Windows Phone Mango编程实践

***Windows Phone Mango Programming Practice***

第一篇 基础篇

# 传感器和服务

## 认知传感器

Windows Phone Mango传感器 (Sensor)包括：重力传感器 (G-Sensor)、数字罗盘、趋近传感器、指南针传感器、陀螺仪传感器，以及移动（Motion ）传感器，以及环境光线传感器。传感器可以视为一种特殊的输入设备，使用者可以不需要特别执行任何输入的动作，程序就可以依据传感器输入的数据做出反应。

* 利用数字电子罗盘获得与方向有关的数据；
* 利用光线传感器感应外界光线的强弱，自动调节手机屏幕的亮度；
* 利用接近传感器判断是否贴近使用者的脸部，避免误触影响通话而自动锁定屏幕的操作；
* 利用重力传感器感应智能手机运动的方向，并作为调整智能手机的屏幕显示方向的依据，将重力传感器的数据传送给程序会获得更多的应用。比如极品飞车游戏中控制汽车的转弯方向；在弹珠游戏程序中改变弹珠的滚动方向的动作。此外，重力传感器还能在GPS (全球定位系统)中发挥作用，当智能手机接收不到卫星讯号时，利用智能手机的运动方向推断用户的位置。

传感器在游戏程序的应用很广泛，例如使用者挥动手臂的动作可以模拟使用球棒挥击棒球的动作，模拟掷出保龄球的动作，拍击网球、羽毛球、或乒乓球的动作，模拟丢掷骰子，甚至可以模拟游戏者身体移动的方向，跳跃的高度与距离，让使用者融入游戏的场景，达到与游戏真实互动的感觉，而不是像传统的游戏，游戏的使用者感觉较像局外人。

Windows Phone智能手机支持完整的传感器功能，例如利用传感器控制的俄罗斯方块游戏，能够在不靠键盘输入的状况下利用倾斜智能手机的方式控制方块掉落的位置。

传感器给游戏用户哪些震撼的体验，在下面的章节我们细细分解。

## 重力加速传感

### 应用重力加速传感器的体感游戏设计

Windows Phone提供的重力传感器利用量测重力的原理判断智能手机移动的方向，允许使用者利用摇摆智能手机的方式控制游戏的执行，其原理和汽车的安全气囊相同，在侦测到汽车快速减速的时候立刻充气以保护驾驶人与乘客不会受伤。

要使用重力传感器当做游戏程序的输入，以XNA为基础的游戏程序可以利用Accelerometer 类别提供的功能启用/停用重力加速器，取得重力加速器的状态，以及处理重力加速器引发的事件。有关 Accelerometer 类别常用的属性可以参考表4-1 Accelerometer 类别常用的属性的说明：

|  |  |
| --- | --- |
| 属性名称 | 说明 |
| State | 管理重力加速器状态的属性，其型态为 SensorState 列举型态。有关 SensorState 列举型态合法的内容值可以参考表4 的说明。 |

表4-1 Accelerometer 类别常用的属性

Accelerometer 类别常用的方法可以参考表4-2 Accelerometer 类别常用的方法的说明：

|  |  |
| --- | --- |
| 方法名称 | 说明 |
| Start | 开始从重力加速器读取数据。 |
| Stop | 结束从重力加速器读取数据。 |

表4-2 Accelerometer 类别常用的方法

Accelerometer 类别常用的事件可以参考表4-3 Accelerometer 类别常用的事件的说明：

|  |  |
| --- | --- |
| 事件名称 | 说明 |
| ReadingChanged | 当重力加速器读取到数据时会引发的事件。 |

表4-3 Accelerometer 类别常用的事件

处理 ReadingChanged 事件的事件为 AccelerometerReadingEventArgs ，其 X、Y、Z属性的内容值代表智能手机在 X 轴、Y 轴、和 Z 轴的加速方向，而不是三维空间的坐标，其单位为重力单位，即G (1G = 9.81 m/s2)。除了 X、Y、与 Z 三个属性以外，还有一个名称为 Timestamp 的属性，负责记录重力加速器读取数据的时间点。

请注意当智能型手机放在平坦的桌面上，而且正面朝上的时候，AccelerometerReadingEventArgs 类别的 Z 字段的内容值会是 -1.0，表示 Z 轴承受 -1G 的重力，而当智能型手机放在平坦的桌面上，而且正面朝下的时候，AccelerometerReadingEventArgs 类别的 Z 字段的内容值就会是 +1.0，表示 Z 轴承受 1G 的重力。

透过 Accelerometer 类别的 State 属性取得的重力加速器状态是 SensorState 列举型态的数据，其合法的内容值请参考表4-4 SensorState 列举型态合法的内容值的说明：

|  |  |
| --- | --- |
| 内容值名称 | 说明 |
| NotSupported | 未支持重力加速器。 |
| Ready | 重力加速器处于可以处理数据的状态。 |
| Initializing | 重力加速器正在初始化。 |
| NoData | 未支持重力加速器。 |
| NoPermissions | 呼叫者没有权限取用重力加速器接收到的数据。 |
| Disabled | 重力加速器处于禁用的状态。 |

表4-4 SensorState 列举型态合法的内容值

要使用重力加速器判断智能手机加速的方向，首先您必须使用鼠标的右键点中 [Solution Explorer] 窗口中的项目名称，从出现的菜单选择 **[Add Reference]** 功能，然后于出现的窗口中选择名称为 Microsoft.Devices.Sensors 的组件，如图4-1 参考名称为 Microsoft.Devices.Sensors 的组件的画面所示：

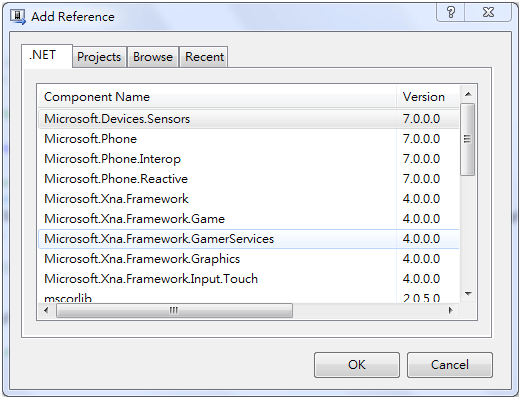


图4-1 参考名称为 Microsoft.Devices.Sensors 的组件的画面

做好之后请按下 [OK] 键完成参考组件的动作。接下来请于Game1类别中加入以下的变量宣告，负责管理重力加速器装置：

Accelerometer gSensor; //管理重力感应的变量

然后于 Initialize 方法执行建立 Accelerometer 类别的对象的动作，为 Accelerometer 类别的对象的 ReadingChanged 事件制作事件处理程序，并呼叫 Accelerometer 类别的 Start 方法，开始接收从重力加速器输入的数据，如下：

protected override void Initialize()

{

gSensor = new Accelerometer(); //新建Accelerometer对象

gSensor.ReadingChanged += new

EventHandler<AccelerometerReadingEventArgs>(

gSensor\_ReadingChanged); //处理Accelerometer对象的 ReadingChanged 事件

gSensor.Start(); //启动重力感应

base.Initialize();

}

应用程序只要于 Accelerometer 类别的对象的 ReadingChanged 事件的事件处理程序中利用型态为 AccelerometerReadingEventArgs 类别，名称为 e 的参数，就可以得知 Windows Phone装置加速的方向，如下：

void gSensor\_ReadingChanged(object sender, AccelerometerReadingEventArgs e)

{

//取用 e.X, e.Y, e.Z

}

### 动手实践——Silverlight获取重力加速度感应数据

以AccelerometerSample实例讲解Silvelight如何获取重力加速度感应数据。

1. 新建工程

打开Visual Studio，新建Silverlight for Windows Phone工程，名称为AccelerometerSample，如图。

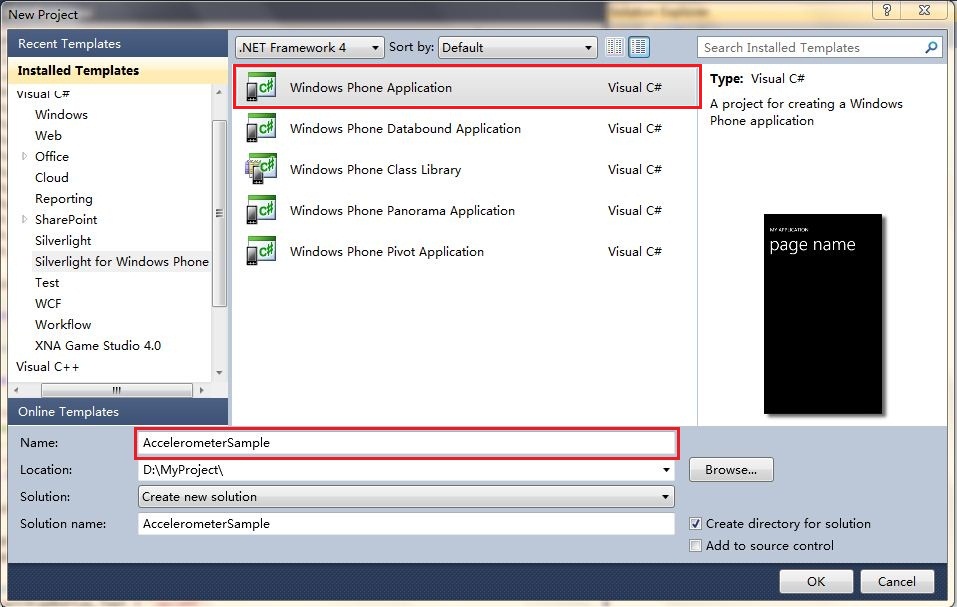


图4-2 新建AccelerometerSample工程

1. 添加引用

使用鼠标的右键点中 [Solution Explorer] 窗口中的项目名称AccelerometerSample，从出现的菜单选择 **[Add Reference]** 功能，如图。

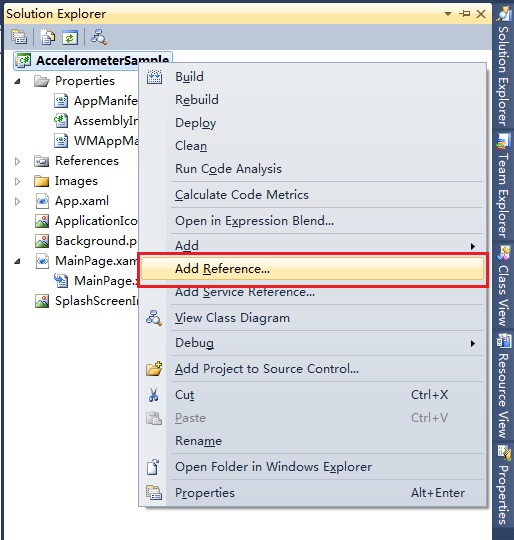


图4-3 添加引用

选择名称为 Microsoft.Devices.Sensors 的组件，如图：

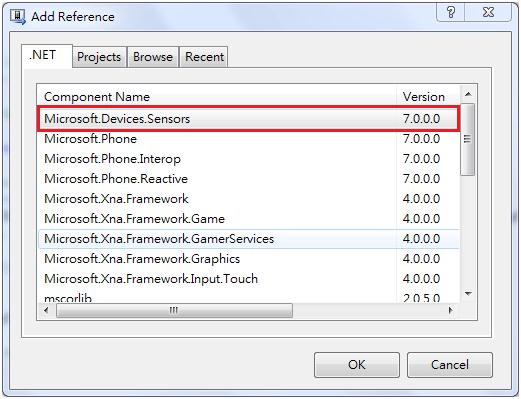


图4-4 Microsoft.Devices.Sensors 的组件

1. 添加Sensors命名空间

在MainPage.xaml.cs中添加Sensors的命名空间。

Silverlight Project: AccelerometerSample File: MainPage.xaml.cs

using Microsoft.Devices.Sensors;

1. 声明Accelerometer类的变量

在MainPage类中添加Accelerometer的变量。在MainPage类的生命周期内此变量都有效。

Silverlight Project: AccelerometerSample File: MainPage.xaml.cs

Accelerometer accelerometer;

通过使用无参数的构造函数中获得重力加速感应类的新实例。在此示例中，以便用户可以启动和停止重力加速感应，重力加速感应类初始化被放置在按钮的click事件处理程序中。

1. 重力加速感应类的实例化

在此例中，用户通过按钮事件启动和停止重力加速传感功能。

Silverlight Project: AccelerometerSample File: MainPage.xaml.cs

void startStopButton\_Click(object sender, EventArgs e)

{

// If the accelerometer is null, it is initialized and started

if (accelerometer == null)

{

// Instantiate the accelerometer sensor object

accelerometer = new Accelerometer();

增加ReadingChanged事件处理程序。

// Add an event handler for the ReadingChanged event.

accelerometer.ReadingChanged += new EventHandler<AccelerometerReadingEventArgs>(accelerometer\_ReadingChanged);

调用 Start 方法启动重力加速传感器。Start 方法启动加速度传感器会引发异常，所以在 try 块中调用它。

// The Start method could throw and exception, so use a try block

try

{

statusTextBlock.Text = "starting accelerometer";

accelerometer.Start();

}

catch (AccelerometerFailedException exception)

{

statusTextBlock.Text = "error starting accelerometer";

}

}

else

{

// if the accelerometer is not null, call Stop

try

{

accelerometer.Stop();

accelerometer = null;

statusTextBlock.Text = "accelerometer stopped";

}

catch (AccelerometerFailedException exception)

{

statusTextBlock.Text = "error stopping accelerometer";

}

}

}

1. 实现ReadingChanged 事件处理程序

每当有新的重力加速传感器数据产生时触发此事件。因为事件处理程序是被另一个线程调用的，需要用到System.Windows.Deployment名字空间的Dispatcher类的BeginInvoke方法调用。事件处理程序MyReadingChanged将AccelerometerReadingEventArgs 对象作为输入参数。

Silverlight Project: AccelerometerSample File: MainPage.xaml.cs

void accelerometer\_ReadingChanged(object sender, AccelerometerReadingEventArgs e)

{

Deployment.Current.Dispatcher.BeginInvoke(() => MyReadingChanged(e));

}

/// <summary>

/// Method for handling the ReadingChanged event on the UI thread.

/// This sample just displays the reading value.

/// </summary>

/// <param name="e"></param>

void MyReadingChanged(AccelerometerReadingEventArgs e)

{

if (accelerometer != null)

{

statusTextBlock.Text = accelerometer.State.ToString();

XTextBlock.Text = e.X.ToString("0.00");

YTextBlock.Text = e.Y.ToString("0.00");

ZTextBlock.Text = e.Z.ToString("0.00");

}

}

1. 运行与调试

按F5运行应用程序，或者点击Start Debugging按钮运行，如图4-5 Start Debugging。



图4-5 Start Debugging

使用重力加速度的模拟器测试。

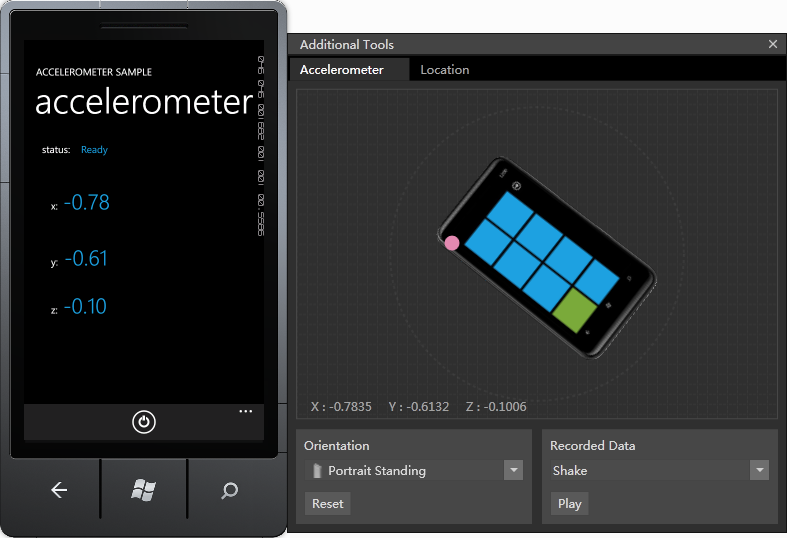


图4-6 运行结果

### 动手实践——XNA中使用传感器移动物体

了解Silverlight获取重力加速度感应数据的方法之后，接下来我们就要设计一个能够支持体感控制的 XNA 游戏，让使用者能够以倾斜 Windows Phone智能手机的方式移动物体。

1. 新建Windows Phone Game (4.0)工程

首先请启动 Visual Studio 2010 Express for Windows Phone，建立一个 [Windows Phone Game (4.0)]的项目，工程名称为AccelerometerEnabledGame，然后加入游戏程序显示的图片到 Content Pipeline 项目中。

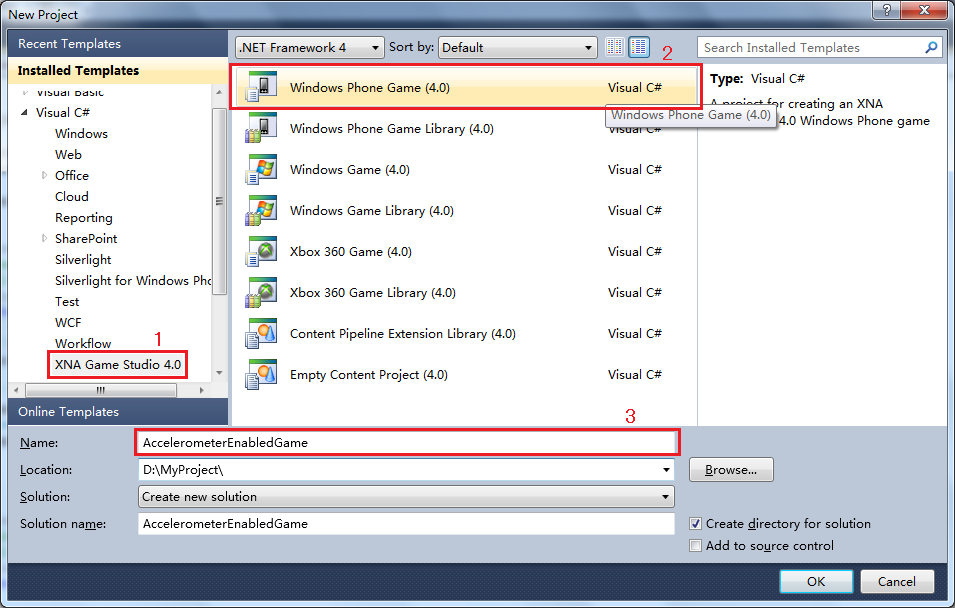


图4-7 新建

1. 声明管理重力加速传感器的变量

Game1类中加入以下的变量声明，管理显示的图片和显示位置，以及管理重力加速器和记载加速度方向的变量：

XNA Project: AccelerometerEnabledGame File: Game1.cs

Texture2D Logo;//显示的图片

Vector2 LogoPosition;//显示图片的位置

Accelerometer gSensor;//重力加速传感器

Vector2 LogoVelocity; //重力加速传感器方向

1. 设定游戏窗口的属性

在Game1类的构造函数中，设定游戏窗口的高度与宽度的工作，设定游戏窗口高度与宽度。Game1类的构造函数如下：

XNA Project: AccelerometerEnabledGame File: Game1.cs

public Game1()

{

graphics = new GraphicsDeviceManager(this);

Content.RootDirectory = "Content";

graphics.SupportedOrientations = DisplayOrientation.Portrait |

DisplayOrientation.LandscapeLeft |

DisplayOrientation.LandscapeRight;

graphics.PreferredBackBufferWidth = 480;//游戏视窗的宽度

graphics.PreferredBackBufferHeight = 800;//游戏视窗的高度

// Frame rate is 30 fps by default for Windows Phone.

TargetElapsedTime = TimeSpan.FromTicks(333333);

}

1. 启动重力加速感应器

Game1类的初始化（Initialize）方法中启用重力加速传感器，程序利用传感器控制游戏显示的对象，初始化（Initialize）的方法如下：

XNA Project: AccelerometerEnabledGame File: Game1.cs

/// <summary>

/// Allows the game to perform any initialization it needs to before starting to run.

/// This is where it can query for any required services and load any non-graphic

/// related content. Calling base.Initialize will enumerate through any components

/// and initialize them as well.

/// </summary>

protected override void Initialize()

{

// TODO: Add your initialization logic here

gSensor = new Accelerometer();//建立Accelerometer对象

gSensor.ReadingChanged += new EventHandler<AccelerometerReadingEventArgs>(gSensor\_ReadingChanged);// ReadingChanged事件的处理程序

gSensor.Start();//启动重力加速传感器

base.Initialize();

}

1. 事件处理程序

上述的程序代码中名称为 gSensor\_ReadingChanged 的事件处理程序，负责处理 Accelerometer 类的对象 (即重力加速传感器) 引发的 ReadingChanged 事件。在 Game1类中加入以下的方法：

XNA Project: AccelerometerEnabledGame File: Game1.cs

void gSensor\_ReadingChanged(object sender, AccelerometerReadingEventArgs e)

{

LogoVelocity.X += (float)e.X;//X轴加速度

LogoVelocity.Y += -(float)e.Y;//Y轴加速度

LogoPosition += LogoVelocity;//将加速度的值赋值给图片的位置控制变量

}

1. 加载游戏显示的图片

Game1 类别的 LoadContent 方法，加载游戏程序欲显示的图片，顺便设定图片默认的显示位置，做好的 LoadContent 方法如下：

XNA Project: AccelerometerEnabledGame File: Game1.cs

/// <summary>

/// LoadContent will be called once per game and is the place to load

/// all of your content.

/// </summary>

protected override void LoadContent()

{

// Create a new SpriteBatch, which can be used to draw textures.

spriteBatch = new SpriteBatch(GraphicsDevice);

// TODO: use this.Content to load your game content here

Logo = Content.Load<Texture2D>("xna");//载入图片

Viewport viewport = graphics.GraphicsDevice.Viewport;//取得视区

LogoPosition = new Vector2(

(viewport.Width - Logo.Width) / 2,

(viewport.Height - Logo.Height) / 2);//设定图片显示的位置在游戏视窗的中央位置

}

1. 关闭重力加速传感器

在初始化的阶段启用了重力加速器传感器，因此在游戏程序结束之前关闭重力加速器。在Game1类的UnloadContent方法中呼叫Accelerometer类别的对象的Stop方法，关闭重力加速传感器。

XNA Project: AccelerometerEnabledGame File: Game1.cs

/// <summary>

/// UnloadContent will be called once per game and is the place to unload

/// all content.

/// </summary>

protected override void UnloadContent()

{

// TODO: Unload any non ContentManager content here

gSensor.Stop();//关闭重力加速传感器

}

1. 在Update方法中改变物体显示的位置

游戏程序允许用户以倾斜的方式让用户移动游戏程序显示的对象，要实现此功能就需要在 Game1 类的 Update 方法中改变物体显示的位置，当游戏程序显示的物体碰撞到游戏窗口的四个边界的时候，停止移动物体的动作。在物体碰撞到游戏窗口的四个边界时，设定对象显示的位置，并将重力加速度的内容值设定为 0：

XNA Project: AccelerometerEnabledGame File: Game1.cs

/// <summary>

/// Allows the game to run logic such as updating the world,

/// checking for collisions, gathering input, and playing audio.

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Update(GameTime gameTime)

{

// Allows the game to exit

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

this.Exit();

// TODO: Add your update logic here

Viewport viewport = graphics.GraphicsDevice.Viewport;//获取游戏窗体的视区

if (LogoPosition.X < 0)

{

LogoPosition.X = 0;

LogoVelocity.X = 0;

}

else if (LogoPosition.X > viewport.Width - Logo.Width)

{

LogoPosition.X = viewport.Width - Logo.Width;

LogoVelocity.X = 0;

}

// keep the sprite on the screen - clamp Y

if (LogoPosition.Y < 0)

{

LogoPosition.Y = 0;

LogoVelocity.Y = 0;

}

else if (LogoPosition.Y > viewport.Height - Logo.Height)

{

LogoPosition.Y = viewport.Height - Logo.Height;

LogoVelocity.Y = 0;

}

base.Update(gameTime);

}

在Update方法中，程序代码会在游戏程序显示的物体碰撞到窗口的四个边界的时候将物体停放在窗口的边界。如果想让游戏更逼真更生动，让物体碰撞窗口的边界时产生反弹的效果，可以将加速度的方向设定为负值，再依据模拟的摩擦系数进行递减，让游戏程序显示的物体呈现自然的反弹效果。

1. 重载Draw方法绘制游戏画面

重载Game1 类的 Draw 方法绘制游戏画面。

XNA Project: AccelerometerEnabledGame File: Game1.cs

/// <summary>

/// This is called when the game should draw itself.

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(Color.CornflowerBlue);

// TODO: Add your drawing code here

spriteBatch.Begin();//宣告绘制动作开始

spriteBatch.Draw(Logo, LogoPosition, Color.White);// Adds a sprite to a batch of sprites for rendering using the specified texture, destination rectangle, and color.

spriteBatch.End();//宣告绘制动作结束

base.Draw(gameTime);

}

请将开发好的游戏程序部署到Windows Phone智能手机执行,画面如图4-8 使用重力加速器功能的游戏程序执行的情形。

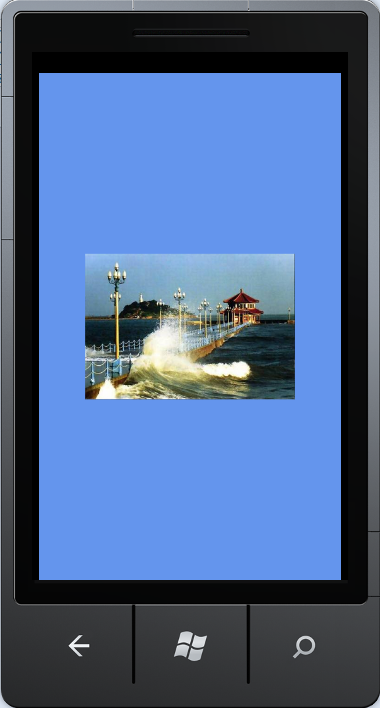
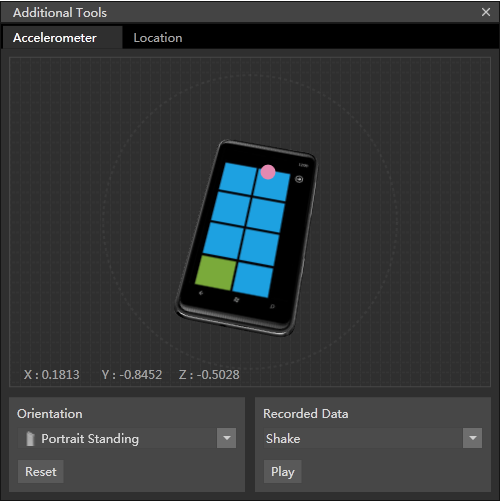
 

图4-8 使用重力加速器功能的游戏程序执行的情形

## 地理位置服务

使用地理位置服务(Location Service)能够开发具备位置感知功能的应用程序。确定位置的最准确的方法获取GPS（信号从卫星全球定位系统）的地理位置信息。但是GPS的速度慢且耗电，除此以外还可以从Wi-Fi和移动网络基站获得位置信息，虽然耗电减少但准确度也相应降低。

地理位置服务(Location Service)能够从一个或者多个数据源计算出位置信息，通过统一的事件驱动的托管代码的接口提供给应用程序。

有效的平衡电量的消耗与位置信息的准确性是设计人员需要首先考虑的事情。权衡在世界中无处不在，软件中也不例外。

### 动手实践——读取地理位置信息

1. 引用位置服务

使用地位位置服务的API，首先要添加引用System.Device.dll。

1. 新建[Silverlight for Windows Phone]工程，设定工程名称为”LocationServiceSample”，如图所示。

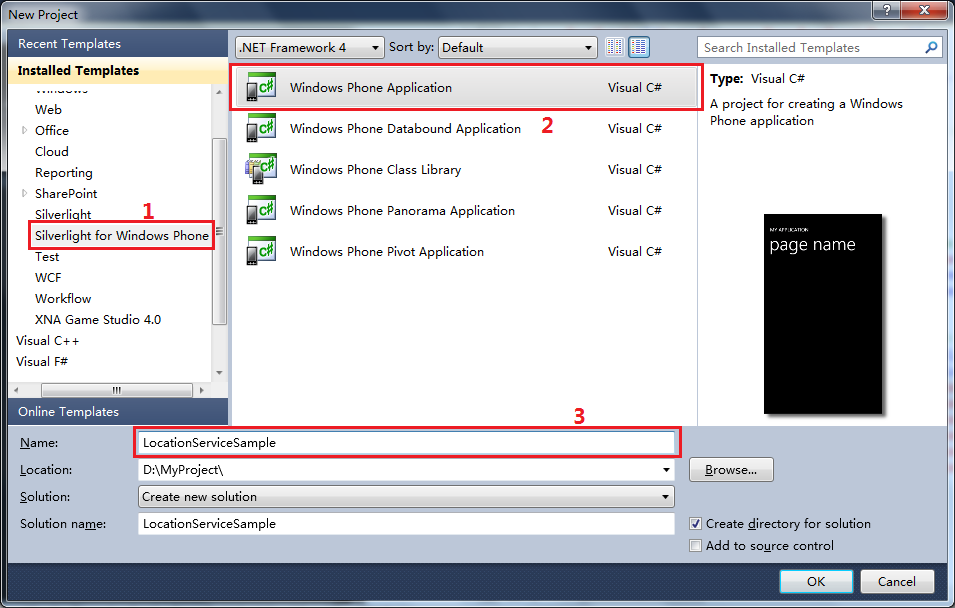


图4-9 新建

1. 在[Solution Explorer]窗体中选择[LocationServiceSample]工程中，然后点击右键，选择[Add Reference….]，如图。

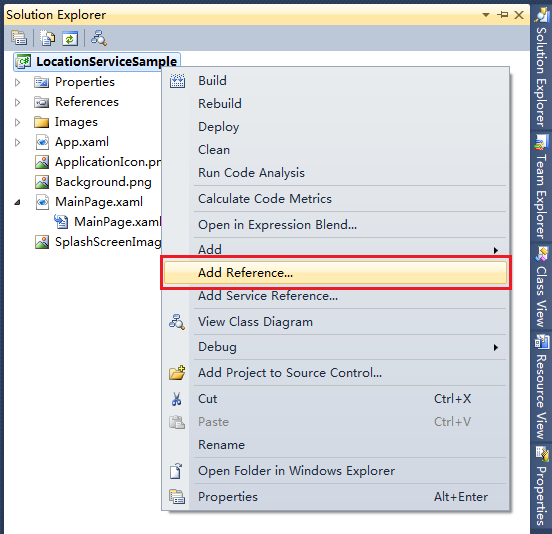


图4-10 添加服务

1. 在[.NET]选项卡，选择[System.Device]点击[ OK]。

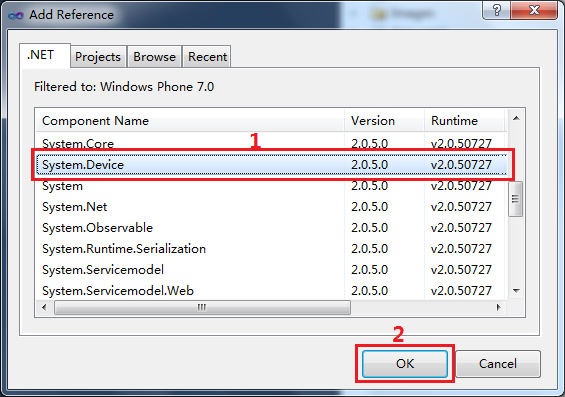


图4-11 System.Device

1. 在使用地理位置服务的MainPage.xaml.cs文件首部添加using指令。

Silverlight Project: LocationServiceSample File: MainPage.xaml.cs

using System.Device.Location;

1. 声明GeoCoordinateWatcher对象的变量watcher，通过watcher控制地位位置服务。

Silverlight Project: LocationServiceSample File: MainPage.xaml.cs

public partial class MainPage : PhoneApplicationPage

{

/// <summary>

/// This sample receives data from the Location Service and displays the geographic coordinates of the device.

/// </summary>

GeoCoordinateWatcher watcher;

1. 地理位置服务的使用

最简单的位置的应用程序类型是设定其地理位置服务为可选项。我们要设计一个地理位置服务的精度可选的应用程序，通过选项设置选择高精度或者电力优化的方式来获取地理位置信息。为了提醒您在使用完毕地理位置服务时要将其及时关闭，以达到节省电力的作用。我们专门做了启用和关闭地理位置服务的按钮。

建议您在程序中友好的提醒用户初次启动地理位置服务需要等待一段时间。

为了更好控制功耗，建议您在使用时启动服务，而不是在应用程序的构造函数中启动服务。在本例中我们通过按钮的click事件处理程序中启动该服务。

使用GeoCoordinateWatcher 对象启动数据采集。

Silverlight Project: LocationServiceSample File: MainPage.xaml.cs

/// <summary>

/// Click event handler for the low accuracy button

/// </summary>

/// <param name="sender">The control that raised the event</param>

/// <param name="e">An EventArgs object containing event data.</param>

private void LowButtonClick(object sender, EventArgs e)

{

// Start data acquisition from the Location Service, low accuracy

accuracyText = "power optimized";

StartLocationService(GeoPositionAccuracy.Default);

}

/// <summary>

/// Click event handler for the high accuracy button

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void HighButtonClick(object sender, EventArgs e)

{

// Start data acquisition from the Location Service, high accuracy

accuracyText = "high accuracy";

StartLocationService(GeoPositionAccuracy.High);

}

/// <summary>

/// Helper method to start up the location data acquisition

/// </summary>

/// <param name="accuracy">The accuracy level </param>

private void StartLocationService(GeoPositionAccuracy accuracy)

{

// Reinitialize the GeoCoordinateWatcher

StatusTextBlock.Text = "starting, " + accuracyText;

watcher = new GeoCoordinateWatcher(accuracy);

watcher.MovementThreshold = 20;

添加StatusChanged和PositionChanged的事件的事件处理程序。

// Add event handlers for StatusChanged and PositionChanged events

watcher.StatusChanged += new EventHandler<GeoPositionStatusChangedEventArgs>(watcher\_StatusChanged);

watcher.PositionChanged += new EventHandler<GeoPositionChangedEventArgs<GeoCoordinate>>(watcher\_PositionChanged);

启动地理位置服务。

// Start data acquisition

watcher.Start();

}

1. 实现StatusChanged事件处理程序

当地理位置服务的状态发生变化时，将引发此事件。通过GeoPositionStatusChangedEventArgs 对象的Status获得地理位置服务的当前状态。

Silverlight Project: LocationServiceSample File: MainPage.xaml.cs

/// <summary>

/// Handler for the StatusChanged event. This invokes MyStatusChanged on the UI thread and

/// passes the GeoPositionStatusChangedEventArgs

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void watcher\_StatusChanged(object sender, GeoPositionStatusChangedEventArgs e)

{

Deployment.Current.Dispatcher.BeginInvoke(() => MyStatusChanged(e));

}

/// <summary>

/// Custom method called from the StatusChanged event handler

/// </summary>

/// <param name="e"></param>

void MyStatusChanged(GeoPositionStatusChangedEventArgs e)

{

switch (e.Status)

{

case GeoPositionStatus.Disabled:

// The location service is disabled or unsupported.

// Alert the user

StatusTextBlock.Text = "location is unsupported on this device";

break;

case GeoPositionStatus.Initializing:

// The location service is initializing.

// Disable the Start Location button

StatusTextBlock.Text = "initializing location service," + accuracyText;

break;

case GeoPositionStatus.NoData:

// The location service is working, but it cannot get location data

// Alert the user and enable the Stop Location button

StatusTextBlock.Text = "data unavailable," + accuracyText;

break;

case GeoPositionStatus.Ready:

// The location service is working and is receiving location data

// Show the current position and enable the Stop Location button

StatusTextBlock.Text = "receiving data, " + accuracyText;

break;

}

}

1. 实现PositionChanged事件处理程序

地理位置服务一旦准备和接收的数据，它将开始引发 PositionChanged 事件并调用您的应用程序的处理程序。在事件处理程序中，获取GeoPositionChangedEventArg对象，读取经度和纬度值。

Silverlight Project: LocationServiceSample File: MainPage.xaml.cs

/// <summary>

/// Handler for the PositionChanged event. This invokes MyStatusChanged on the UI thread and

/// passes the GeoPositionStatusChangedEventArgs

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void watcher\_PositionChanged(object sender, GeoPositionChangedEventArgs<GeoCoordinate> e)

{

Deployment.Current.Dispatcher.BeginInvoke(() => MyPositionChanged(e));

}

/// <summary>

/// Custom method called from the PositionChanged event handler

/// </summary>

/// <param name="e"></param>

void MyPositionChanged(GeoPositionChangedEventArgs<GeoCoordinate> e)

{

// Update the TextBlocks to show the current location

LatitudeTextBlock.Text = e.Position.Location.Latitude.ToString("0.000");

LongitudeTextBlock.Text = e.Position.Location.Longitude.ToString("0.000");

}

1. 停止服务

为最大程度的优化程序的功耗，在不使用时请关闭地理位置服务。此示例中，"停止位置"按钮允许用户停止位置服务实现。

Silverlight Project: LocationServiceSample File: MainPage.xaml.cs

private void StopButtonClick(object sender, EventArgs e)

{

if (watcher != null)

{

watcher.Stop();

}

StatusTextBlock.Text = "location service is off";

LatitudeTextBlock.Text = " ";

LongitudeTextBlock.Text = " ";

}

1. Debug程序

假设我们在编写一套汽车上使用的出行智能导航的程序，项目在测试阶段需要获取地理位置信息的数据支持程序的运行，此时又不具备安装到汽车上行驶测试的条件，该怎么办呢？微软提供Reactive Extensions 可实现的地理位置服务仿真模拟GPS数据，为程序的测试提供模拟数据，完成实验室测试的任务。科技无所不能，智能科技轻松搞定繁杂的工作，犹如威猛先生，轻松搞定厨房的油污。更酷的是在Windows Phone Mango提供的位置服务的模拟器，应用程序轻松找到了我所在的位置坐标。

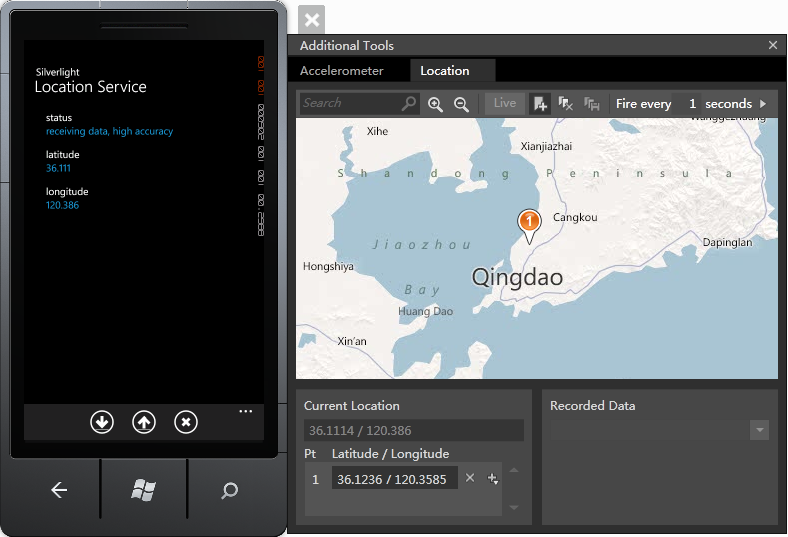


图4-12 运行结果

## 云计算服务

对于Windows phone，微软CEO鲍尔默表示：“全新的Windows手机把网络、个人电脑和手机的优势集于一身，让人们可以随时随地享受到想要的体验。”

微软针对Windows Phone的云计算服务包括：信息推送通知服务、地理位置服务、Xbox Live集成，以及应用程序部署等等。Windows Phone俨然是一部“云计算手机”。

微软技术顾问王立楠先生曾在《让云触手可及—微软云计算实践指南》一书中讲解了在Windows Phone上开发云计算的客户端应用。下面我们就站在巨人的肩膀上，在该书中的云计算实例的基础上，发挥创造力更进一步开发Windows Phone的云计算客户端应用程序。

### 先决条件

在动手实践—Windows Phone的云计算客户端应用程序需要具备的先决条件：

1. 安装云计算开发的工具

请参考MSDN云计算开发（<http://msdn.microsoft.com/zh-cn/ff380142>）。

1. 在本机模拟云计算服务平台

Debug运行示例云计算解决方案HelloCloud。如果您有Windows Azure的账号，那么此步骤可省略。您可以在动手实践—Windows Phone的云计算客户端应用程序中直接连接云端平台。我们提供解决方案HelloCloud的目的就是在没有云端平台支持的情况下，模拟出一个云端平台。

Debug运行后会出现两个窗体：

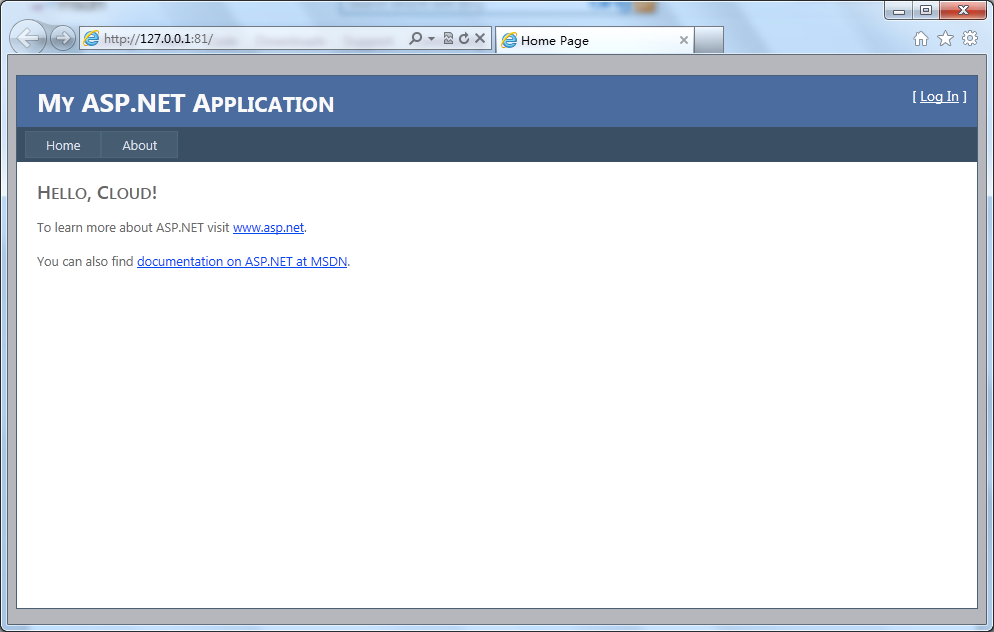


图4-13 HelloCloud云端平台

HelloCloud解决方案中的创建的可运行于云计算平台的ASP.NET应用程序。HelloCloud为我们创建了一个云计算服务，如图4-14 云端服务所示。

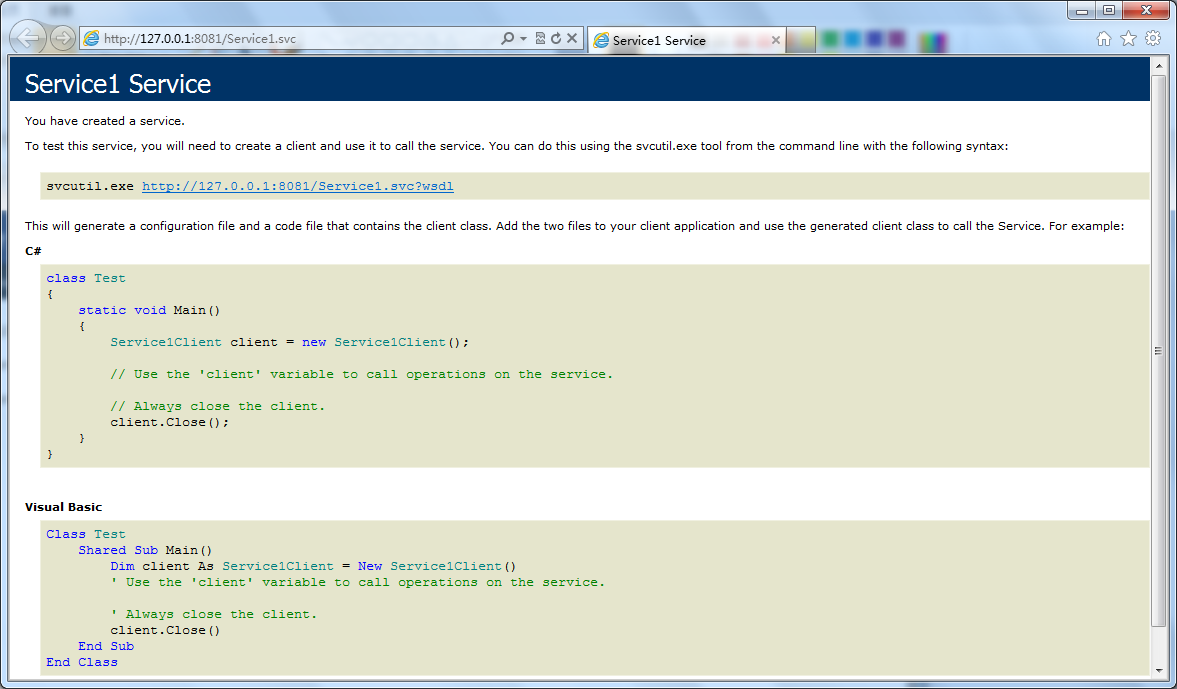


图4-14 云端服务

模拟云计算服务的地址为<http://127.0.0.1:8081/Service1.svc>

在动手实践—Windows Phone的云计算客户端应用程序将会连接此模拟的云计算服务。

### 动手实践——Windows Phone的云计算客户端应用程序

在本例中我们调用云计算平台的Web服务，获取云计算平台的系统时间。

1. 新建工程

打开Visual Studio 2010，新建Silverlight for Windows Phone工程，名称为CloudClientOnWindowsPhone，如图。

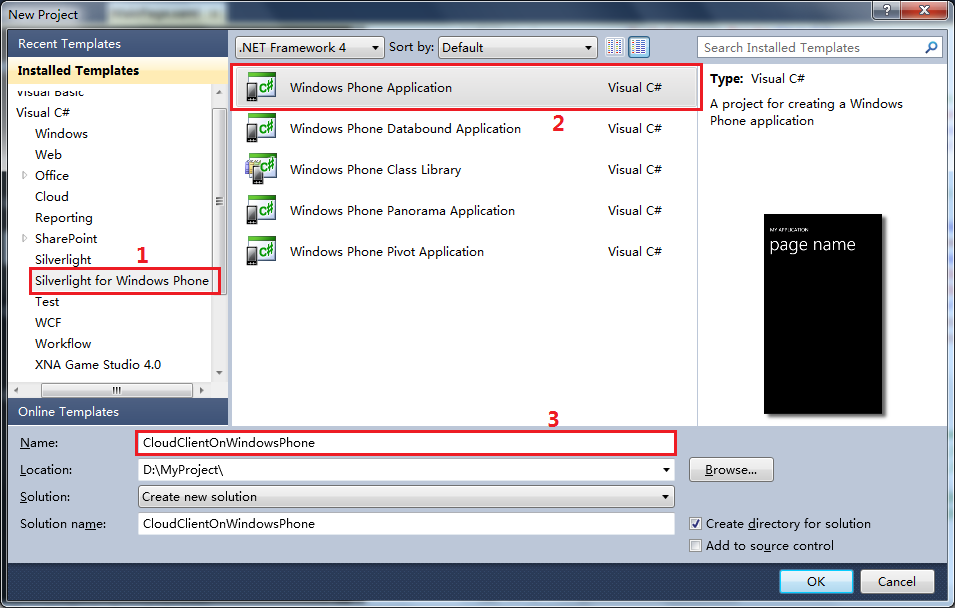


图4-15 新建CloudClientOnWindowsPhone工程

1. 添加服务引用

使用鼠标的右键点中 [Solution Explorer] 窗口中的项目名称CloudClientOnWindowsPhone，从出现的菜单选择 [Add Service Reference…] 功能，如图。

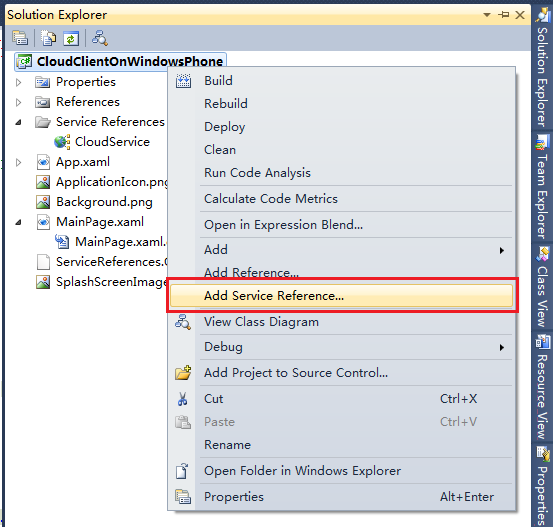


图4-16 添加服务引用

1. 在弹出的[Add Service Reference]向导中的[Address]输入云计算服务的地址为<http://127.0.0.1:8081/Service1.svc>，点击[GO]；
2. 在[Services]中会出现HelloCloud工程提供的服务Service1；
3. 在[Namespace]框中输入“CloudService”，单击[OK]。如图4-17 [Add Service Reference]向导。

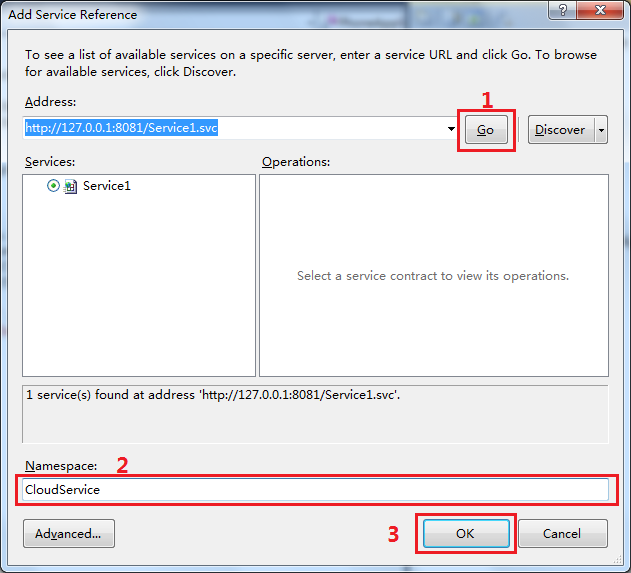


图4-17 [Add Service Reference]向导

服务引用添加完毕后，我们就可以在客户端程序的代码里直接访问云计算平台上的Web服务了。

1. 添加显示时间的控件

打开MainPage.xaml文件，添加显示日期和时间的DatePicker、TimePicker和TextBlock控件。因为DatePicker和TimePicker控件默认不在[Toolbox]中显示，所以我们要手动将其添加入Toolbox。

在Toolbox任意位置点击右键，选择[Choose Items…]选项，如图4-18 [Choose Items…]选项

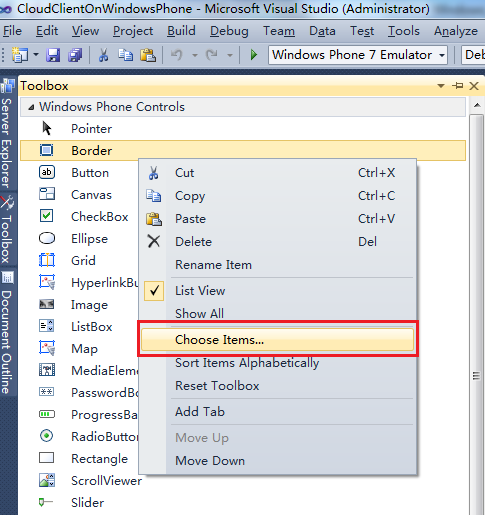


图4-18 [Choose Items…]选项

在[Windows Phone Components]选项卡中选中[DatePicker]和[TimerPicker]，然后点击[OK]。如图4-19 [Windows Phone Components]选项卡。这样我们就在Toolbox中可以选择DatePicker和TimePicker，将其加入到程序中。

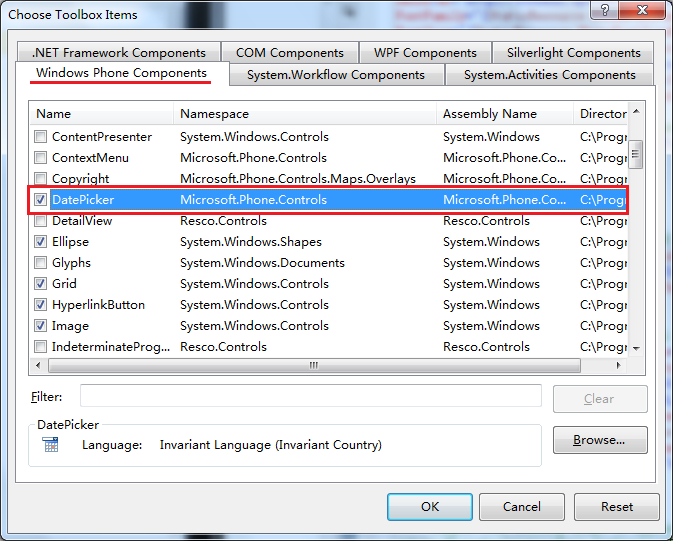


图4-19 [Windows Phone Components]选项卡

修改MainPage.xaml中ContentPanel的代码如下：

Silverlight Project: CloudClientOnWindowsPhone File: MainPage.xaml

<!--ContentPanel - place additional content here-->

<Grid x:Name="ContentGrid" Grid.Row="1">

<TextBlock Text="Local Time" Style="{StaticResource PhoneTextNormalStyle}" Margin="10,47,259,532" />

<toolkit:DatePicker Margin="0,91,307,445" />

<toolkit:TimePicker Margin="156,91,207,451" />

<TextBlock Name="textBlock1" Text="\*\* \*\*" FontSize="40" TextAlignment="Center" Height="59" VerticalAlignment="Bottom" Margin="12,0,335,167" Visibility="Visible" />

<TextBlock Style="{StaticResource PhoneTextNormalStyle}" Text="Cloud Time" Margin="12,225,347,351" />

<TextBlock Name="textBlock2" Text="\*\* \*\*" FontSize="72" TextAlignment="Center" Height="97" VerticalAlignment="Top" Margin="10,278,297,0" Visibility="Visible" />

<TextBlock Height="48" Margin="6,499,0,0" Name="textBlock3" Text="Touch To Get Cloud Date and Time" VerticalAlignment="Top" TextAlignment="Center" Opacity="0.5" Visibility="Visible" />

</Grid>

1. 添加鼠标消息响应事件调用云计算服务

在MainPage.xaml中选择“PhoneApplicationPage”对象，打开[Properties]浏览器，在[Events]选项卡中找到[MouseLeave]，双击[MouseLeave]事件，Visual Studio会打开代码编辑界面，并自动添加“MouseLeave”的事件处理函数。如图4-20 “PhoneApplicationPage”的**[MouseLeave]**所示。

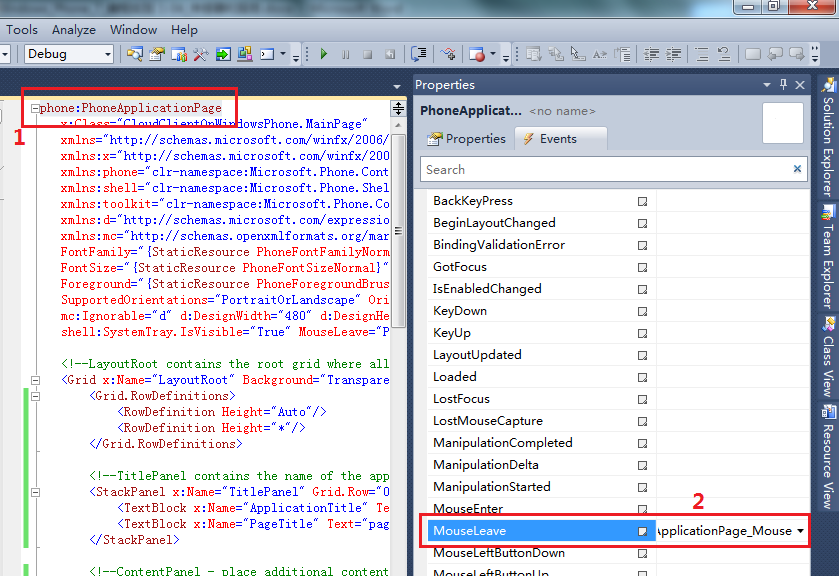


图4-20 “PhoneApplicationPage”的**[MouseLeave]**

添加Windows Azure平台上Web服务的客户端对象CloudService.Service1Client，修改事件处理函数PhoneApplicationPage\_MouseLeave。

Silverlight Project: CloudClientOnWindowsPhone File: MainPage.xaml.cs

public partial class MainPage : PhoneApplicationPage

{

CloudService.Service1Client sc;//Windows Azure平台上Web服务的客户端对象

// Constructor

public MainPage()

{

InitializeComponent();

}

private void PhoneApplicationPage\_MouseLeave(object sender, MouseEventArgs e)

{

if (sc == null)

{

sc = new CloudService.Service1Client();

sc.GetServerTimeCompleted +=

new EventHandler<CloudService.GetServerTimeCompletedEventArgs>

(sc\_GetServerTimeCompleted);

}

this.textBlock1.Text = "";

this.textBlock2.Text = "Checking...";

sc.GetServerTimeAsync();

}

}

事件处理函数PhoneApplicationPage\_MouseLeave初始化了一个Windows Azure平台上Web服务的客户端对象实例，并调用了函数GetServerTimeAsync获取系统时间。这是一个异步调用的函数，需要一个回调函数来执行GetServerTimeAsync调用完成后的操作。

在事件处理函数PhoneApplicationPage\_MouseLeave之后添加如下代码。

Silverlight Project: CloudClientOnWindowsPhone File: MainPage.xaml.cs

void sc\_GetServerTimeCompleted(object sender, CloudService.GetServerTimeCompletedEventArgs e)

{

this.textBlock1.Text = e.Result.ToLongDateString();

this.textBlock2.Text = e.Result.ToShortTimeString();

}

这个函数会在Web服务调用完成之后执行，它会把调用结果写在相应的TextBlock控件里。

1. Debug程序

从[Debug]菜单中选择[Start Debugging]或者按F5键启动应用程序。在模拟器中运行结果如图4-21 程序启动所示。

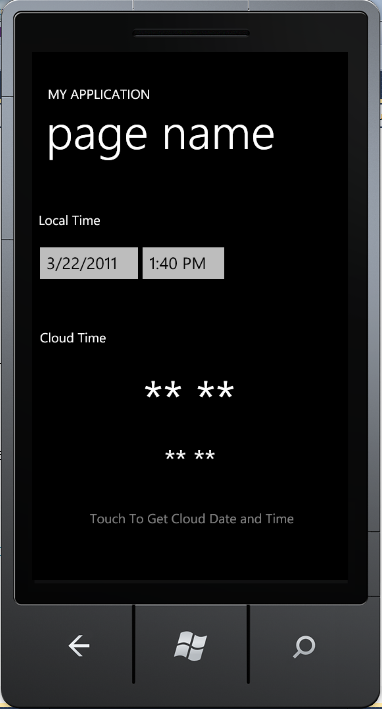


图4-21 程序启动

在代码中设置两个断点，以方便查看程序运行方式。鼠标点击模拟器屏幕的任何位置，程序开始响应鼠标事件。Windows Azure平台上Web服务的客户端对象CloudService.Service1Client调用函数GetServerTimeAsync获取系统时间，如图4-22 响应鼠标事件。

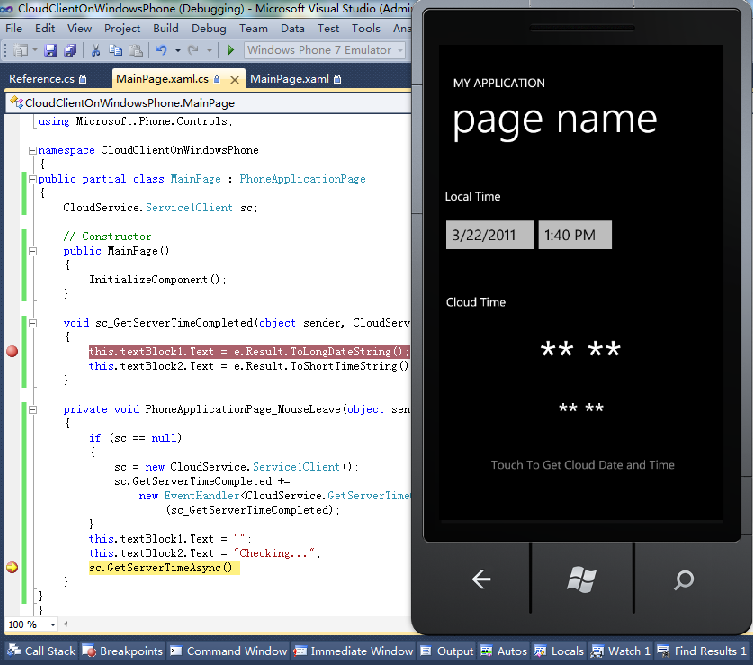


图4-22 响应鼠标事件

按F5继续执行调试，程序进入回调函数sc\_GetServerTimeCompleted中，如图4-23 回调函数sc\_GetServerTimeCompleted。

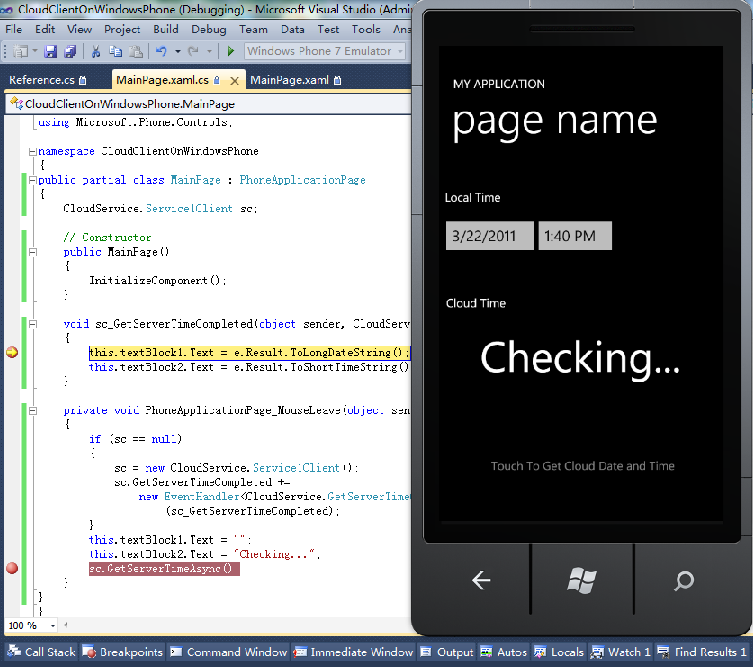


图4-23 回调函数sc\_GetServerTimeCompleted

按F5继续执行调试，程序显示运行结果，如图4-24 运行结果。

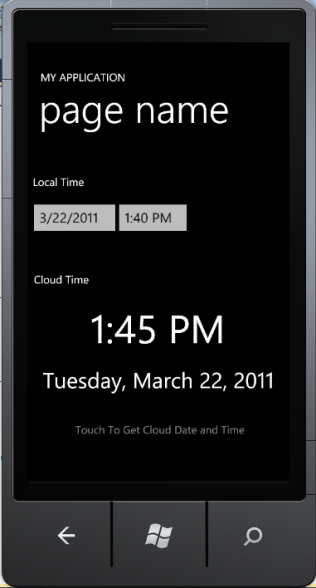


图4-24 运行结果

在此例中我们用DatePicker（日期选择）和TimePicker（时间选择）控件显示程序启动时刻的日期和时间，这两个是典型Metro风格的控件。

图4-25 日期和时间选择

## 设备状态和网络信息

Windows Phone Mango中增加了可选的硬件组件—陀螺仪，因为硬件开始出现差异化，所以也为应用程序提供了检测硬件设备能力和状态的方法。本节将重点介绍设备信息的查询方法，包括设备状态和网络信息。

### 动手实践——获取和显示设备状态和网络信息

1. 获取和显示设备信息

示例应用程序DeviceStatus采用Model –View–ViewModel架构，在解决方案中，有Model、Service和ViewModel文件夹。

* **Model** --不包含设备信息的数据模型:
  + **Information** --设备属性的抽象类
  + **CapabilityInformation** --继承于**Information**的抽象类，增加了设备功能属性。
  + **NetworkInformation** --继承于**Information**的抽象类，增加了网络状态和性能的属性。
  + **DeviceInformation**--继承于**Information**的抽象类， 增加了设备硬件版本等信息的属性。
* **Service**—包含ViewModel所使用的获取设备信息的方法和类。
  + **IInformationProvider**—定义一个实体，可以提供有关设备本身的信息，其功能和它的网络功能和状态。
  + **Fake**—设计阶段使用的虚拟数据。
  + **Real**—真实设备的实际数据。
* **ViewModel**—包含应用程序的ViewModel模型，用于连接数据和用户界面。
  + **DeviceInformationViewModel**—以**IInformationProvider**类的形式提供设备信息，在设计模式时提供模拟数据，在运行状态下提供真实数据。

在DeviceStatus\Service\Real\RealDeviceInformation.cs中提供了获取设备信息的方法，包括电力PowerSource、系统版本DeviceFirmwareVersion、硬件版本DeviceHardwareVersion、制造商DeviceManufacturer、设备名称DeviceName、内存DeviceTotalMemory和IsKeyboardPresent。

Silverlight Project: DeviceStatus File: \Service\Real\RealDeviceInformation.cs

using System;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Ink;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace DeviceStatus

{

/// <summary>

/// Provides device information for the actual device.

/// </summary>

public class RealDeviceInformation : DeviceInformation

{

/// <summary>

/// Gathers information from the device.

/// </summary>

public override void RefreshData()

{

PowerSource = Microsoft.Phone.Info.DeviceStatus.PowerSource.ToString();

FirmwareVersion = Microsoft.Phone.Info.DeviceStatus.DeviceFirmwareVersion;

HardwareVersion = Microsoft.Phone.Info.DeviceStatus.DeviceHardwareVersion;

Manufacturer = Microsoft.Phone.Info.DeviceStatus.DeviceManufacturer;

Name = Microsoft.Phone.Info.DeviceStatus.DeviceName;

TotalMemory = (Microsoft.Phone.Info.DeviceStatus.DeviceTotalMemory / 1048576).ToString() + "MB";

HasKeyboard = Microsoft.Phone.Info.DeviceStatus.IsKeyboardPresent;

}

}

}

1. 获取和显示网络信息

本节中我们将介绍Microsoft.Phone.Net.NetworkInformation命名空间的DeviceNetworkInformation 和NetworkInterface类，介绍判断网络状态和性能的方法，检测网络是否连接以及网络连接的类型。

打开解决方案中的**Service\Real\ RealNetworkInformation.cs**文件，引用中包含Microsoft.Phone.Net.NetworkInformation。

Silverlight Project: DeviceStatus File: \Service\Real\ RealNetworkInformation.cs

using Microsoft.Phone.Net.NetworkInformation;

RefreshData方法获取设备的连接状态，GetInterfaceTypeString方法获取可用的连接类型。

Silverlight Project: DeviceStatus File: \Service\Real\ RealNetworkInformation.cs

* + 1. public class RealNetworkInformation : NetworkInformation
    2. {
    3. public override void RefreshData()
    4. {
    5. IsConnected = Microsoft.Phone.Net.NetworkInformation.

DeviceNetworkInformation.IsNetworkAvailable;

* + 1. ConnectionType =
    2. GetInterfaceTypeString(Microsoft.Phone.Net.NetworkInformation.
    3. NetworkInterface.NetworkInterfaceType);
    4. MobileOperator = Microsoft.Phone.Net.NetworkInformation.
    5. DeviceNetworkInformation.CellularMobileOperator;
    6. if (String.IsNullOrEmpty(MobileOperator))
    7. {
    8. MobileOperator = "N/A";
    9. }
    11. IsCellularDataEnabled = Microsoft.Phone.Net.NetworkInformation.
    12. DeviceNetworkInformation.IsCellularDataEnabled;
    13. IsCellularDataRoamingEnabled = Microsoft.Phone.Net.NetworkInformation.
    14. DeviceNetworkInformation.IsCellularDataRoamingEnabled;
    15. IsWifiEnabled = Microsoft.Phone.Net.NetworkInformation.
    16. DeviceNetworkInformation.IsWiFiEnabled;
    17. }
    18. private string GetInterfaceTypeString(Microsoft.Phone.Net.
    19. NetworkInformation.NetworkInterfaceType networkInterfaceType)
    20. {
    21. switch (networkInterfaceType)
    22. {
    23. case NetworkInterfaceType.AsymmetricDsl:
    24. return "Asymmetric DSL";
    25. case NetworkInterfaceType.Atm:
    26. return "Atm";
    27. case NetworkInterfaceType.BasicIsdn:
    28. return "Basic ISDN";
    29. case NetworkInterfaceType.Ethernet:
    30. return "Ethernet";
    31. case NetworkInterfaceType.Ethernet3Megabit:
    32. return "3 Mbit Ethernet";
    33. case NetworkInterfaceType.FastEthernetFx:
    34. return "Fast Ethernet";
    35. case NetworkInterfaceType.FastEthernetT:
    36. return "Fast Ethernet";
    37. case NetworkInterfaceType.Fddi:
    38. return "FDDI";
    39. case NetworkInterfaceType.GenericModem:
    40. return "Generic Modem";
    41. case NetworkInterfaceType.GigabitEthernet:
    42. return "Gigabit Ethernet";
    43. case NetworkInterfaceType.HighPerformanceSerialBus:
    44. return "High Performance Serial Bus";
    45. case NetworkInterfaceType.IPOverAtm:
    46. return "IP Over Atm";
    47. case NetworkInterfaceType.Isdn:
    48. return "ISDN";
    49. case NetworkInterfaceType.Loopback:
    50. return "Loopback";
    51. case NetworkInterfaceType.MobileBroadbandCdma:
    52. return "CDMA Broadband Connection";
    53. case NetworkInterfaceType.MobileBroadbandGsm:
    54. return "GSM Broadband Connection";
    55. case NetworkInterfaceType.MultiRateSymmetricDsl:
    56. return "Multi-Rate Symmetrical DSL";
    57. case NetworkInterfaceType.None:
    58. return "None";
    59. case NetworkInterfaceType.Ppp:
    60. return "PPP";
    61. case NetworkInterfaceType.PrimaryIsdn:
    62. return "Primary ISDN";
    63. case NetworkInterfaceType.RateAdaptDsl:
    64. return "Rate Adapt DSL";
    65. case NetworkInterfaceType.Slip:
    66. return "Slip";
    67. case NetworkInterfaceType.SymmetricDsl:
    68. return "Symmetric DSL";
    69. case NetworkInterfaceType.TokenRing:
    70. return "Token Ring";
    71. case NetworkInterfaceType.Tunnel:
    72. return "Tunnel";
    73. case NetworkInterfaceType.Unknown:
    74. return "Unknown";
    75. case NetworkInterfaceType.VeryHighSpeedDsl:
    76. return "Very High Speed DSL";
    77. case NetworkInterfaceType.Wireless80211:
    78. return "Wireless";
    79. default:
    80. return "Unknown";
    81. }
    82. }
    83. }

按F5运行应用程序，或者点击Start Debugging按钮运行，如图4-26 Start Debugging。



图4-26 Start Debugging

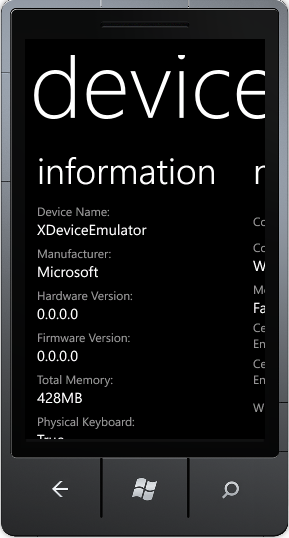


图4-27 运行结果