CHƯƠNG IX: HIỆU SUẤT HOẠT ĐỘNG TRONG ỨNG DỤNG SILVERLIGHT

1 Làm thế nào để chương trình của bạn chạy nhanh và ổn định

Để ứng dụng của bạn hoạt động ổn định với hiệu suất cao bạn cần phải chọn những phương án khả thi và tối ưu và tránh những sai lầm trong việc lựa chọn phương án. Chúng tôi đưa ra một số gợi ý giúp cho một ứng dụng chạy nhanh và hiệu quả dưới đây.

1.1 Thử nghiệm trên nhiều hệ điều hành và trình duyệt

Nếu bạn đang phát triển Silverlight dành cho các ứng dụng dựa trên nhiều hệ điều hành (ví dụ: máy Macintosh, Windows) và các trình duyệt (ví dụ: Internet Explorer, Mozilla Firefox, Apple Safari) bạn nên thường xuyên kiểm tra các ứng dụng của bạn trên các nền tảng và trình duyệt mà bạn đang nhắm tới. Những sự khác nhau trong nền tảng hoặc và cách thức hoạt động của trình duyệt, và các mã Silverlight lập trình ứng dụng có thể ảnh hưởng đến hiệu suất ứng dụng. Bạn nên kiểm tra kỹ lưỡng khi bạn tạo các ứng dụng có sử dụng nhúng các plug-in có sự kiểm trứng và minh bạch nguồn gốc plug-in.

1.2 Đặt EnableFrameRateCounter cho đúng trong thời gian phát triển

Hiệu suất khi render đối với các plug-in là khác nhau với các thông số và dữ liệu phức tạp. Chúng tôi khuyên bạn nên đặt EnableFrameCounter trong quá trình phát triển. Thiết lập này sẽ hiển thị các khung hình trên giây (fps: frame per second) của Silverlight trên thanh trạng thái trình duyệt, do đó bạn có thể tinh chỉnh các ứng dụng của bạn đúng với yêu cầu mà bạn đặt ra:

Fps: currentFramerate / maxFramerate

CurrentFramerate là số tỷ lệ khung hình trên giây hiện hành của ứng dụng dựa trên điều kiện môi trường của plug-in. maxFramerate là số tỉ lệ khung hình tối đa được cấu hình thông qua các tham số framerate initialization giá trị maxFramerate là giá trị tới hạn, nghĩa là bất cứ trường hợp nào chỉ số thực tế currentFramerate cũng sẽ thấp hơn maxFramerate.

Ví dụ khi bạn vào một trang HTML trên Silverlight maxFramerate mạc định là 24khung hình / giây

1.3 Sử dụng Transparent Background

Sử dụng Transparent background có thể sẽ hữu ích với ví dụ sau: khi bạn muốn hiển thị đồng thời giao thoa giữa các lớp đối tượng đồ họa chồng lớp lên nhau như image, shape.

Tuy nhiên tránh việc lạm dụng Transparent background ở bất cứ đâu, nó sẽ làm ảnh hưởng đến hiệu suất hoạt động của ứng dụng.

1.4 Tránh việc sử dụng các kịch bản làm biến đổi kích cỡ font của Text

Thay đổi kích cỡ của Text sẽ ảnh hưởng tiêu tốn khá nhiều tài nguyên hệ thống, bởi vì Silverlight sử dụng hinting để làm mịn văn bản khi render text. Nếu bạn biến đổi text size bởi transform hoặc thuộc tính FontSize Silverlight sẽ làm mịn lại toàn bộ text cho mỗi frame, việc dó sẽ làm tiêu tốn tài nguyên.

Nếu ứng dụng của bạn đòi hỏi phải thay đổi quy mô văn bản lớn, sẽ tốt hơn nếu sử dụng đồ họa vector làm đại diện các văn bản.

1.5 Tránh sử dụng chế độ Windowless

Chỉ nên đặt thuộc tính Windowless khi cần thiết. Hiệu suất sẽ bị ảnh hưởng nghiêm trọng khi ở chế độ Windowless. Do vậy khuyến cáo các bạn không nên sử dụng chế độ này.

1.6 Sử dụng Visibility thay cho việc sử dụng Opacity trong rất nhiều trường hợp không cần đến sự có mặt của Opacity

Nếu bạn đơn giản chỉ muốn thực hiện tắt hiển thị một đối tượng thì hoàn toàn không nên sử dụng thuộc tính opacity, trong trường hợp này tôi khuyên bạn nên sử dụng thuộc tính Visibitily. Opacity sử dụng chi phí tài nguyên cao hơn bởi vì đối tượng này vẫn sử dụng các kỹ thuật Rendered. Sử dụng Visibility để tránh việc lãng phí tài nguyên.

1.7 Silverlight sử dụng Multi-Core trong Rendering và Media

Silverlight mang lại ưu điểm của Multi-core cho Render đồ họa và Media. Bởi vậy các ứng dụng Silverlight của bạn sẽ chạy nhanh hơn trên hệ thống Multi-core(đa lõi).

1.8 Trong chế độ Full-Screen, ẩn những đối tượng không sử dụng

Khi ứng dụng của bạn ở chế độ Full-Screen, ẩn các đối tượng không được Render hoặc ngắt kết nối chúng trong cây. Bạn có thể ẩn một đối tượng bằng tùy chỉnh thuộc tính Visibility bằng Collapsed.

1.9 Tránh sử dụng thuộc tính Width và Height đối với đối tượng MediaElement

Tránh việc thiết lập Width và Height của một đối tượng MediaElement. Thay vào đó cho phép các MediaElement hiển thị kích cỡ tự nhiên. Nếu bạn cần thay đổi kích cỡ màn hình hiển thị của các Element, cách tốt nhất là mã hóa lại file Media với kích cỡ mong muốn bằng các công cụ khác.

1.10 Tránh sử dụng thuộc tính Width và Height đối với đối tượng Path

Tránh thiết lập thuộc tính Width, Height cho đối tượng Path. Thiết lập các thuộc tính sẽ bổ sung stretching tự động nới rộng phạm vi ảnh hưởng đến hiệu suất. Thay vào đó căn cứ vào các tọa đọ rõ ràng của các đối tượng Path điều khiển hình dạng và vị trí của nó, khi đó Width và Height sẽ tự động được set.

1.11 Nguy cơ đổ vỡ khi CPU sử ly cường độ lớn công việc

Khi đang thực thi các mã code (C# hay VisualBasic) các plug-in ngừng vẽ. Thông thường điều này không phải là vấn đề khi ta thực hiện những công việc tối thiểu mà dễ dàng kiểm soát được. Tuy nhiên nếu ứng dụng có quy mô tương đối và có sử dụng lập trình thread, chúng tôi khuyên bạn nên chia nhỏ công việc và các tác vụ. Điều này sẽ cho phép ứng dụng Render theo kịp với tỷ lệ Frame mong muốn.

1.12 Nguy cơ đổ vỡ đối với ứng dụng có những Package lớn

Trong một số trường hợp Silverlight plug-in không phải là thành công khi chạy các ứng dụng có tập tin (.Xap) lớn, bạn cần cân nhắc tập chung một số nguồn lực, tập hợp thư viện vào modul riêng biệt và chỉ tải về theo yêu cầu nhằm tối ưu hóa hệ thống và nguồn lực.

1.13 Sử dụng Double.ToString(CultureInfo.InvariantCulture) hiệu quả hơn Double.ToString()

Phương thức Double.ToString(IFormatProvider) cung cấp giá trị CultureInfo.InvariantCulture tối ưu hóa hiệu suất. Tổng quan thì phương thức Double.ToString(CultureInfo.InvariantCulture) thực thi tốt nhất khi bạn không muốn hiển thị dữ liệu tới người dung hoặc dùng cho việc so sánh String.

Nếu ứng dụng của bạn hiển thị các số tới người dùng và bạn muốn hiển thị chúng chính xác, bạn nên sử dụng phương thức Double.ToString(IFormatProvider) với giá trị CultureInfo.CurrentCulture. Với những yếu tố hữu ích trên bạn nên thực hiện việc chuyển đổi.

2 Sử dụng BackGroundWorker

2.1 Bắt đầu với việc tạo một BackGroundWorker

C#

```
BackgroundWorker bw = new BackgroundWorker();

// Xác các thuộc tính để hệ thống background cho phép hủy bỏ và báo cáo tiến trình

bw.WorkerSupportsCancellation = true;
bw.WorkerReportsProgress = true;
```

2.2 Tạo một Event handler cho background worker bởi DoWork event

DoWork event handler là nơi mà bạn chạy hệ thống trên nền thread. Bất kỳ một thay đổi nào của hệ thống nền thông qua đối số của DoWorkEventArgs đối tượng đó sẽ được thông qua với sự kiện handler.

Để thông báo tiến trình quay trở lại gọi tới hàm ReportProgress và thông qua nó hoàn thành một giá trị từ 0 đến 100. ReportProgress gây lên một sự kiện ProgressChanged mà bạn có thể xử lý riêng biệt.

Chú ý: nếu báo cáo tiến trình WorkerReportsProgress của BackGroundWorker không được đặt là True mà bạn gọi thủ tục ReportProgress, một ngoại lệ sẽ xảy ra.

Để xác định xem có một yêu cầu nào đó đang chờ thực thi yêu cầu hủy bỏ background ngầm, hãy kiểm tra thuộc tính CancellationPending của BackgroundWorker. Nếu thuộc tính đó trả về là True thì thủ tục CancelAsync đã được gọi. Đối tượng BackGroundWorker được hủy bỏ và hệ thống sẽ dừng lại.

Để chuyển dữ liệu quay trở lại quá trình xử lý, thiết lập thuộc tính cho DoWorkEventArgs của đối tượng được thông qua với sự kiện handler. Giá trị này có thể được đọc khi mà RunWorkerCompleted sự kiện được gây ra khi kết thúc hệ thống.

C#

```
private void bw_DoWork(object sender, DoWorkEventArgs e)
{
   BackgroundWorker worker = sender as BackgroundWorker;

   for (int i = 1; (i <= 10); i++)
   {
      if ((worker.CancellationPending == true))
      {
         e.Cancel = true;
         break;
      }
      else
      {
         // Perform a time consuming operation and report progress.
         System.Threading.Thread.Sleep(500);
         worker.ReportProgress((i * 10));
      }
   }
}
```

2.3 Tạo một event handler cho sự kiện ProgressChanged của backgroundworker

Trong sự kiện ProgressChanged thêm mã xử lý tiến trình chẳng hạn như cập nhật giao diện người dùng.

Để xác định bao nhiêu tỉ lệ phần trăm các hoạt động được hoàn thành, hãy kiểm tra thuộc tính ProgressPercentage thông qua sự kiện ProgressChangedEventArgs.

C#

```
private void bw_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
   this.tbProgress.Text = (e.ProgressPercentage.ToString() + "%");
}
```

2.4 Tạo một sự kiện cho RunWorkerCompleted

Sự kiện RunWorkerCompleted được gây ra khi tiến trình backgroundworker hoàn thành. Tùy thuộc vào việc tiến trình background sẽ đưa ra các trạng thái như hoàn thành , lỗi, hủy mà update giao diện người dùng cho phù hợp.

Để xác định một lỗi xảy ra, hãy kiểm tra các lỗi từ RunWorkerCompletedEventArgs thông qua sự kiện. Nếu một lỗi xảy ra thuộc tính này sẽ bao gồm những thông tin về trường hợp ngoại lệ.

Nếu hệ thống hoạt động không cho phép hủy bỏ và bạn muốn kiểm tra xem các hoạt động của hệ thống đã được hủy bỏ hay không sự kiện được thông qua RunWorkerCompletedEventArgs. Nếu là True thì thủ tục CancelAsync đã được gọi.

C#

```
private void bw_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
   if ((e.Cancelled == true))
   {
      this.tbProgress.Text = "Canceled!";
   }

   else if (!(e.Error == null))
     {
        this.tbProgress.Text = ("Error: " + e.Error.Message);
```

```
    }

  else
    {
        this.tbProgress.Text = "Done!";
    }
}
```

2.5 Bổ xung sự kiện vào BackGroundWorker

Ví dụ sau đây cho thấy làm thế nào để bổ xung thêm sự kiện vào DoWork, ProgressChanged, và các sự kiện RunWorkerCompleted.

```
bw.DoWork += new DoWorkEventHandler(bw_DoWork);
bw.ProgressChanged += new ProgressChangedEventHandler(bw_ProgressChanged);
bw.RunWorkerCompleted += new RunWorkedCompletedEventHandler(bw_RunWorkerCompleted);
```

2.6 Bắt đầu chạy background gọi bởi thủ tục RunWorkerAsync.

```
private void buttonStart_Click(object sender, RoutedEventArgs e)
{
  if (bw.IsBusy != true)
  {
     bw.RunWorkerAsync();
  }
}
```

2.7 Hủy bỏ hoạt động của background gọi bởi thủ tục CancelAsync.

```
private void buttonCancel_Click(object sender, RoutedEventArgs e)
{
   if (bw.WorkerSupportsCancellation == true)
   {
      bw.CancelAsync();
   }
}
```