets);
        }
    });
}
```

4.Toast 通知限制

变更内容

- 阻止后台应用发送自定义 Toast

- 新增 addCallback()方法监听 Toast 显示和消失

```
// Before Android 11: Custom Toasts in Foreground and Background
// Create custom Toast
Toast toast = new Toast(context);

// Create custom view
LinearLayout layout = new LinearLayout(context);
layout.setBackgroundColor(Color.RED);
TextView textView = new TextView(context);
textView.setText("This is a custom Toast");
textView.setTextColor(Color.WHITE);
layout.addView(textView);

// Set custom view
toast.setView(layout);
toast.setDuration(Toast.LENGTH_LONG);

// Show Toast - works normally in both foreground and background
toast.show();

// DIFFERENCE: Cannot monitor Toast show and hide
// No callback mechanism

// Android 11 and Later: Changes and Restrictions
// 1. Background Custom Toast Restrictions
// In background service:
Toast toast = new Toast(context);
LinearLayout layout = new LinearLayout(context);
// ... set up custom view ...
toast.setView(layout);
toast.show();

// Result: Toast won't show, system will log in logcat:
// W/NotificationService: Blocking custom toast from package <package> due to package not in the
foreground
```

```
// 2. Background Text Toast Still Allowed
// In background service:
// This standard text Toast can still be shown in background
Toast.makeText(context, "This is a standard text Toast", Toast.LENGTH_LONG).show();

// 3. New Toast Callback API
// Create Toast
Toast toast = Toast.makeText(context, "Toast with callback", Toast.LENGTH_LONG);

// Add callback to monitor Toast show and hide
toast.addCallback(new Toast.Callback() {
    @Override
    public void onToastShown() {
        Log.d("ToastDemo", "Toast has been shown");
    }

    @Override
    public void onToastHidden() {
        Log.d("ToastDemo", "Toast has been hidden");
    }
});

toast.show();
// DIFFERENCE: Android 11 added the addCallback() method, allowing applications to monitor Toast show and hide events.

// 4. Text Toast API Changes
// Before Android 11
Toast toast = Toast.makeText(context, "Text Toast", Toast.LENGTH_LONG);
toast.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL, 0, 100);
View view = toast.getView(); // Returns actual view
float horizontalMargin = toast.getHorizontalMargin(); // Returns actual value

// Android 11 and later (for applications targeting SDK version >= 30)
Toast toast = Toast.makeText(context, "Text Toast", Toast.LENGTH_LONG);
```

```
toast.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL, 0, 100); // No effect
View view = toast.getView(); // Returns null
float horizontalMargin = toast.getHorizontalMargin(); // Return value doesn't reflect actual value
```

5.设备标识符限制

变更名称：设备标识符限制

变更内容：

Android 12 进一步限制了应用访问设备标识符的能力，增强了用户隐私保护。

参考链接：

https://developer.android.com/about/versions/12/behavior-changes-all?hl=zh-cn#device-identifiers

示例代码：

```
// Method to access device identifiers, highlighting differences between Android 10 and Android 12
private void getDeviceIdentifiers() {
    // DIFFERENCE: IMEI Access
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.S) {
        // Android 10: Access IMEI (requires READ_PHONE_STATE permission)
        TelephonyManager telephonyManager = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);
        String imei = telephonyManager.getImei();
    } else {
        // Android 12: IMEI access is restricted, requires special permission
        if (checkSelfPermission(Manifest.permission.READ_PHONE_STATE) ==
PackageManager.PERMISSION_GRANTED) {
            TelephonyManager telephonyManager = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);
            // Getting IMEI requires special permission
            // String imei = telephonyManager.getImei(); // May not be accessible
        }
```

```
    }

    // DIFFERENCE: MAC Address Access
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.S) {
        // Android 10: Get MAC address
        WifiManager wifiManager = (WifiManager)
getApplicationContext().getSystemService(Context.WIFI_SERVICE);
        WifiInfo wifiInfo = wifiManager.getConnectionInfo();
        String macAddress = wifiInfo.getMacAddress();
    } else {
        // Android 12: MAC address access is restricted
        // Cannot get real MAC address, use alternative methods
    }

    // Android ID Access
    String androidId = Settings.Secure.getString(getContentResolver(), Settings.Secure.ANDROID_ID);

    // RECOMMENDATION: Use more modern identification methods
    String appSpecificId = UUID.randomUUID().toString();
    // Store in SharedPreferences
    SharedPreferences prefs = getSharedPreferences("app_prefs", MODE_PRIVATE);
    if (!prefs.contains("app_id")) {
        prefs.edit().putString("app_id", appSpecificId).apply();
    }
    String storedId = prefs.getString("app_id", appSpecificId);
}
```

6 分区存储强制执行

变更内容

Android 10（分区存储可选择退出）

Android 10 引入了分区存储，但应用可以通过在清单文件中添加 requestLegacyExternalStorage 属性选择退出：

```
<manifest ... >
  <application
    android:requestLegacyExternalStorage="true"
    ... >
    ...
  </application>
</manifest>
```

设置此标志后，应用仍可使用传统存储访问模式。

Android 11（强制执行分区存储）

在 Android 11 中，无论 requestLegacyExternalStorage 设置如何，所有针对 API 30 的应用都必须使用分区存储。以下是新的访问模式：

1. 应用专用目录访问

```
// 访问应用专用目录（不需要存储权限）
File appSpecificExternalDir = getExternalFilesDir(null);
File myFile = new File(appSpecificExternalDir, "data.txt");

try {
  FileOutputStream fos = new FileOutputStream(myFile);
  fos.write("Hello World".getBytes());
  fos.close();
  // 成功写入文件到应用专用目录
} catch (IOException e) {
  e.printStackTrace();
}

// 这些文件会在应用卸载时被删除
```

2. 媒体文件访问（使用 MediaStore API）

```
// 保存图片到共享媒体存储
ContentValues values = new ContentValues();
values.put(MediaStore.Images.Media.DISPLAY_NAME, "my_image.jpg");
values.put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg");
values.put(MediaStore.Images.Media.RELATIVE_PATH, "Pictures/MyApp");

ContentResolver resolver = getContentResolver();
Uri imageUri = resolver.insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);

try {
    OutputStream os = resolver.openOutputStream(imageUri);
    // 将图片数据写入输出流
    Bitmap bitmap = getBitmapFromSomewhere();
    bitmap.compress(Bitmap.CompressFormat.JPEG, 90, os);
    os.close();
} catch (IOException e) {
    e.printStackTrace();
}

// 查询自己创建的媒体文件（需要 READ_EXTERNAL_STORAGE 权限）
String[] projection = {
    MediaStore.Images.Media._ID,
    MediaStore.Images.Media.DISPLAY_NAME
};

String selection = MediaStore.Images.Media.RELATIVE_PATH + " LIKE ?";
String[] selectionArgs = new String[]{"Pictures/MyApp%"};

TestModel testModel = resolver.query(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    projection,
    selection,
    selectionArgs,
    null
);
```

```
// 处理查询结果
while (testModel.moveToNext()) {
    // 获取图片信息
}
testModel.close();
```

3. 访问其他应用的文件（不再允许）

```
// Android 11 中，这段代码将失败
File externalDir = Environment.getExternalStorageDirectory();
File otherAppFile = new File(externalDir, "OtherApp/data.txt");
try {
    FileInputStream fis = new FileInputStream(otherAppFile);
    // 将抛出异常，无法访问其他应用的文件
    fis.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

4. 使用存储访问框架（SAF）访问文件

```
// 启动文件选择器让用户选择文件
Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
intent.addCategory(Intent.CATEGORY_OPENABLE);
intent.setType("*/*");
startActivityForResult(intent, REQUEST_CODE);

// 在 onActivityResult 中处理选择的文件
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
        if (data != null) {
            Uri uri = data.getData();
            try {
```

```
            InputStream is = getContentResolver().openInputStream(uri);
            // 读取文件内容
            is.close();

            // 保持长期访问权限（如果需要）
            getContentResolver().takePersistableUriPermission(uri,
                Intent.FLAG_GRANT_READ_URI_PERMISSION);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  }
}
```

5. 访问特定目录（使用 SAF）

```
// 启动目录选择器
Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT_TREE);
startActivityForResult(intent, REQUEST_DIRECTORY_CODE);

// 在 onActivityResult 中处理选择的目录
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode == REQUEST_DIRECTORY_CODE && resultCode == RESULT_OK) {
      if (data != null) {
         Uri treeUri = data.getData();

         // 保持长期访问权限
         getContentResolver().takePersistableUriPermission(treeUri,
            Intent.FLAG_GRANT_READ_URI_PERMISSION |
            Intent.FLAG_GRANT_WRITE_URI_PERMISSION);

         // 使用 DocumentFile 访问目录内容
         DocumentFile pickedDir = DocumentFile.fromTreeUri(this, treeUri);
```

```
        DocumentFile[] files = pickedDir.listFiles();
        for (DocumentFile file : files) {
            Log.d("Files", "Found: " + file.getName());
        }

        // 在选定目录中创建新文件
        DocumentFile newFile = pickedDir.createFile("text/plain", "newfile.txt");
        try {
            OutputStream os = getContentResolver().openOutputStream(newFile.getUri());
            os.write("Hello World".getBytes());
            os.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
  }
}
```

7. 前台服务启动限制 (Foreground Service Launch Restrictions)

变更内容: Android 12 限制了从后台启动前台服务的能力，必须使用特定方式触发

参考链接: https://developer.android.com/about/versions/12/behavior-changes-12#foreground-service-launch-restrictions

代码示例:

```
// BackgroundWorker class for starting services
public class BackgroundWorker extends Worker {
  @NonNull
  @Override
  public Result doWork() {
    Context context = getApplicationContext();
```

```
// DIFFERENCE: Starting Foreground Service
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    // Android 12: Cannot start foreground service directly from background
    // Use notification to prompt user action
    Intent serviceIntent = new Intent(context, MyForegroundService.class);
    PendingIntent pendingIntent = PendingIntent.getService(
            context, 0, serviceIntent, PendingIntent.FLAG_IMMUTABLE);

    NotificationCompat.Builder builder = new NotificationCompat.Builder(context, "channel_id")
            .setContentTitle("Service Start Required")
            .setContentText("Click this notification to start necessary service")
            .setSmallIcon(R.drawable.ic_notification)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true);

    NotificationManager notificationManager =
            (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);

    // Ensure notification channel is created
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(
                "channel_id", "Channel Name", NotificationManager.IMPORTANCE_DEFAULT);
        notificationManager.createNotificationChannel(channel);
    }

    notificationManager.notify(2, builder.build());
} else {
    // Android 11 and below: Can start foreground service directly
    Intent serviceIntent = new Intent(context, MyForegroundService.class);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        context.startForegroundService(serviceIntent);
    } else {
        context.startService(serviceIntent);
    }
}
```

```
        return Result.success();
    }
}

// MyForegroundService class for foreground service implementation
public class MyForegroundService extends Service {
    @Override
    public void onCreate() {
        super.onCreate();

        // Create notification channel for foreground service
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel(
                    "channel_id", "Channel Name", NotificationManager.IMPORTANCE_DEFAULT);
            NotificationManager notificationManager = getSystemService(NotificationManager.class);
            notificationManager.createNotificationChannel(channel);
        }

        Notification notification = new NotificationCompat.Builder(this, "channel_id")
                .setContentTitle("Foreground Service")
                .setContentText("Service is running")
                .setSmallIcon(R.drawable.ic_notification)
                .build();

        startForeground(1, notification);
    }
}

// ForegroundWorker class using WorkManager for foreground tasks
public class ForegroundWorker extends Worker {
    public ForegroundWorker(@NonNull Context context, @NonNull WorkerParameters params) {
        super(context, params);
    }

    @NonNull
```

```
@Override
public Result doWork() {
    // Execute work that needs to be done in foreground service
    // ...

    return Result.success();
}

// Use WorkManager to start foreground service (recommended method)
public static void enqueueForegroundWork(Context context) {
    Constraints constraints = new Constraints.Builder()
            .setRequiredNetworkType(NetworkType.CONNECTED)
            .build();

    // Create foreground info
    ForegroundInfo foregroundInfo = createForegroundInfo(context);

    // Create one-time work request
    OneTimeWorkRequest workRequest = new OneTimeWorkRequest.Builder(ForegroundWorker.class)
            .setConstraints(constraints)
            .build();

    // Enqueue work request
    WorkManager.getInstance(context)
            .enqueueUniqueWork(
                    "foreground_work",
                    ExistingWorkPolicy.REPLACE,
                    workRequest);

    // Set as foreground service
    WorkManager.getInstance(context).getWorkInfoByIdLiveData(workRequest.getId())
            .observeForever(workInfo -> {
                if (workInfo != null && workInfo.getState() == WorkInfo.State.RUNNING) {
                    WorkManager.getInstance(context).setForegroundAsync(
                            workRequest.getId(), foregroundInfo);
                }
```

```
            });
    }

    // Create foreground info needed for foreground service
    private static ForegroundInfo createForegroundInfo(Context context) {
        // Create notification channel
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel(
                    "foreground_channel",
                    "Foreground Work",
                    NotificationManager.IMPORTANCE_LOW);

            NotificationManager notificationManager =
                    (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
            notificationManager.createNotificationChannel(channel);
        }

        // Create notification
        Notification notification = new NotificationCompat.Builder(context, "foreground_channel")
                .setContentTitle("Foreground Work")
                .setContentText("Work in progress...")
                .setSmallIcon(R.drawable.ic_notification)
                .setOngoing(true)
                .build();

        return new ForegroundInfo(NOTIFICATION_ID, notification);
    }
}
```

8. 启动画面(SplashScreen)变更

变更点：启动画面 API

Android 12 引入了新的 SplashScreen API，这是一个重大变更，为所有应用提供了统一的启动体验。

```
// Android 10 Implementation: Manual Splash Screen
public class SplashActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);

        // DIFFERENCE: Manual delay for splash screen
        new Handler().postDelayed(() -> {
            startActivity(new Intent(SplashActivity.this, MainActivity.class));
            finish();
        }, 2000);
    }
}

// AndroidManifest.xml for Android 10
<activity
    android:name=".SplashActivity"
    android:theme="@style/SplashTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

// Android 12 Implementation: Using SplashScreen API
// Configure in themes.xml
<style name="Theme.App" parent="Theme.MaterialComponents.DayNight.NoActionBar">
    <!-- DIFFERENCE: Splash screen configuration in theme -->
    <item name="android:windowSplashScreenBackground">@color/splash_background</item>
    <item name="android:windowSplashScreenAnimatedIcon">@drawable/splash_icon</item>
    <item name="android:windowSplashScreenAnimationDuration">1000</item>
    <item name="android:windowSplashScreenBrandingImage">@drawable/branding_image</item>
</style>

// MainActivity for Android 12
```

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // DIFFERENCE: Use SplashScreen API on Android 12
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            SplashScreen splashScreen = getSplashScreen();

            // Extend splash screen display time until data loading is complete
            splashScreen.setOnExitAnimationListener(splashScreenView -> {
                // Execute custom exit animation
                ObjectAnimator fadeOut = ObjectAnimator.ofFloat(
                    splashScreenView.getView(),
                    View.ALPHA,
                    1f,
                    0f
                );
                fadeOut.setDuration(500);
                fadeOut.addListener(new AnimatorListenerAdapter() {
                    @Override
                    public void onAnimationEnd(Animator animation) {
                        splashScreenView.remove();
                    }
                });
                fadeOut.start();
            });
        }

        setContentView(R.layout.activity_main);
    }
}
```

主要变化：

1. 自动生成启动画面：Android 12 会为所有应用自动生成启动画面，无需开发者手动创建专门的启动 Activity
2. 统一体验：提供了一致的过渡动画，从应用图标到应用内容
3. 声明式配置：通过主题属性配置启动画面外观
4. 可编程控制：可以通过代码控制启动画面的显示时长和退出动画
5. 品牌展示：支持在启动画面底部显示品牌图像
6. 兼容性支持：通过 Core Splashscreen 库可向后兼容到 API 级别 23

9. 自定义意图过滤器验证 (Custom Intent Filter Verification)

变更内容: Android 12 对应用间交互的意图过滤器有更严格的验证

参考链接: https://developer.android.com/about/versions/12/behavior-changes-12#custom-intent-filter-verification

代码示例:

```
<!-- AndroidManifest.xml -->
<!-- Android 10: Declare intent filters -->
<activity android:name=".CustomActivity">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="custom" android:host="example.com" />
  </intent-filter>
</activity>

<!-- Android 12: Declare intent filters with export state -->
<activity
```

```
    android:name=".CustomActivity"
    android:exported="true"> <!-- DIFFERENCE: Must declare exported state -->
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="custom" android:host="example.com" />
    </intent-filter>
</activity>


// Android 10: Send custom intent
private void sendCustomIntent() {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("custom://example.com/path"));
    startActivity(intent);
}

// Android 12: Send custom intent safely
private void sendCustomIntentSafely() {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("custom://example.com/path"));

    // DIFFERENCE: Verify receiver exists before sending intent
    PackageManager packageManager = getPackageManager();
    List<ResolveInfo> activities = packageManager.queryIntentActivities(
            intent, PackageManager.MATCH_DEFAULT_ONLY);

    if (activities.size() > 0) {
        // Found application that can handle this intent
        startActivity(intent);
    } else {
        // No application can handle this intent
        Toast.makeText(this, "No application can handle this action", Toast.LENGTH_SHORT).show();

        // Provide fallback option
```

```
        Intent browserIntent = new Intent(Intent.ACTION_VIEW);
        browserIntent.setData(Uri.parse("https://example.com/path"));
        startActivity(browserIntent);
    }
}
```

10.AppSearch 和 WebView 变更

变更名称：AppSearch 和 WebView 变更

变更内容：

Android 12 引入了 AppSearch API，提供了更强大的应用内搜索功能，同时对 WebView 进行了多项改进。

参考链接：

https://developer.android.com/about/versions/12/features#appsearch

https://developer.android.com/about/versions/12/behavior-changes-all?hl=zh-cn#webview

示例代码：

```
// Android 10: In-app search using SQLite
public class SearchManager {
    private SQLiteDatabase db;

    public SearchManager(Context context) {
        SearchDbHelper dbHelper = new SearchDbHelper(context);
        db = dbHelper.getWritableDatabase();
    }

    public List<SearchResult> search(String query) {
        List<SearchResult> results = new ArrayList<>();
        Cursor cursor = db.query(
            "search_table",
```

```
            new String[]{"id", "title", "content"},
            "title LIKE ? OR content LIKE ?",
            new String[]{"%" + query + "%", "%" + query + "%"},
            null, null, null);

        while (cursor.moveToNext()) {
            SearchResult result = new SearchResult(
                cursor.getLong(0),
                cursor.getString(1),
                cursor.getString(2)
            );
            results.add(result);
        }
        cursor.close();
        return results;
    }
}

// Android 12: In-app search using AppSearch API
@RequiresApi(api = Build.VERSION_CODES.S)
public class AppSearchManager {
    private AppSearchSession searchSession;

    public AppSearchManager(Context context) {
        // Initialize AppSearch
        AppSearchManager appSearchManager = (AppSearchManager)
context.getSystemService(Context.APP_SEARCH_SERVICE);
        appSearchManager.createSearchSession(
            new SearchContext.Builder(context, "database_name").build())
            .addOnSuccessListener(session -> {
                searchSession = session;
            });
    }

    public void search(String query, SearchResultsCallback callback) {
        if (searchSession == null) return;
```

```
        SearchSpec searchSpec = new SearchSpec.Builder()
            .setTermMatch(SearchSpec.TERM_MATCH_PREFIX)
            .build();

        searchSession.search(
            query,
            searchSpec,
            new Executor() {
                @Override
                public void execute(Runnable command) {
                    new Handler(Looper.getMainLooper()).post(command);
                }
            },
            new SearchResultsCallback() {
                @Override
                public void onResult(SearchResults results) {
                    List<SearchResult> searchResults = new ArrayList<>();
                    for (SearchResult result : results.getResults()) {
                        // Process search results
                        searchResults.add(result);
                    }
                    callback.onResult(results);
                }
            });
    }
}

// WebView usage for both Android 10 and Android 12
WebView webView = findViewById(R.id.webview);
WebSettings settings = webView.getSettings();
settings.setJavaScriptEnabled(true);
webView.loadUrl("https://example.com");

// Android 12: WebView improvements
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
```

```
    // Set WebView render priority
    settings.setWebViewRenderProcessGoneListener((webView1, renderer) -> {
        // Handle render process crash
        return true; // Return true indicates application has handled the crash
    });

    // Use new Cookie management API
    CookieManager.getInstance().setAcceptThirdPartyCookies(webView, false);
}
```

11.近似位置权限

变更点：位置权限精细化

Android 12 引入了近似位置权限，允许用户只授予应用近似位置而非精确位置。

```
// Location permission request for Android 10 and Android 12
private void requestLocationPermission() {
    // DIFFERENCE: Android 12 allows users to choose between precise and approximate location
    String[] permissions = new String[]{
        Manifest.permission.ACCESS_FINE_LOCATION
    };

    requestPermissions(permissions, REQUEST_LOCATION_PERMISSION);
}

// Check if has precise location permission
private boolean hasExactLocationPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        // Android 12 and above needs to check if has precise location permission
        return checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) ==
                PackageManager.PERMISSION_GRANTED;
    } else {
        // Old versions with any location permission is precise
        return checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) ==
```

```
            PackageManager.PERMISSION_GRANTED;
    }
}

// Get location for Android 10 and Android 12
private void getLocation() {
    boolean hasCoarseLocation = checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION)
==
            PackageManager.PERMISSION_GRANTED;

    if (hasCoarseLocation) {
        LocationManager locationManager =
            (LocationManager) getSystemService(Context.LOCATION_SERVICE);

        // DIFFERENCE: Choose provider based on permission type
        String provider = hasExactLocationPermission() ?
                LocationManager.GPS_PROVIDER : LocationManager.NETWORK_PROVIDER;

        locationManager.requestLocationUpdates(provider, 0, 0, locationListener);
    }
}
```

主要变化：

1. 近似位置选项：用户可以选择只授予应用近似位置权限
2. 权限对话框更新：位置权限对话框增加了精确/近似选项
3. 权限检查：应用需要检查是否获得了精确位置权限
4. 降级处理：应用需要处理只有近似位置的情况
5. 位置精度：近似位置精度约为 3 公里范围

12. 精确闹钟权限 (Exact Alarm Permission)

变更内容: Android 12 需要 SCHEDULE_EXACT_ALARM 权限才能设置精确闹钟

参考链接: https://developer.android.com/about/versions/12/behavior-changes-12#exact-alarm-permission

代码示例:

```
<!-- AndroidManifest.xml -->
<!-- DIFFERENCE: Add permission for exact alarms in Android 12 -->
<uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM" />


// Schedule exact alarm for Android 10 and Android 12
private void scheduleExactAlarm() {
    AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);

    // DIFFERENCE: Check permission for exact alarms in Android 12
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        if (!alarmManager.canScheduleExactAlarms()) {
            // No permission, guide user to grant permission
            Intent intent = new Intent(Settings.ACTION_REQUEST_SCHEDULE_EXACT_ALARM);
            intent.setData(Uri.parse("package:" + getPackageName()));
            startActivity(intent);
            return;
        }
    }

    Intent intent = new Intent(this, AlarmReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(
            this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE);

    // Set alarm time
    long triggerTimeMillis = System.currentTimeMillis() + 60 * 60 * 1000; // 1 hour later
```

```
    // Set exact alarm
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        alarmManager.setExactAndAllowWhileIdle(
                AlarmManager.RTC_WAKEUP, triggerTimeMillis, pendingIntent);
    } else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
        alarmManager.setExact(AlarmManager.RTC_WAKEUP, triggerTimeMillis, pendingIntent);
    } else {
        alarmManager.set(AlarmManager.RTC_WAKEUP, triggerTimeMillis, pendingIntent);
    }
}
```

13. 非 SDK 接口限制 (Non-SDK Interface Restrictions)

变更内容: Android 12 进一步限制了对非 SDK 接口的访问，更多之前可用的非公开 API 被列入灰名单或黑名单

参考链接: https://developer.android.com/about/versions/12/behavior-changes-12#non-sdk-interfaces

代码示例:

```
// Android 10: Accessing non-SDK interfaces using reflection
private void accessHiddenApis() {
    try {
        // DIFFERENCE: Access hidden method of ActivityManager
        Class<?> activityManagerClass = Class.forName("android.app.ActivityManager");
        Method getDefaultMethod = activityManagerClass.getDeclaredMethod("getDefault");
        getDefaultMethod.setAccessible(true);
        Object activityManagerInstance = getDefaultMethod.invoke(null);

        // Access hidden field
        Field mConfigField = activityManagerClass.getDeclaredField("mConfiguration");
        mConfigField.setAccessible(true);
        Object config = mConfigField.get(activityManagerInstance);
```

```
        Log.d("HiddenAPI", "Successfully accessed hidden API: " + config);
    } catch (Exception e) {
        Log.e("HiddenAPI", "Failed to access hidden API", e);
    }
}

// Android 12: Using public APIs and handling non-SDK interfaces
private void usePublicApis() {
    // DIFFERENCE: Use public ActivityManager API
    ActivityManager activityManager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);

    // Get configuration information
    Configuration configuration = getResources().getConfiguration();

    Log.d("PublicAPI", "Using public API: " + configuration);

    // Enable StrictMode to detect non-SDK interface usage
    StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
            .detectNonSdkApiUsage()
            .penaltyLog()
            .build());

    // If really need to access non-SDK interfaces, check API availability before using reflection
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
        try {
            Method forNameMethod = Class.class.getDeclaredMethod("forName", String.class);
            Method getDeclaredMethodMethod = Class.class.getDeclaredMethod("getDeclaredMethod",
                    String.class, Class[].class);

            Class<?> vmRuntimeClass = (Class<?>) forNameMethod.invoke(null, "dalvik.system.VMRuntime");
            Method getRuntime = (Method) getDeclaredMethodMethod.invoke(vmRuntimeClass, "getRuntime",
                    null);
            Method setHiddenApiExemptions = (Method) getDeclaredMethodMethod.invoke(vmRuntimeClass,
                    "setHiddenApiExemptions", new Class[] { String[].class });

            Object vmRuntime = getRuntime.invoke(null);
```

```
            setHiddenApiExemptions.invoke(vmRuntime, new Object[] { new String[] { "L" } });
        } catch (Exception e) {
            Log.e("HiddenAPI", "Cannot enable hidden API access", e);
        }
    }
}
```

14.Intent 接收器显式声明

变更名称：Intent 接收器显式声明

变更内容：

Android 12 要求开发者明确声明 Intent 接收器的 exported 属性，增强了应用安全性。

参考链接：

https://developer.android.com/about/versions/12/behavior-changes-all?hl=zh-cn#receiver-exported

示例代码：

```
// Android 10: Send custom intent
private void sendCustomIntent() {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("custom://example.com/path"));
    startActivity(intent);
}

// Android 12: Send custom intent safely
private void sendCustomIntentSafely() {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("custom://example.com/path"));

    // DIFFERENCE: Verify receiver exists before sending intent
    PackageManager packageManager = getPackageManager();
```

```
    List<ResolveInfo> activities = packageManager.queryIntentActivities(
            intent, PackageManager.MATCH_DEFAULT_ONLY);

    if (activities.size() > 0) {
        // Found application that can handle this intent
        startActivity(intent);
    } else {
        // No application can handle this intent
        Toast.makeText(this, "No application can handle this action", Toast.LENGTH_SHORT).show();

        // Provide fallback option
        Intent browserIntent = new Intent(Intent.ACTION_VIEW);
        browserIntent.setData(Uri.parse("https://example.com/path"));
        startActivity(browserIntent);
    }
}


<!-- AndroidManifest.xml -->
<!-- Android 10: Declare intent filters -->
<activity android:name=".CustomActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="custom" android:host="example.com" />
    </intent-filter>
</activity>

<!-- Android 12: Declare intent filters with export state -->
<activity
    android:name=".CustomActivity"
    android:exported="true"> <!-- DIFFERENCE: Must declare exported state -->
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
```

```
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="custom" android:host="example.com" />
  </intent-filter>
</activity>
```

15. 应用休眠功能 (App Hibernation)

变更内容: Android 12 引入应用休眠功能，长期未使用的应用将进入休眠状态，权限会被撤销，缓存会被清除

参考链接: https://developer.android.com/about/versions/12/behavior-changes-12#app-hibernation

代码示例:

```
// Android 10: Perform background operations without checking hibernation state
private void performBackgroundOperations() {
  // Unconditionally execute background operations
  syncData();
  prefetchContent();
  updateCache();
}

// Android 12: Perform background operations adaptively based on hibernation state
private void performBackgroundOperationsAdaptively() {
  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    AppHibernationManager hibernationManager = getSystemService(AppHibernationManager.class);

    if (hibernationManager != null &&
        !hibernationManager.isHibernatingForUser(getPackageName())) {
      // App is not in hibernation state, can perform operations
      syncData();
      prefetchContent();
      updateCache();
    } else {
```

```
            // App is in hibernation state, minimize operations
            Log.d("Hibernation", "App is in hibernation state, skipping non-essential operations");
        }
    } else {
        // Android 11 and below don't have hibernation feature
        performBackgroundOperations();
    }
}

// Schedule background tasks for both Android 10 and Android 12
private void scheduleBackgroundWork() {
    WorkManager workManager = WorkManager.getInstance(this);

    // Create periodic work request
    PeriodicWorkRequest workRequest = new PeriodicWorkRequest.Builder(
            SyncWorker.class,
            1, TimeUnit.HOURS)
            .build();

    workManager.enqueueUniquePeriodicWork(
            "sync_work",
            ExistingPeriodicWorkPolicy.REPLACE,
            workRequest);
}

// Android 12: Detect app usage patterns and adjust operations
private void adaptToUsagePatterns() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
        UsageStatsManager usageStatsManager =
                (UsageStatsManager) getSystemService(Context.USAGE_STATS_SERVICE);

        long now = System.currentTimeMillis();
        long dayAgo = now - (24 * 60 * 60 * 1000);

        // Get app usage stats for last day
        if (checkUsageStatsPermission()) {
```

```
        List<UsageStats> stats = usageStatsManager.queryUsageStats(
                UsageStatsManager.INTERVAL_DAILY, dayAgo, now);

        // Check if has recent usage
        boolean hasRecentUsage = false;
        for (UsageStats usageStats : stats) {
            if (usageStats.getPackageName().equals(getPackageName()) &&
                usageStats.getLastTimeUsed() > dayAgo) {
                hasRecentUsage = true;
                break;
            }
        }

        // Adjust cache size and sync frequency based on usage
        if (hasRecentUsage) {
            increaseCacheQuota();
            scheduleFrequentSync();
        } else {
            reduceCacheQuota();
            scheduleInfrequentSync();
        }
    }
  }
}

// Android 12: Handle app recovery from hibernation
@Override
protected void onResume() {
  super.onResume();

  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    AppHibernationManager hibernationManager = getSystemService(AppHibernationManager.class);

    if (hibernationManager != null &&
        hibernationManager.isHibernatingForUser(getPackageName())) {
      // App is recovering from hibernation
```

```
        Log.d("Hibernation", "App is recovering from hibernation");

        // Reinitialize necessary components
        reinitializeComponents();

        // Request necessary permissions (might have been revoked during hibernation)
        checkAndRequestPermissions();

        // Rebuild cache
        rebuildCache();
    }
  }
}
```

16.蓝牙权限变更

变更点：蓝牙权限精细化

Android 12 对蓝牙权限进行了细分，增强了用户隐私保护。

```
// Request Bluetooth permissions for Android 10 and Android 12
private void requestBluetoothPermissions() {
  List<String> permissionsList = new ArrayList<>();

  // DIFFERENCE: Android 12 requires explicit Bluetooth permissions
  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    // New Bluetooth permissions for Android 12
    permissionsList.add(Manifest.permission.BLUETOOTH_SCAN);
    permissionsList.add(Manifest.permission.BLUETOOTH_CONNECT);
    permissionsList.add(Manifest.permission.BLUETOOTH_ADVERTISE);
  } else {
    // Android 10 and below require location permission for Bluetooth scanning
    permissionsList.add(Manifest.permission.ACCESS_FINE_LOCATION);
  }
```

```
    String[] permissions = permissionsList.toArray(new String[0]);
    requestPermissions(permissions, REQUEST_BLUETOOTH_PERMISSIONS);
}

// Start Bluetooth scanning for Android 10 and Android 12
private void startBluetoothScan() {
    BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    if (bluetoothAdapter != null && bluetoothAdapter.isEnabled()) {
        // DIFFERENCE: Check for new permissions in Android 12
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            if (checkSelfPermission(Manifest.permission.BLUETOOTH_SCAN) !=
                    PackageManager.PERMISSION_GRANTED) {
                // No permission, request permission
                return;
            }
        }

        BluetoothLeScanner scanner = bluetoothAdapter.getBluetoothLeScanner();
        scanner.startScan(scanCallback);
    }
}
```

主要变化：

1. 权限细分：将蓝牙功能拆分为 BLUETOOTH_SCAN、BLUETOOTH_CONNECT 和 BLUETOOTH_ADVERTISE 三个独立权限
2. 与位置权限分离：蓝牙扫描不再自动需要位置权限
3. 精确控制：用户可以更精确地控制应用的蓝牙访问范围
4. 运行时权限：所有蓝牙权限都是运行时权限，需要动态请求
5. 权限描述：需要在清单文件中提供权限使用原因

17. 模糊组件导出 (Fuzzy Component Export)

变更内容: Android 12 要求所有组件明确声明 exported 属性，不再允许根据意图过滤器隐式决定导出状态

参考链接: https://developer.android.com/about/versions/12/behavior-changes-all#exported

代码示例:

```
// Android 10: Send broadcast
private void sendBroadcast() {
    Intent intent = new Intent("com.example.app.CUSTOM_ACTION");
    intent.putExtra("data", "some_data");
    sendBroadcast(intent);
}

// Android 12: Send internal and public broadcasts
private void sendInternalBroadcast() {
    Intent intent = new Intent("com.example.app.CUSTOM_ACTION");
    intent.putExtra("data", "some_data");
    // DIFFERENCE: Specify receiver package
    intent.setPackage(getPackageName());
    sendBroadcast(intent);
}

private void sendPublicBroadcast() {
    Intent intent = new Intent("com.example.app.PUBLIC_ACTION");
    intent.putExtra("data", "public_data");
    // DIFFERENCE: Specify required permission
    sendBroadcast(intent, "com.example.app.CUSTOM_PERMISSION");
}

// Access content provider for Android 10 and Android 12
private void accessContentProvider() {
    // Android 12: Check for permission before accessing
    if (ContextCompat.checkSelfPermission(this, "com.example.app.READ_PROVIDER")
```

```
        == PackageManager.PERMISSION_GRANTED) {
        ContentResolver resolver = getContentResolver();
        Uri uri = Uri.parse("content://com.example.app.provider/items");
        Cursor cursor = resolver.query(uri, null, null, null, null);
        // Process query results...
        if (cursor != null) {
            cursor.close();
        }
    } else {
        // No permission, cannot access
        Log.e("ContentProvider", "No permission to access content provider");
    }
}


<!-- AndroidManifest.xml -->
<!-- Android 10: No need to specify exported attribute -->
<receiver android:name=".MyBroadcastReceiver">
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
    </intent-filter>
</receiver>

<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.app.provider" />

<!-- Android 12: Must specify exported attribute -->
<!-- Set to false for internal components -->
<receiver
    android:name=".MyBroadcastReceiver"
    android:exported="false"> <!-- DIFFERENCE: Specify exported attribute -->
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
    </intent-filter>
</receiver>
```

```
<!-- Set to true for components accessed by other apps, with permission protection -->
<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.app.provider"
    android:exported="true" <!-- DIFFERENCE: Specify exported attribute -->
    android:permission="com.example.app.READ_PROVIDER"
    android:writePermission="com.example.app.WRITE_PROVIDER" />

<!-- Declare custom permissions for exported components -->
<permission
    android:name="com.example.app.READ_PROVIDER"
    android:protectionLevel="signature" />
<permission
    android:name="com.example.app.WRITE_PROVIDER"
    android:protectionLevel="signature" />
```

18. 带宽检测限制 (Microphone and Camera Toggle)

变更内容: Android 12 允许用户在快速设置中完全禁用麦克风和相机，应用需要处理此情况

参考链接: https://developer.android.com/about/versions/12/behavior-changes-12#mic-camera-toggles

代码示例:

```
// Capture photo for Android 10 and Android 12
private void capturePhoto() {
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
            == PackageManager.PERMISSION_GRANTED) {

        CameraManager cameraManager = (CameraManager) getSystemService(Context.CAMERA_SERVICE);

        // DIFFERENCE: Check system-level camera toggle in Android 12
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
```

```
try {
    String cameraId = cameraManager.getCameraIdList()[0];
    cameraManager.openCamera(cameraId, new CameraDevice.StateCallback() {
        @Override
        public void onOpened(@NonNull CameraDevice camera) {
            // Camera opened successfully
            mCameraDevice = camera;
            createCameraPreviewSession();
        }

        @Override
        public void onDisconnected(@NonNull CameraDevice camera) {
            camera.close();
            mCameraDevice = null;
        }

        @Override
        public void onError(@NonNull CameraDevice camera, int error) {
            camera.close();
            mCameraDevice = null;

            if (error == CameraDevice.StateCallback.ERROR_CAMERA_DISABLED) {
                // Camera is disabled by system
                Toast.makeText(MainActivity.this,
                        "Camera is disabled by system, please enable in settings",
                        Toast.LENGTH_LONG).show();
                showCameraDisabledDialog();
            } else {
                Toast.makeText(MainActivity.this,
                        "Camera error: " + error,
                        Toast.LENGTH_SHORT).show();
            }
        }
    }, null);
} catch (CameraAccessException | SecurityException e) {
    Log.e("Camera", "Cannot access camera", e);
```

```
            Toast.makeText(this, "Cannot access camera, please check system settings",
                    Toast.LENGTH_SHORT).show();
            showCameraDisabledDialog();
        }
    } else {
        // Camera usage for Android 11 and below
        try {
            String cameraId = cameraManager.getCameraIdList()[0];
            cameraManager.openCamera(cameraId, cameraStateCallback, null);
        } catch (CameraAccessException e) {
            Log.e("Camera", "Cannot access camera", e);
        }
    }
  } else {
    // Request camera permission
    ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.CAMERA},
            REQUEST_CAMERA_PERMISSION);
  }
}

// Show camera disabled dialog
private void showCameraDisabledDialog() {
  new AlertDialog.Builder(this)
        .setTitle("Camera Disabled")
        .setMessage("Camera is disabled by system. Please go to Settings > Privacy > Camera toggle to enable
camera access.")
        .setPositiveButton("Go to Settings", (dialog, which) -> {
            Intent intent = new Intent(Settings.ACTION_PRIVACY_SETTINGS);
            startActivity(intent);
        })
        .setNegativeButton("Cancel", null)
        .show();
}

// Start recording for Android 10 and Android 12
```

```
private void startRecording() {
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.RECORD_AUDIO)
            == PackageManager.PERMISSION_GRANTED) {

        audioRecorder = new MediaRecorder();

        try {
            audioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
            audioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
            audioRecorder.setOutputFile(getRecordingFilePath());
            audioRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

            audioRecorder.prepare();

            // DIFFERENCE: Check system-level microphone toggle in Android 12
            try {
                audioRecorder.start();
                isRecording = true;

                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                    Handler handler = new Handler();
                    handler.postDelayed(() -> {
                        if (audioRecorder != null) {
                            try {
                                int amplitude = audioRecorder.getMaxAmplitude();
                                if (amplitude <= 0 && isRecording) {
                                    Log.w("AudioRecorder", "Recording amplitude is 0, microphone might be disabled");
                                }
                            } catch (Exception e) {
                                // Ignore
                            }
                        }
                    }, 1000);
                }
            } catch (IllegalStateException e) {
                Log.e("AudioRecorder", "Failed to start recording", e);
```

```
            Toast.makeText(this, "Cannot start recording, please check if microphone is disabled",
                    Toast.LENGTH_SHORT).show();
            releaseMediaRecorder();
            showMicrophoneDisabledDialog();
        }
    } catch (IOException | IllegalArgumentException | IllegalStateException e) {
        Log.e("AudioRecorder", "Recording setup failed", e);
        releaseMediaRecorder();
        if (e instanceof IllegalArgumentException || e instanceof IllegalStateException) {
            Toast.makeText(this, "Cannot use microphone, please check system settings",
                    Toast.LENGTH_SHORT).show();
            showMicrophoneDisabledDialog();
        }
    }
  } else {
    // Request recording permission
    ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.RECORD_AUDIO},
            REQUEST_RECORD_AUDIO_PERMISSION);
  }
}

// Show microphone disabled dialog
private void showMicrophoneDisabledDialog() {
  new AlertDialog.Builder(this)
        .setTitle("Microphone Disabled")
        .setMessage("Microphone is disabled by system. Please go to Settings > Privacy > Microphone toggle to
enable microphone access.")
        .setPositiveButton("Go to Settings", (dialog, which) -> {
            Intent intent = new Intent(Settings.ACTION_PRIVACY_SETTINGS);
            startActivity(intent);
        })
        .setNegativeButton("Cancel", null)
        .show();
}
```

```
// Release MediaRecorder resources
private void releaseMediaRecorder() {
    if (audioRecorder != null) {
        if (isRecording) {
            try {
                audioRecorder.stop();
            } catch (Exception e) {
                // Ignore stop exceptions
            }
            isRecording = false;
        }
        audioRecorder.reset();
        audioRecorder.release();
        audioRecorder = null;
    }
}
```

19. 剪贴板访问通知 (Clipboard Access Toast)

变更内容: Android 12 当应用访问剪贴板内容时会显示通知，应用应该明确何时需要访问剪贴板

参考链接: https://developer.android.com/about/versions/12/behavior-changes-12#clipboard-access

代码示例:

```
// Android 10: Access clipboard anytime without notification
private void pasteFromClipboard() {
    ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);

    // Read clipboard without user interaction
    if (clipboard.hasPrimaryClip()) {
        ClipData clipData = clipboard.getPrimaryClip();
        if (clipData != null && clipData.getItemCount() > 0) {
```

```
            CharSequence text = clipData.getItemAt(0).getText();
            if (text != null) {
                // Auto-fill text to input field
                EditText editText = findViewById(R.id.edit_text);
                editText.setText(text);
            }
        }
    }
}

// Android 12: Access clipboard only when explicitly requested by user
private void setupPasteButton() {
    Button pasteButton = findViewById(R.id.paste_button);
    EditText editText = findViewById(R.id.edit_text);

    // Provide explicit paste button
    pasteButton.setOnClickListener(v -> {
        // Access clipboard when user explicitly clicks paste button
        ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);

        if (clipboard.hasPrimaryClip()) {
            ClipData clipData = clipboard.getPrimaryClip();
            if (clipData != null && clipData.getItemCount() > 0) {
                CharSequence text = clipData.getItemAt(0).getText();
                if (text != null) {
                    // Fill text to input field
                    editText.setText(text);
                }
            }
        }
    });

    // Use system paste menu
    editText.setOnLongClickListener(v -> {
        // Show context menu on long press, including system paste option
        return false; // Allow system to handle long press
```

```
    });
}

// Android 10: Automatically read clipboard on app launch
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Read clipboard on app launch
    pasteFromClipboard();

    // Monitor clipboard changes
    ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
    clipboard.addPrimaryClipChangedListener(() -> {
        // Automatically read clipboard when content changes
        pasteFromClipboard();
    });
}

// Android 12: Avoid automatically monitoring clipboard changes
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Set up paste button
    setupPasteButton();

    // Don't read clipboard automatically on app launch
    // Don't register clipboard change listener to automatically read content
}

// Check for relevant content when specific conditions are met
private void checkForRelevantContent() {
    // Only access clipboard in specific scenarios
```

```
    EditText trackingField = findViewById(R.id.tracking_number);
    if (trackingField.hasFocus()) {
        ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);

        if (clipboard.hasPrimaryClip()) {
            ClipData clipData = clipboard.getPrimaryClip();
            if (clipData != null && clipData.getItemCount() > 0) {
                CharSequence text = clipData.getItemAt(0).getText();
                if (text != null && isTrackingNumberFormat(text.toString())) {
                    // Notify user they can paste this content
                    Toast.makeText(this, "Detected tracking number in clipboard, tap paste button to use",
                            Toast.LENGTH_SHORT).show();
                    findViewById(R.id.paste_button).setVisibility(View.VISIBLE);
                }
            }
        }
    }
}

// Check if text matches tracking number format
private boolean isTrackingNumberFormat(String text) {
    // Implement tracking number format validation logic
    return text.matches("\\d{12,15}") ||
           text.matches("[A-Z]{2}\\d{9}[A-Z]{2}");
}
```

20. 更安全的组件导出 (Safer Component Exporting)

变更内容: Android 12 要求应用更安全地导出组件，包括提供更严格的权限和意图过滤器

参考链接: https://developer.android.com/about/versions/12/behavior-changes-all#exported

代码示例:

```
// Android 10: Send broadcast
private void sendBroadcast() {
    Intent intent = new Intent("com.example.app.CUSTOM_ACTION");
    intent.putExtra("data", "some_data");
    sendBroadcast(intent);
}

// Android 12: Send internal and public broadcasts
private void sendInternalBroadcast() {
    Intent intent = new Intent("com.example.app.CUSTOM_ACTION");
    intent.putExtra("data", "some_data");
    // DIFFERENCE: Specify receiver package
    intent.setPackage(getPackageName());
    sendBroadcast(intent);
}

private void sendPublicBroadcast() {
    Intent intent = new Intent("com.example.app.PUBLIC_ACTION");
    intent.putExtra("data", "public_data");
    // DIFFERENCE: Specify required permission
    sendBroadcast(intent, "com.example.app.CUSTOM_PERMISSION");
}

// Access content provider for Android 10 and Android 12
private void accessContentProvider() {
    // Android 12: Check for permission before accessing
    if (ContextCompat.checkSelfPermission(this, "com.example.app.READ_PROVIDER")
            == PackageManager.PERMISSION_GRANTED) {
        ContentResolver resolver = getContentResolver();
        Uri uri = Uri.parse("content://com.example.app.provider/items");
        Cursor cursor = resolver.query(uri, null, null, null, null);
        // Process query results...
        if (cursor != null) {
            cursor.close();
        }
    } else {
```

```
        // No permission, cannot access
        Log.e("ContentProvider", "No permission to access content provider");
    }
}


<!-- AndroidManifest.xml -->
<!-- Android 10: Declare broadcast receiver without specifying exported attribute -->
<receiver android:name=".MyBroadcastReceiver">
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
    </intent-filter>
</receiver>

<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.app.provider" />

<!-- Android 12: Must specify exported attribute -->
<!-- Internal broadcast receiver -->
<receiver
    android:name=".MyBroadcastReceiver"
    android:exported="false"> <!-- DIFFERENCE: Specify exported attribute -->
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
    </intent-filter>
</receiver>

<!-- Public broadcast receiver with permission -->
<receiver
    android:name=".PublicBroadcastReceiver"
    android:exported="true" <!-- DIFFERENCE: Specify exported attribute -->
    android:permission="com.example.app.CUSTOM_PERMISSION">
    <intent-filter>
        <action android:name="com.example.app.PUBLIC_ACTION" />
    </intent-filter>
```

```
</receiver>


<!-- Content provider with permissions -->
<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.app.provider"
    android:exported="true" <!-- DIFFERENCE: Specify exported attribute -->
    android:permission="com.example.app.READ_PROVIDER"
    android:writePermission="com.example.app.WRITE_PROVIDER" />


<!-- Declare custom permissions -->
<permission
    android:name="com.example.app.CUSTOM_PERMISSION"
    android:protectionLevel="signature" />
<permission
    android:name="com.example.app.READ_PROVIDER"
    android:protectionLevel="signature" />
<permission
    android:name="com.example.app.WRITE_PROVIDER"
    android:protectionLevel="signature" />
```

21.材料设计组件更新

变更点：Material You 设计语言

Android 12 引入了 Material You 设计语言，提供了更个性化的 UI 体验，包括动态颜色系统。

```
<!-- Android 10: Use fixed theme colors -->
<style name="Theme.App" parent="Theme.MaterialComponents">
    <item name="colorPrimary">@color/primary</item>
    <item name="colorPrimaryDark">@color/primary_dark</item>
    <item name="colorAccent">@color/accent</item>
</style>


<!-- Android 12: Use dynamic colors in themes.xml -->
```

```
<style name="Theme.App" parent="Theme.Material3.DayNight">
    <item name="colorPrimary">@color/material_dynamic_primary40</item>
    <item name="colorPrimaryDark">@color/material_dynamic_primary80</item>
    <item name="colorAccent">@color/material_dynamic_secondary40</item>
</style>


// Android 10: Use fixed colors in code
button.setBackgroundColor(getResources().getColor(R.color.primary));

// Android 12: Get and use dynamic colors in code
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    // DIFFERENCE: Get system-extracted dynamic colors
    Context context = getContext();
    TypedArray dynamicColors = context.obtainStyledAttributes(
            new int[] {
                android.R.attr.colorPrimary,
                android.R.attr.colorAccent
            });
    int dynamicPrimary = dynamicColors.getColor(0, 0);
    int dynamicAccent = dynamicColors.getColor(1, 0);
    dynamicColors.recycle();

    // Apply dynamic colors
    button.setBackgroundColor(dynamicPrimary);
}
```

主要变化：

1. 动态颜色系统：根据用户壁纸自动提取颜色方案
2. Material You 组件：更新的 UI 组件，支持更圆润的形状和动态主题
3. 个性化主题：允许应用适应用户的个性化偏好
4. Material 3：新的设计系统，提供更现代的视觉语言

5. 自适应布局：更好地适应不同屏幕尺寸和形状

22.微件(Widget)改进

变更点：微件 API 更新

Android 12 对微件系统进行了重大更新，提供了更现代的外观和更好的用户体验。

```
// Widget implementation for Android 10 and Android 12
public class ExampleAppWidget extends AppWidgetProvider {
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        for (int appWidgetId : appWidgetIds) {
            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.example_widget);
            views.setTextViewText(R.id.widget_text, "Widget Example");

            // Set click event
            Intent intent = new Intent(context, MainActivity.class);
            // DIFFERENCE: Use PendingIntent.FLAG_IMMUTABLE for Android 12
            PendingIntent pendingIntent = PendingIntent.getActivity(
                context, 0, intent, PendingIntent.FLAG_IMMUTABLE);
            views.setOnClickPendingIntent(R.id.widget_layout, pendingIntent);

            // DIFFERENCE: Use new rounded corners API for Android 12
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                views.setViewOutlinePreference(R.id.widget_layout,
                    RemoteViews.OUTLINE_PREFERENCE_ROUNDED);
            }

            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}
```

```
<!-- widget_info.xml for Android 10 and Android 12 -->
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="1800000"
    android:initialLayout="@layout/example_widget"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen"
    android:description="@string/widget_description" <!-- DIFFERENCE: Add description for Android 12 -->
    android:targetCellWidth="2" <!-- DIFFERENCE: Specify target cell dimensions for Android 12 -->
    android:targetCellHeight="1">
</appwidget-provider>
```

主要变化：

1. 圆角支持：微件现在支持圆角边框，与系统 UI 风格一致
2. 响应式布局：改进的布局系统，更好地适应不同尺寸
3. 动态颜色：支持 Material You 动态颜色系统
4. 性能改进：更新机制优化，提供更流畅的体验
5. 新属性：新增 targetCellWidth 和 targetCellHeight 等属性，更好地控制微件尺寸
6. PendingIntent 安全性：必须使用 FLAG_IMMUTABLE 或 FLAG_MUTABLE 标志

23.屏幕截图和录制功能

变更点：屏幕截图和录制 API

Android 12 引入了新的屏幕截图和录制 API，同时增加了隐私保护措施。

```
// Android 10: Screen capture using MediaProjection
private void startScreenCapture() {
    MediaProjectionManager projectionManager =
```

```
        (MediaProjectionManager) getSystemService(Context.MEDIA_PROJECTION_SERVICE);
    startActivityForResult(projectionManager.createScreenCaptureIntent(), REQUEST_CODE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
        // Use MediaProjection for screen recording
        MediaProjection mediaProjection = projectionManager.getMediaProjection(resultCode, data);
        // Set recording parameters...
    }
}

// Android 12: Screenshot API
@RequiresApi(api = Build.VERSION_CODES.S)
private void takeScreenshot() {
    // Create screenshot callback
    ScreenshotCallback screenshotCallback = new ScreenshotCallback() {
        @Override
        public void onScreenCaptured(Bitmap bitmap, Rect source) {
            // Handle screenshot
            // bitmap contains screenshot content
        }

        @Override
        public void onScreenCaptureError(int errorCode) {
            // Handle error
        }
    };

    // Request screenshot
    ScreenshotManager screenshotManager =
        getSystemService(ScreenshotManager.class);
    screenshotManager.takeScreenshot(
        WindowManager.ScreenshotSource.SCREENSHOT_OTHER,
        new Handler(Looper.getMainLooper()),
```

```
        screenshotCallback);
}

// Android 12: Screen capture with privacy indicator
@RequiresApi(api = Build.VERSION_CODES.S)
private void startScreenCapture() {
    MediaProjectionManager projectionManager =
        (MediaProjectionManager) getSystemService(Context.MEDIA_PROJECTION_SERVICE);
    startActivityForResult(projectionManager.createScreenCaptureIntent(), REQUEST_CODE);
    // DIFFERENCE: Android 12 shows a recording indicator in the status bar
}
```

主要变化：

1. 官方截图 API：提供了原生的屏幕截图 API
2. 隐私指示器：录制屏幕时会显示永久性指示器
3. 限制后台访问：限制了后台应用访问屏幕内容
4. 安全增强：增加了对敏感内容的保护机制
5. 截图编辑：系统截图工具提供了更多编辑选项

24.微件 API 更新(widgets)

变更内容：

Android 12 对微件系统进行了重大更新，提供了更现代的外观和更好的用户体验，包括圆角支持、响应式布局和动态颜色支持。

参考链接：

https://developer.android.com/about/versions/12/features#widgets

示例代码：

```
// Widget implementation for Android 10 and Android 12
public class ExampleAppWidget extends AppWidgetProvider {
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        for (int appWidgetId : appWidgetIds) {
            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.example_widget);
            views.setTextViewText(R.id.widget_text, "Widget Example");

            // Set click event
            Intent intent = new Intent(context, MainActivity.class);
            // DIFFERENCE: Use PendingIntent.FLAG_IMMUTABLE for Android 12
            PendingIntent pendingIntent = PendingIntent.getActivity(
                context, 0, intent, PendingIntent.FLAG_IMMUTABLE);
            views.setOnClickPendingIntent(R.id.widget_layout, pendingIntent);

            // DIFFERENCE: Use new rounded corners API for Android 12
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                views.setViewOutlinePreference(R.id.widget_layout,
                    RemoteViews.OUTLINE_PREFERENCE_ROUNDED);
            }

            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}


<!-- widget_info.xml for Android 10 and Android 12 -->
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="1800000"
    android:initialLayout="@layout/example_widget"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen"
```

```
    android:description="@string/widget_description" <!-- DIFFERENCE: Add description for Android 12 -->
    android:targetCellWidth="2" <!-- DIFFERENCE: Specify target cell dimensions for Android 12 -->
    android:targetCellHeight="1">
</appwidget-provider>
```

25.通知模板和样式更新(notifications)

变更内容：

Android 12 对通知系统进行了视觉和功能上的更新，使其与 Material You 设计语言保持一致，包括更大的触摸目标、自适应颜色和改进的媒体控制。

参考链接：

https://developer.android.com/about/versions/12/features#notifications

示例代码：

```
// Notification implementation for Android 10 and Android 12
private void showNotification() {
    NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("Notification Title")
        .setContentText("Notification Content")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);

    // DIFFERENCE: Use richer styles in Android 12
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        builder.setStyle(new NotificationCompat.DecoratedCustomViewStyle());
        // On Android 12, notification visual style will automatically adapt to Material You design
    }

    NotificationManagerCompat notificationManager = NotificationManagerCompat.from(context);
    notificationManager.notify(notificationId, builder.build());
```

```
}
```

26.渲染流水线更改

变更内容：

Android 12 更改了渲染流水线，可能影响某些自定义 UI 实现。引入了新的帧率控制 API，提供了不同的兼容性模式。

参考链接：

https://developer.android.com/about/versions/12/behavior-changes-all?hl=zh-cn#rendering-pipeline

示例代码：

```
// Custom view implementation for Android 10 and Android 12
public class CustomView extends View {
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        // Drawing operations
        canvas.drawRect(...);

        // DIFFERENCE: Be aware of different rendering behaviors in Android 12
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            // Consider rendering optimizations or changes in behavior
        }
    }

    // Android 10: Use Choreographer for frame rate control
    private void setFrameRate() {
        Choreographer.getInstance().postFrameCallback(new Choreographer.FrameCallback() {
            @Override
            public void doFrame(long frameTimeNanos) {
                // Handle frame update
```

```
            invalidate();
            // Continue requesting next frame
            Choreographer.getInstance().postFrameCallback(this);
        }
    });
  }

  // Android 12: Use Surface.setFrameRate for controlling render frame rate
  @RequiresApi(api = Build.VERSION_CODES.S)
  private void setFrameRate(Surface surface) {
    // Set fixed frame rate mode
    surface.setFrameRate(60.0f,
            Surface.FRAME_RATE_COMPATIBILITY_FIXED_SOURCE,
            Surface.CHANGE_FRAME_RATE_ALWAYS);
  }
}
```