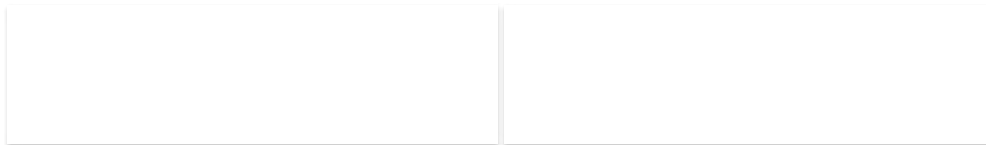




Производительность ListView

Последнее обновление: 29.03.2017



Хотя ListView является довольно мощным элементом для отображения данных, он все таки имеет некоторые ограничения. При определении сложного содержимого, сложной структуры ячеек в ListView может ухудшиться скроллинг, поскольку при пролистывании системе нужно будет выполнить множество вычислений. Однако существуют приемы и способы, которые позволят повысить производительность и постараться нивелировать возможный отрицательный эффект при подобных действиях.

Кэширование

Элемент ListView в Xamarin Forms опирается на нативные реализации этого элемента на Android, iOS, UWP, и на каждой из этих платформ реализации ListView имеют встроенные возможности по кэшированию ранее использованных строк. То есть, как правило, в память загружаются только те ячейки ListView, которые в текущий момент видны на экране. Соответственно контент загружается только в эти ячейки. Это позволяет не создавать тысячи объектов, которые номинально имеются в списке, тем самым уменьшая потребление памяти.

В Xamarin.Forms существуют две стратегии кэширования, которые описываются перечислением **ListViewCachingStrategy**. Для каждой стратегии в этом перечислении определено соответствующее значение:

- **RetainElement**
- **RecycleElement**

RetainElement

По умолчанию к ListView в Xamarin.Forms применяется значение **RetainElement**. Оно означает, что ListView будет создавать ячейку для каждого элемента в списке.

В каких случаях данная стратегия может быть предпочтительной:

- Когда каждая ячейка ListView имеет большое количество привязок (20-30 и более)
- Когда шаблон ячейки часто меняется
- Когда другая стратегия RecycleElement при тех же условиях показывает худшие результаты

RecycleElement

RecycleElement позволяет повторно использовать одни и те же ячейки, вместо их создания каждый раз, когда они попадают в зону видимости. Эта стратегия может быть предпочтительной в следующих случаях:

- Когда каждая ячейка ListView имеет небольшое количество привязок
- Когда свойство BindingContext ячейки определяет все ее данные
- Когда ячейки во многом аналогичны, а их шаблон остается неизменным

Для установки стратегии кэширования у ListView в XAML применяется атрибут **CachingStrategy**:

```
<ListView CachingStrategy="RecycleElement">
  <ListView.ItemTemplate>
    <DataTemplate>
      <ViewCell>
        ...
      </ViewCell>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

```

        </ViewCell>
    </DataTemplate>
</ListView.ItemTemplate>
</ListView>

```

В коде C# аналогом является передача в конструктор значения `ListViewCachingStrategy`:

```

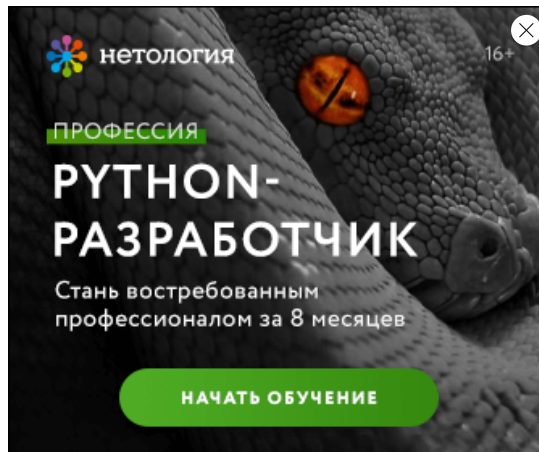
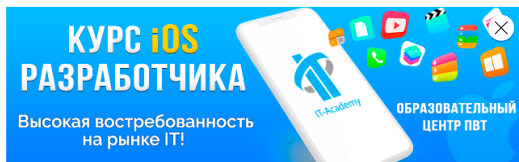
ListView listView = new ListView(ListViewCachingStrategy.RecycleElement);

```

Техники по оптимизации производительности

- Оптимальнее к свойству **ItemsSource** привязывать объект `IList<T>` вместо `IEnumerable<T>`, так как коллекции `IEnumerable<T>` не поддерживают произвольный доступ.
- Где возможно, используйте встроенные типы ячеек - `TextCell` или `ImageCell` вместо `ViewCell`.
- Используйте как можно меньше элементов.
- Если необходимо вывести данные различных типов, используйте элемент `Tableview` вместо `ListView`.
- Ограниченно используйте метод `Cell.ForceUpdateSize`, так как он снижает производительность.
- На Android избегайте установки видимости или цвета разделителя элементов в `ListView`, после инициализации `ListView`.
- Избегайте изменения ячеек, которые основаны на `BindingContext`
- Избегайте глубоковложенных иерархий элементов. Вместо подобных иерархий используйте `AbsoluteLayout` или `Grid`, чтобы уменьшить уровни вложенности элементов.
- Избегайте установки любого значения структуры `LayoutOptions`, кроме значения `Fill`, так как на при этом значении на вычисления тратятся наименьшие ресурсы.
- Избегайте размещения `ListView` внутри `ScrollView`, так как `ListView` имеет свой собственный скроллинг
- Если необходимо построение комплексной сложной ячейки, то лучше для этой цели лучше создать новый элемент с помощью [рендереров](#). Это поможет снизить отрицательный эффект от вычислений при скроллинге в списке.

`AbsoluteLayout` в ряде случаев может производить компоновку элементов вообще без вычислений, что делает данный элемент довольно производительным. Если по каким-то причинам `AbsoluteLayout` не может использоваться, стоит посмотреть в сторону `RelativeLayout`. При использовании `RelativeLayout` передавайте применяемым ограничениям (`Constraints`) прямые значения, вместо использования выражений.



[Назад](#) [Содержание](#) [Вперед](#)



CUBA Platform - Open Source Java Web Framework

Design UI and data model visually, develop in any Java IDE cuba-platform.com



0 Комментариев metanit.com

1 Войти ▾

 Рекомендовать  Поделиться

Лучшее в начале ▾



Начать обсуждение...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS 

Прокомментируйте первым.

ТАКЖЕ НА METANIT.COM

Kotlin | Введение в язык. Первая программа

1 комментарий • 2 месяца назад



Jonny Manowar — Есть мнение что на самом деле язык был создан и продвигается компанией JetBrains для продажи IDE, по сути не IDE пислась под язык а язык

Vue.js | Передача данных из дочернего компонента в родительский

1 комментарий • 4 месяца назад



SizeX — Не особо понятно как работает sync, описания вообще нет, обязательны ли update":"users в данном сокращении следовательно тоже неизвестно. А так все ...

Руководство по языку Go

2 комментария • месяц назад






Alexey — О, спасибо большое, что уделили внимание этому языку

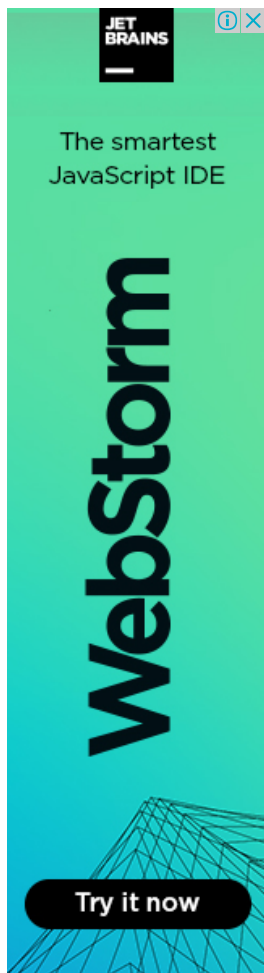
Vue.js | Навигация и ссылки

1 комментарий • 3 месяца назад



Delirium4Dude — Огромное спасибо тебе за твои труды! Одни из самых толковых мануалов в Рунете

 Подписаться  Добавь Disqus на свой сайт [Добавить Disqus](#) [Добавить](#)  Конфиденциальность



[Вконтакте](#) | [Twitter](#) | [Google+](#) | [Канал сайта на youtube](#) | [Помощь сайту](#)

Copyright © metanit.com, 2012-2017. Все права защищены.