# ViewPager Fragment 懒加载 可见 总结 MD

发表于 2019-04-06 | 分类于 四大组件 , Frag | 字数统计: 2.5k 字 | 阅读时长 ≈ 10 分钟



| Markdown版本笔记 | 我的GitHub首页 | 我的博客 | 我的微信 | 我的邮箱 |
| --- | --- | --- | --- | --- |
| MyAndroidBlogs | baiqiantao | baiqiantao | bqt20094 | baiqiantao@sina.com |





## 目录

最小化后再回到前台

# Fragment 懒加载

## setUserVisibleHint 方法文档

```
@param isVisibleToUser true if this fragment's UI is currently visible to the user (default), false if it i
public void setUserVisibleHint(boolean isVisibleToUser) {}
```

Set a hint to the system about whether this fragment's UI is currently visible to the user. This hint defaults to true and is persistent across fragment instance state save and restore.

> 向系统提供有关此fragment的UI当前是否对用户可见的提示。此提示默认为true，**并且在fragment实例状态保存和还原期间保持不变。**

An app may set this to false to indicate that the fragment's UI is scrolled out of visibility or is otherwise not directly visible to the user. This may be used by the system to prioritize operations such as fragment lifecycle updates or loader ordering behavior.

> 应用程序可能将此设置为false，以指示fragment的UI滚动超出可见性，或者不直接对用户可见。系统可以使用它来确定诸如fragment生命周期更新或加载器排序行为之类的操作的优先级。

Note: This method may be called outside of the fragment lifecycle and thus has no ordering guarantees with regard to fragment lifecycle method calls.

> 注意：**此方法可以在fragment生命周期之外调用，因此对fragment生命周期方法调用没有排序保证。**

Note: Prior to Android N there was a platform bug that could cause setUserVisibleHint to bring a fragment up to the started state before its FragmentTransaction had been committed. As some apps relied on this behavior, it is preserved for apps that declare a targetSdkVersion of 23 or lower.

> 注意：在Android N之前，存在一个平台错误，可能导致setUserVisibleHint在FragmentTransaction提交之前将fragment提升到started状态。由于某些应用程序依赖于此行为，因此会为声明targetSdkVersion为23或更低的应用程序保留该行为。

## 懒加载实现思路

懒加载意思也就是当需要的时候才会去加载

那么，为什么Fragment需要懒加载呢，一般我们都会在onCreate()或者onCreateView()里去启动一些数据加载操作，比如从本地加载或者从服务器加载。大部分情况下，这样并不会出现什么问题，但是当你使用 `ViewPager + Fragment` 的时候，问题就来了。

ViewPager为了让滑动的时候可以有很好的用户的体验，也就是防止出现卡顿现象，因此它有一个缓存机制。默认情况下，ViewPager会提前创建好当前Fragment旁的两个Fragment。如果Fragment加载时比较耗时或者需要加载图片等占用大量内存的对象，这时就应该考虑想想是否该实现懒加载。也就是，当我 `打开` 哪个Fragment的时候，它才会去执行加载数据、解析数据，更新UI等操作。

本来Fragment的 onResume() 表示的是当前Fragment处于可见且可交互状态，但由于ViewPager的缓存机制，它已经失去了意义了，回

调Fragment的onResume()和其`对用户是否可见`不再具有关联性。

此时，我们可以使用Fragment的`setUserVisibleHint()`方法判断当前Fragment是否对用户可见，setUserVisibleHint 这个方法优于onCreate() 方法，它会通过 isVisibleToUser 告诉我们当前 Fragment 对用户是否可见，我们可以在可见的时候再进行网络加载等操作。

> 重要：懒加载一般和`setOffscreenPageLimit`配合使用：当用户没有点击某一个tab去查看相应的Fragment时，此Fragment懒加载，一旦用户查看过某一Fragment，则把此Fragment缓存下来，以后不再重复创建。

## 整个加载过程解析

下面两种方案本质上是完全一致的，其回调过程都符合以下规律。

**启动时(即默认显示第一个页面)**

- 第1次回调第1个Fragment的setUserVisibleHint：此时其`getUserVisibleHint`为false，所以不会触发其onLazyLoad()
- 第1次回调第2个Fragment的setUserVisibleHint：此时其`getUserVisibleHint`为false，所以不会触发其onLazyLoad()
- 第2次回调第1个Fragment的setUserVisibleHint：此时其`getUserVisibleHint`为true，但因为此时`isPrepared`为false，所以仍不会触发其onLazyLoad()

> 你没看错，上面的调用过程是确切的！

- 回调第1个Fragment的onCreate、onCreateView、onActivityCreated等一些列方法：此时其`getUserVisibleHint`为true，`isPrepared`被我们置为了true，`isLazyLoaded`仍是默认的false，所以会触发其onLazyLoad()
- 回调第2个Fragment的onCreate、onCreateView、onActivityCreated等一些列方法：此时其`getUserVisibleHint`为false，所以不会触发其onLazyLoad()，【但】其`isPrepared`被置为了true
- 后续第1个Fragment因为其`isLazyLoaded`被我们置为了true，所以永远不会再触发其onLazyLoad()

**从第一个页面滑到第二个页面时**

- 第1次回调第3个Fragment的setUserVisibleHint：此时其`getUserVisibleHint`为false，所以不会触发其onLazyLoad()
- 第2次回调第1个Fragment的setUserVisibleHint：此时其`getUserVisibleHint`为false，表明其不再对用户可见了
- 第2次回调第2个Fragment的setUserVisibleHint：此时其`getUserVisibleHint`为true，【并且】因为此前其`isPrepared`被置为了true，`isLazyLoaded`仍是默认的false，所以会触发其onLazyLoad()

> 你没看错，上面的调用过程也是确切的，不要怀疑！

- 同样，后续第2个Fragment因为其`isLazyLoaded`被我们置为了true，所以永远不会再触发其onLazyLoad()
- 回调第3个Fragment的onCreate、onCreateView、onActivityCreated等一些列方法：此时其`getUserVisibleHint`为false，所以不会触发其onLazyLoad()，【但】同样其`isPrepared`被置为了true

后面的逻辑就是类似的了。

## 实现方案

### BaseLazyFragment 版本一

```
public abstract class BaseLazyFragment extends Fragment {
```

```
    protected boolean isLazyLoaded;//懒加载过，保证只加载一次
    private boolean isPrepared;//防止没有inflateView以及findView之前操作View导致空指针异常

    @Override
    public void setUserVisibleHint(boolean isVisibleToUser) {
        super.setUserVisibleHint(isVisibleToUser);
        lazyLoad();
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        isPrepared = true;
        lazyLoad(); //必须调用，否则首次打开会显示空白界面
    }

    private void lazyLoad() {
        if (getUserVisibleHint() && isPrepared && !isLazyLoaded) {
            onLazyLoad();//调用懒加载
            isLazyLoaded = true;
        }
    }

    @UiThread
    public abstract void onLazyLoad();
}
```

## 实现类一

```
public class SimpleTabFragment extends BaseLazyFragment {
    private String name;
    private View rootView;
    public static final String ARGUMENT = "name";

    public static SimpleTabFragment newInstance(String name) {
        SimpleTabFragment fragment = new SimpleTabFragment();
        Bundle bundle = new Bundle();
        bundle.putString(ARGUMENT, name);
        fragment.setArguments(bundle);
        return fragment;
    }

    @Override
    public void onLazyLoad() {
        TextView tv = rootView.findViewById(R.id.tv);
        new Thread(() -> {
            SystemClock.sleep(500);//模拟耗时操作
            getActivity().runOnUiThread(() -> {
                tv.setText(name + ": " + new SimpleDateFormat("HH:mm:ss SSS", Locale.getDefault()).format(ne
                tv.setBackgroundColor(0xFF000000 + new Random().nextInt(0xFFFFFF));
            });
        }).start();
    }

    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Bundle bundle = getArguments();
        name = bundle != null ? bundle.getString(ARGUMENT) : "unknown";
    }

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInstanceSta
        rootView = inflater.inflate(R.layout.fragment_tab, container, false);
        return rootView;
    }
}
```

## BaseLazyFragment 版本二

```
public abstract class BaseLazyFragment extends Fragment {
    protected View rootView;
    protected boolean isLazyLoaded;//懒加载过，保证只加载一次
    private boolean isPrepared;//防止没有inflateView以及findView之前操作View导致空指针异常

    @Override
    public void setUserVisibleHint(boolean isVisibleToUser) {
        super.setUserVisibleHint(isVisibleToUser);
        lazyLoad();
    }

    /**
     * 不要重写onCreateView，请重写getRootView()和lazyLoad()
     */
    @Override
    public final View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInsta
        rootView = getRootView(inflater, container, savedInstanceState);
        isPrepared = true;
        lazyLoad(); //必须调用，否则首次打开会显示空白界面
        return rootView;
    }

    private void lazyLoad() {
        if (getUserVisibleHint() && isPrepared && !isLazyLoaded) {
            onLazyLoad();//调用懒加载
            isLazyLoaded = true;
        }
    }

    public abstract void onLazyLoad();

    protected abstract View getRootView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceS
}
```

## 实现类二

```
public class SimpleTabFragment extends BaseLazyFragment {
    private String name;
    public static final String ARGUMENT = "name";
```

```
    public static SimpleTabFragment newInstance(String name) {
        SimpleTabFragment fragment = new SimpleTabFragment();
        Bundle bundle = new Bundle();
        bundle.putString(ARGUMENT, name);
        fragment.setArguments(bundle);
        return fragment;
    }

    @Override
    public void onLazyLoad() {
        TextView tv = rootView.findViewById(R.id.tv);
        new Thread(() -> {
            SystemClock.sleep(500);//模拟耗时操作
            getActivity().runOnUiThread(() -> {
                tv.setText(name + ": " + new SimpleDateFormat("HH:mm:ss SSS", Locale.getDefault()).format(ne
                tv.setBackgroundColor(0xFF000000 + new Random().nextInt(0xFFFFFF));
            });
        }).start();
    }

    @Override
    protected View getRootView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_tab, container, false);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Bundle bundle = getArguments();
        name = bundle != null ? bundle.getString(ARGUMENT) : "unknown";
    }
}
```

## 回调过程与生命周期详细分析

### SimpleTabFragment

```
public class SimpleTabFragment extends Fragment {
    private String name;

    public static final String ARGUMENT = "name";

    public static SimpleTabFragment newInstance(String name) {
        SimpleTabFragment fragment = new SimpleTabFragment();
        Bundle bundle = new Bundle();
        bundle.putString(ARGUMENT, name);
        fragment.setArguments(bundle);
        return fragment;
    }

    @Override
    public void setUserVisibleHint(boolean isVisibleToUser) {
        super.setUserVisibleHint(isVisibleToUser);
```

```
        Log.i("bqt", " [--------setUserVisibleHint:" + hashCode() + "-" + name + "-" + isVisibleToUser + "--
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Bundle bundle = getArguments();
        name = bundle != null ? bundle.getString(ARGUMENT) : "unknown";
        Log.i("bqt", " [onCreate:" + name + "] ");
    }

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInstanceSta
        Log.i("bqt", " [----onCreateView" + name + "----] ");
        View view = inflater.inflate(R.layout.fragment_tab, container, false);
        TextView tv = view.findViewById(R.id.tv);
        tv.setText(name + ": " + new SimpleDateFormat("HH:mm:ss SSS", Locale.getDefault()).format(new Date()
        tv.setBackgroundColor(0xFF000000 + new Random().nextInt(0xFFFFFF));
        return view;
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.i("bqt", " [onResume:" + name + "-" + getUserVisibleHint() + "] ");
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        Log.i("bqt", " [----onDestroyView:" + name + "------] ");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.i("bqt", " [--------onDestroy:" + name + "--------] ");
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

## 启动后

```
[----------------FragmentPagerAdapter-getItem:0----------------]
[--------setUserVisibleHint:215908580-null-false--------]
[----------------FragmentPagerAdapter-getItem:1----------------]
[--------setUserVisibleHint:173705549-null-false--------]
[--------setUserVisibleHint:215908580-null-true--------]
```

```
【onCreate:微信】
【onCreate:通讯录】
【----onCreateView微信----】
【onResume:微信-true】
【----onCreateView通讯录----】
【onResume:通讯录-false】
```

> 刚开始setUserVisibleHint的参数为false，后来变为true
>
> 注意：两次回调都发生在onCreate之前

## 滑到第二个页面

```
【onPageSelected】1
【----------------FragmentPagerAdapter-getItem:2----------------】
【--------setUserVisibleHint:88702934-null-false--------】
【--------setUserVisibleHint:215908580-微信-false--------】
【--------setUserVisibleHint:173705549-通讯录-true--------】

【onCreate:发现】
【----onCreateView发现----】
【onResume:发现-false】
```

> 创建第三个页面时，目前只回调了一次setUserVisibleHint，且参数为false
>
> 第一个页面不可见了，同时第二个页面可见了，所以这两个Fragment的setUserVisibleHint各回掉了一次

## 滑到第一个页面

```
【onPageSelected】0
【--------setUserVisibleHint:173705549-通讯录-false--------】
【--------setUserVisibleHint:215908580-微信-true--------】
【----onDestroyView:发现------】
//【--------onDestroy:发现--------】//当使用FragmentStatePagerAdapter时会有这个回调
```

> 第三个页面可见状态内有发生变化，所以仍然没有回调其setUserVisibleHint方法

## 再次滑到第二个页面

```
【onPageSelected】1
//【----------------FragmentStatePagerAdapter-getItem:2----------------】//当使用*State*时会有这个回调
【--------setUserVisibleHint:88702934-发现-false--------】
【--------setUserVisibleHint:215908580-微信-false--------】
【--------setUserVisibleHint:173705549-通讯录-true--------】
//【onCreate:发现】//当使用FragmentStatePagerAdapter时会有这个回调
【----onCreateView发现----】
【onResume:发现-false】
```

> 注意这里第三个页面回调了一次setUserVisibleHint方法

## 最小化后再回到前台

**在第一个页面最小化后再回到前台**

```
【onResume:微信-true】
【onResume:通讯录-false】
```

**在第二个页面最小化后再回到前台**

```
【onResume:微信-false】
【onResume:通讯录-true】
【onResume:发现-false】
```

2019-3-25

打赏我一杯咖啡

打赏

‹ Intent 常用场景 FileProvider 拍照 裁剪 MD

TabLayout ViewPager Fragment 简介 案例 MD ›