Утечки памяти в WPF приложениях

Евгений Макаров

Cognitive Technologies

2018

План

- Предыстория
- Основные причины утечек памяти в WPF приложениях
- … и менее распространённые
- Как с ними бороться?
- Инструменты для поиска утечек
- Демо

Предыстория

С чем мы имели дело

- Суммарно 36 View, из них «протекало» больше половины
- При старте потребление памяти ~150 мб
- Зафиксированный предел потребления памяти ~1.5 гб
- После лечения ~300-350 мб в пике

Основные причины утечек памяти в WPF

- Incorrect Binding
- Event Handler Leak
- DispatcherTimer
- Links to objects in parent windows/views
- Many instances of resource dictionaries

Неправильный Binding

- Класс не реализует **INotifyPropertyChanged**
- Свойство привязки не является **DependencyProperty**
- System.ComponentModel.PropertyDesctiptor – **ValueChanged** event
- Объект привязки можно изменять
- Binding к коллекции без **INotifyCollectionChanged**

Как лечить неправильный Binding?

- **INotifyPropertyChanged** или **DependencyProperty**
- Binding Mode=“**OneTime**”
- Декоратор для **OneTime** Binding
- ObservableCollection или любая имплементация **INotifyCollectionChanged** для коллекции
- BindingOperations.ClearBinding => **IDisposable** pattern

Event Handler leak

- Проблема не только в WPF
- Экземплярный метод ссылается на класс, в котором находится
- Издатель живет дольше подписчика – риск получить утечки
- Static event – зло в чистом виде
- От лямбды нельзя отписаться

Как лечить Event Handler leak?

- Явная отписка от событий
- **IDisposable** pattern
- Behavior, Trigger и прочие «дополнения» к View
- Нельзя точно определить момент отписки – используем

  WeakEventManager/Weak event pattern

Weak event example

- Пример для реализаций **ICommand:**

  CommandManager.InvalidateRequery += OnInvalidateRequery;

- Weak Reference для обработчика события
- Может привести к преждевременному уничтожению обработчика события сборщиком мусора

DispatcherTimer

- Частный случай Event Handler Leak
- Dispatcher ссылается на коллекцию DispatcherTimers
- **Лечение** – остановка таймера + отписка от события Tick,

  IDisposable pattern
- Второй вариант – другой тип таймера + **Dispatcher.BeginInvoke**

Links to objects in parent windows

- View1 явно ссылается на свойство или подписывается на событие View2
- При закрытии View2 память не освободится
- Частный случай – **CommandBinding** (команда определена в View1, обработчик определен в View2)
- **Лечение** – **IDisposable** pattern

Many instances of ResourceDictionaries

- Не использовать ResourceDictionary = выстрелить себе в ногу
- Множество экземпляров ResourceDictionary во время жизни приложения
- **Лечение -** SharedResourceDictionary
- [Obsolete] В .Net Framework 3.5 – явный обход всех **MergedDictionaries** при старте приложения

Textbox undo

- Undo – включено по умолчанию
- UndoManager – хранит **весь** вводимый текст
- **Лечение** – ограничить или отключить операцию Undo

  IsUndoEnabled=“False”

  UndoLimit=“10”

Media effect resource leak

- Стиль определен в **ResourceDictionary**
- Стиль использует **ControlTemplate** с медиа эффектом (**DropShadowEffect**)
- Медиа эффект используется через **StaticResource**
- Срабатывание трех условий одновременно – джекпот! (Не всегда)

Как лечить Media effect resource leak?

- Необходимо следить за обновлением состояния незамороженного эффекта
- Атрибут **Freeze** для неизменяемых в рантайме эффектов

  “PresentationOptions:Freeze=True”
- Application **ResourceDictionary** -> View **ResourceDictionary**

x:Name

- Позволяет сослаться на control по **ElementName** или в code-behind
- Создает строгую глобальную ссылку на UI элемент
- Name VS x:Name
- **Лечение** - UnregisterName(“elementName”) при уничтожении View, **IDisposable** pattern
- Второй вариант – использовать свойство Name

Выводы

- Binding всегда с **INotifyPropertyChanged**
- Отписываемся от всех событий
- **IDisposable**
- Меньше используйте **static**

Как искать утечки памяти?

- Запускаем профилировщик и выбранное приложение
- Делаем снимок используемой памяти
- Следуем рекомендациям профилировщика
- Повторять до «полного» устранения утечек памяти

Чем искать утечки памяти в WPF?

	Цена	Удобство
Visual Studio profiler	**Free**	**-**
dotMemory	**299$**	**+**
dotNet Memory Profiler	**389-549$**	**+**
ANTS Memory Profiler	**612$**	**+**

Самое время для демо!

Контактные данные

- makarovevgeniy7@gmail.com
- @Talrandel 
- https://github.com/Talrandel/MemoryLeak - демо

Спасибо за внимание!