> 由 KiOii (*_EM_Cpper*)整理。 (KiOii (*_EM_Cpper*) makes this note.)

Understanding Nodes

In this chapter

- JavaFX中的node指的是什么
- JavaFX中的笛卡尔坐标系(Cartesian coordinate system)
- 怎么设置node的大小和位置
- 怎么保存关于node的用户数据
- 什么是 managed node
- 如何在坐标空间之间转换(transform)节点的边界

1. What Is a Node ?

- define : Every **item** in a scene graph
- subitem
    - define : children of branch node
        > branch node is an instance of the Parent
    - branch node
        - such as : Group、Region、WebView
    - leaf bode :
        - such as : Rectangle, Text, ImageView, and MediaView

- A node may be created and modified on any thread if it is not yet attached to a scene

  > 没有被附加到scene之前，节点是可以在任何线程上被创建和修改的
  >
  > 一旦被attached到一个scene上面时，只能在JavaFX Application Thread上进行修改

- bound : A node has **several types of bounds**

  > 界限是由不同的坐标系决定的

2. The Cartesian Coordinate System

- define : 一种定义二维平面上每个点的方式

  > rectangular coordinate system : 直角坐标系
  >
  > origin： 原点

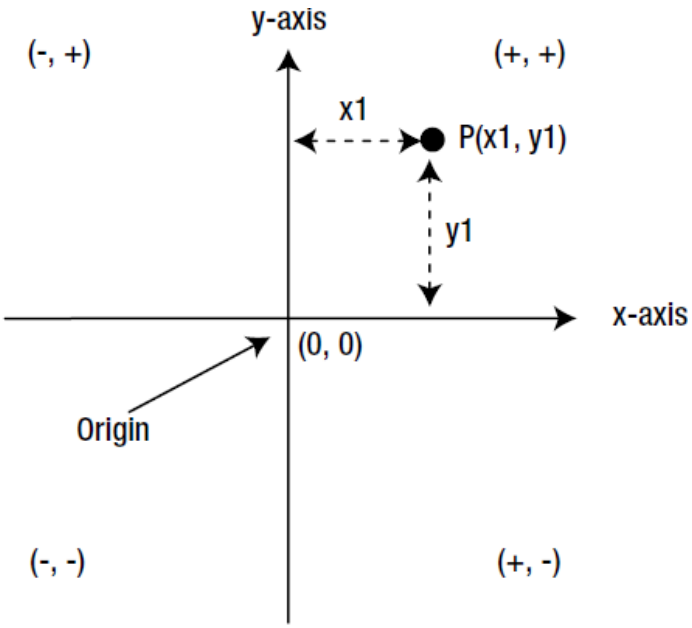

Figure 6-1. *A two-dimensional Cartesian coordinate system used in coordinate geometry*

  - transformation : 平移(translation)、旋转(ratation)、缩放(scaling)和剪切 (shearing)
    - translation : `(x,y)  --------(a,b)-------(x+a,y+b)`
    - ratation : around a pivot point in the coordinate space and the coordinates of points are mapped to the new axes

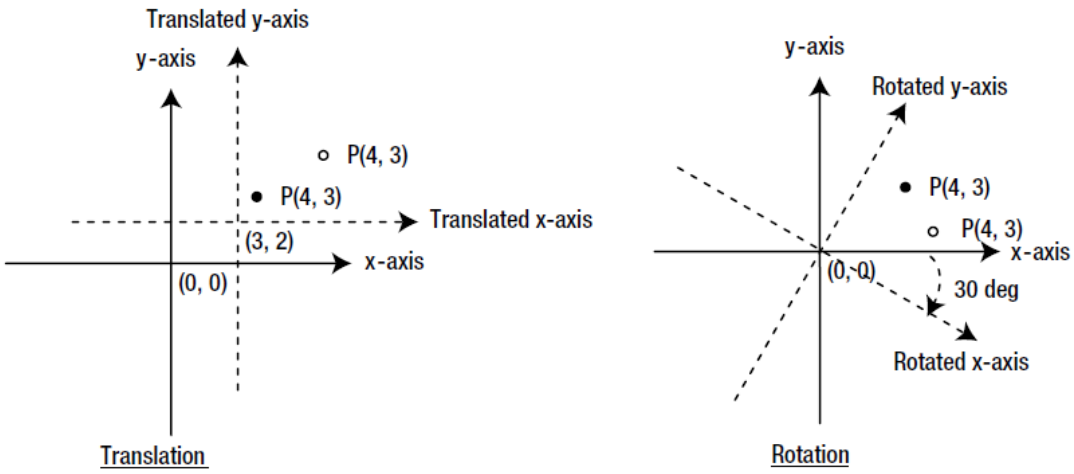

Figure 6-2. *Examples of translation and rotation transformations*

3. Cartesian Coordinate System of a Node

- Each node in a scene graph has its own coordinate system

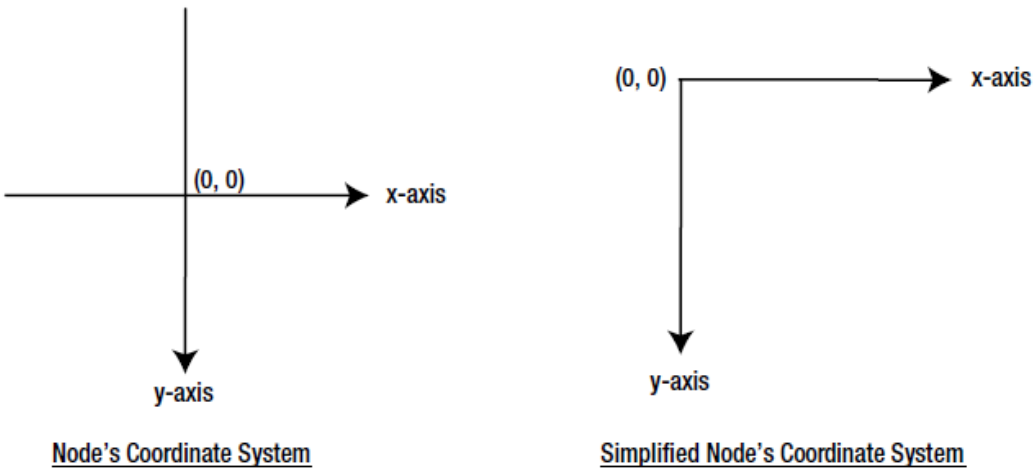

Figure 6-3. *The coordinate system of nodes*

- typical GUI application

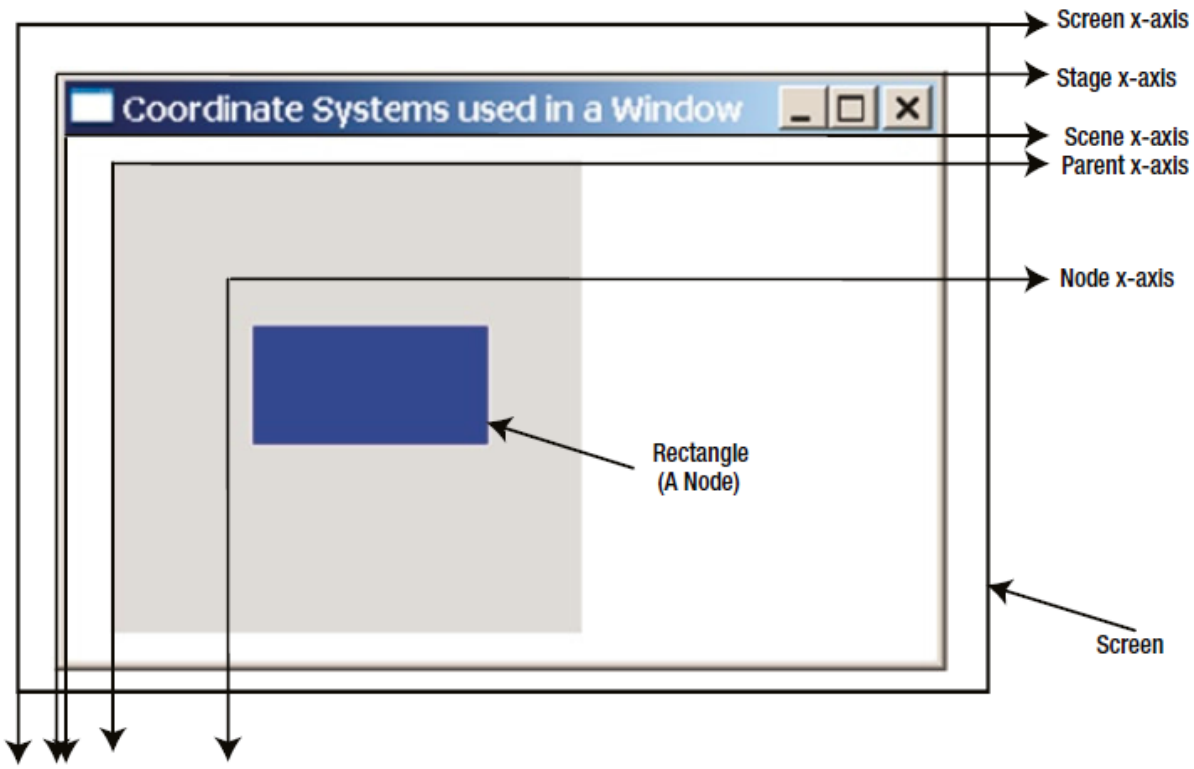

Figure 6-4. *Coordinate systems of all elements comprising a GUI window*

> 这边他五个坐标系统，注意参照对象

4. The concept of Bounds and Bounding Box

> 边界和边界框的概念

- 每个节点都有一个几何图形，被放置在一个坐标空间中
- Bounds : size and position
- The bounds of a node : 边界矩形边框，包含了节点的整个几何图形

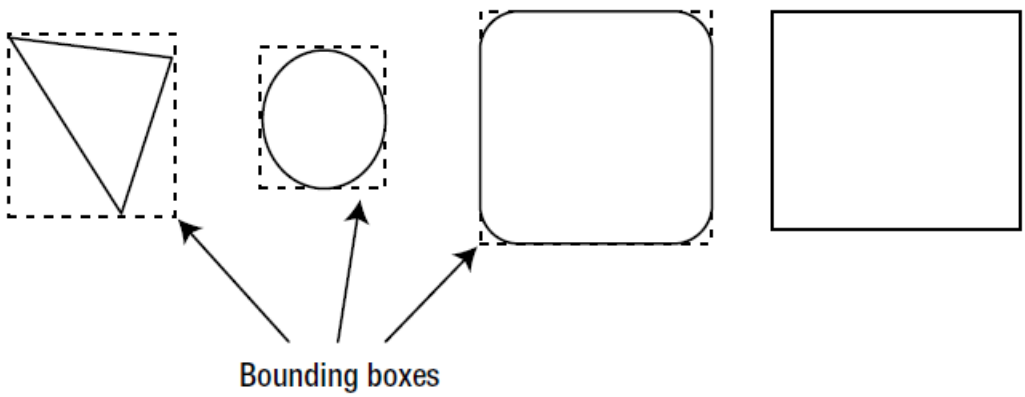

Figure 6-5. *The bounding rectangular boxdefining the geometric shape of nodes*

> 节点的几何图形和bounding box两者可能是不同的

- bounds of a node : An instance of the `javafx.gemotry.Bounds` class
  - **Bounds** class : abstract class

    > 被设计去在3Dspace中处理bounds

    - 它封装了左上角的坐标和边界框的最小深度，bounding box的宽度，高度，深度
    - **getMinX , getMinY, getMinZ**
    - **getWidth, getHeight, getDepth**
    - **getMaxX, getMaxY, getMaxZ**
    - 2D中，z和depth被设置为0

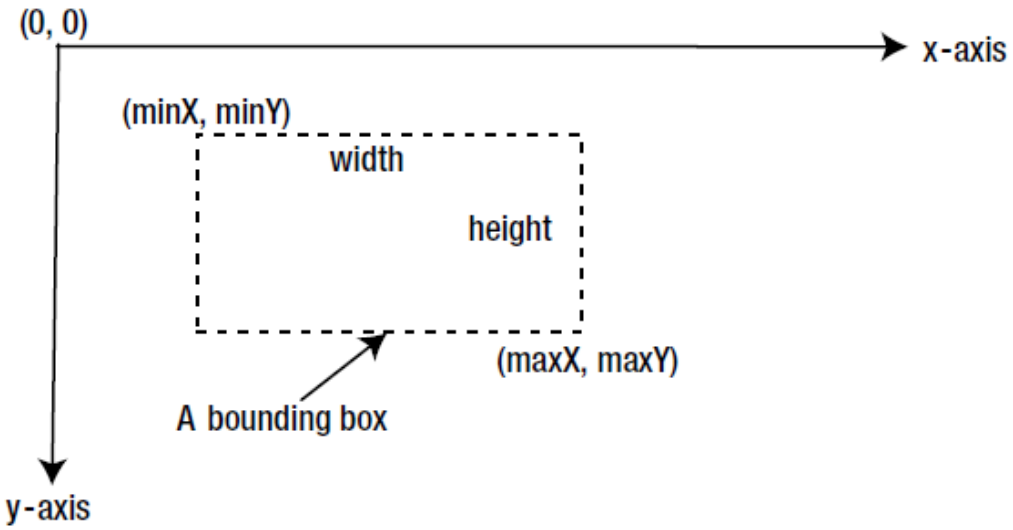

Figure 6-6. *The makings of a bounding box in a 2D space*

    - **isEmpty** : 如果Bounds的三维 （width、height、depth）是负的，则返回true
    - **contains** : 检测一个Bounds是否包含其他的Bounds，2D点，3D点
    - **intersects** : 检查边界的内部与另一个边界(2D或者3D点都行)的内部是否相交
  - **BoundingBox** class : concrete implementation of the **Bounds** class

5. Knowing the Bounds of a Node

- different types of bounds of a node

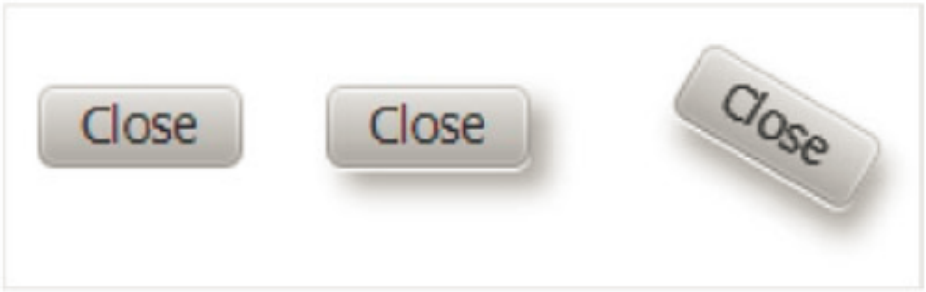

Figure 6-7. *A button with and without an effect and a transformation*

> 1、no effects or transformations
>
> 2、drop shadow effect
>
> 3、drop shadow effect and a rotation transformation
>
> 当应用effects或者tansformations，bounds发生改变

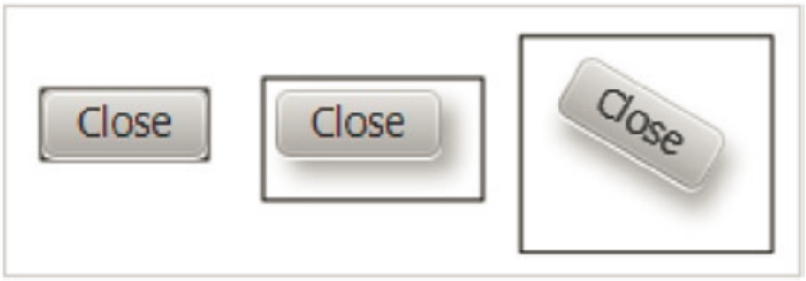

Figure 6-8. *A button with and with an effect and a transformation with bounding boxes*

- read-only **properties** in the Node class
  - about types of bounds
    - layoutBounds
    - boundsInLocal
    - boundsInParent

> 注意坐标是根据坐标空间来定义的（minX,minY）
>
> 注意节点的属性——几何、笔画、效果、剪辑和转换被包含在特定的bounds type中
>
> A nonzero stroke is considered part of the geometry of a node for computing its bounds.

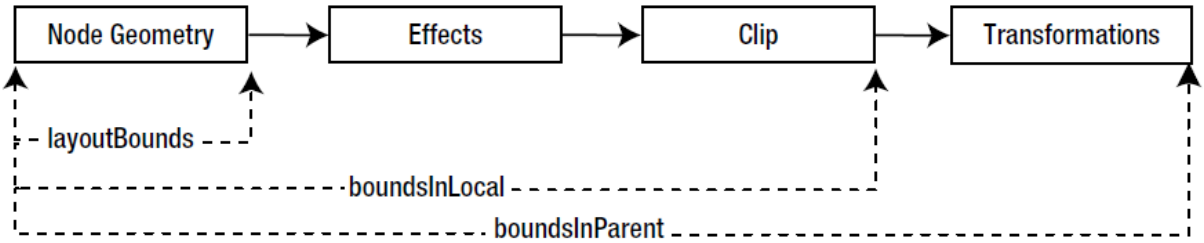

Figure 6-9. *Factors contributing to the size of a node*

Table 6-1. *Contributing Properties to the Bounds of a Node*

Bounds Type	Coordinate Space	Contributors
`layoutBounds`	Node (Untransformed)	Geometry of the node Nonzero stroke
`boundsInLocal`	Node (Untransformed)	Geometry of the node Nonzero stroke Effects Clip
`boundsInParent`	Parent	Geometry of the node Nonzero stroke Effects Clip Transformations

> boundsInLocal和BoundsInParent被称为物理或可视边界对应于节点的视觉效果
>
> layoutBounds也被称为逻辑边界，因为它并不一定与节点的物理边界相对应

5.1. The layoutBounds Property

- 在节点的**untransformed** local coordinate space中，根据**geometric property**计算出**layoutBounds property**
- 不包括： effects、clip、transformations
- 包括：bounding box的左上角坐标
  - for resizable node : 左上角坐标始终是 （0，0） （minX，minY）

    > Region, Control, WebView

  - for nonresizable node : 根据**geometric property**计算出来

    > Shape, Text, Group

```
package com.javafx.test;
```

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.VBox;
import javafx.scene.control.Button;
import javafx.scene.control.Control;

public class LayoutBoundsTest extends Application
{
    public static void main(String[] args)
    {
        Application.launch(args);
    }
    @Override
    public void start(Stage stage)
    {
        Button b1 = new Button("Close");
        b1.setEffect(new DropShadow());
        Button b2 = new Button("Close");
        b2.setRotate(30);
        VBox root = new VBox(b1,b2);
        Scene scene = new Scene(root,300,300);

        stage.setScene(scene);
        stage.show();
        System.out.println("b1=" + b1.getLayoutBounds());
        System.out.println("b2=" + b2.getLayoutBounds());
    }
}
```

> 由于会覆盖，因此如果想解决的话，可以使用Group，自动计算

```
root.getChildren().addAll(new Group(b1),new Group(b2));
```

> Using a Group to allocate space for effects and transformations of a node
>
> 注意：layoutBounds是根据geometric properties计算出来的，因此不能bind到表达式

5.2 The boundsInLocal Property

- boundsInLocal属性是在节点的未转换坐标空间中计算的
- 包含：geometric properties of node,effects, and clip
- 不包括：transformations

```
package com.javafx.test;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;

import javafx.scene.effect.DropShadow;
```

```
import javafx.scene.layout.VBox;
import javafx.scene.control.Button;

public class BoundsInLocalTest extends Application
{
    public static void main(String[] args)
    {
        Application.launch(args);
    }
    @Override
    public void start(Stage stage)
    {
        Button b1 = new Button("Close");
        b1.setEffect(new DropShadow());

        VBox root = new VBox();
        root.getChildren().addAll(b1);

        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();

        System.out.println("b1(layoutBounds) = " + b1.getLayoutBounds());
        System.out.println("b1(boundsInLocal)= " + b1.getBoundsInLocal());
    }
}
//输出
[minX:0.0, minY:0.0, minZ:0.0, width:48.0, height:23.0, depth:0.0, maxX:48.0, maxY:23.0,
maxZ:0.0]
b1(boundsInLocal)= BoundingBox [minX:-9.0, minY:-9.0, minZ:0.0, width:66.0, height:42.0,
depth:0.0, maxX:57.0, maxY:33.0, maxZ:0.0]
```

> 注意，boundInLocal包括了effect的宽度(9的差别)

- You would use boundsInLocal when you need to include the effects and the clip of a node

  > 需要包括effect和clip时才用：
  >
  > 假设你有一个带有reflection的文本节点，你想要垂直居中。如果你使用文本节点layoutBounds，它只会居中节点的文本部分，不包括reflection
  >
  > 另一个例子是检查有效果的球的碰撞。

5.3. The boundsInParent Property

- The boundsInParent property of a node is in **the coordinate space of its parent**
- It includes the **geometric propertie**s of the **node, effects, clip**, and **transformations**. It is rarely used directly in code.

6. Bounds of a Group

- 对于Group的layoutBounds、boundsInLocal、boundsInParent的计算不同于Node
  - A Group takes on the collection bounds of its children

> 包含计算孩子的bounds
>
> 你可以对孩子应用effect、clip等等，或者直接对Group应用，这样就自动对所有孩子应用

- The **layoutBounds** of a Group is the union of the boundsInParent of all its children

> layoutBounds group包含boundsInParent children（所以这些孩子作为新的geometry来对待）

- The **boundsInLocal** of a Group is computed by taking its layoutBounds and including the effects and clip applied directly to the Group
- The boundsInParent of a Group is computed by taking its boundsInLocal and including the transformations applied directly to the Group.

> 因此，如果你想分配足够的空间（boundsInParent），你需要放入Group
>
> 例如：Suppose you have a node with effects and transformations and you only want to allocate layout space for its effects, not its transformations. You can achieve this by applying the effects on the node and wrapping it in a Group, and then applying the transformations on the Group
>
> 注意：Group起到了隔离包裹的作用

7. A Detailed Example On Bounds

> 下面将会介绍演示如何计算出node的bounds

- first :

```
Rectangle r = new Rectangle(0, 0, 50, 20);
r.setFill(Color.GRAY);
```

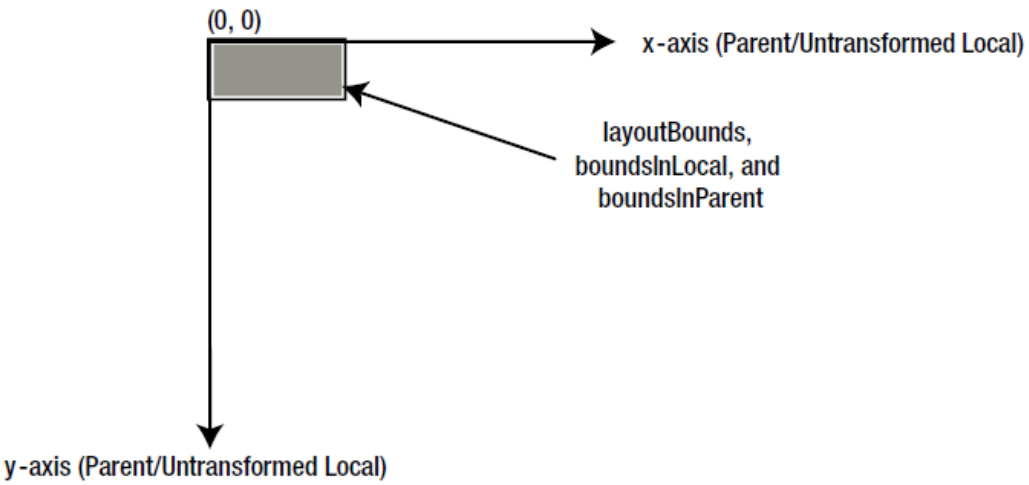

Figure 6-12. *A 50 by 20 rectangle placed at (0, 0) with no effects and transformations*

> 此时：
>
> layoutBounds[minX=0.0, minY=0.0, width=50.0, height=20.0] boundsInLocal[minX=0.0, minY=0.0, width=50.0, height=20.0] boundsInParent[minX=0.0, minY=0.0, width=50.0, height=20.0]
>
> Three types of bounds of the rectangle are the same

- second :

```
Rectangle r = new Rectangle(75, 50, 50, 20);
```

The resulting node is shown in Figure 6-13.

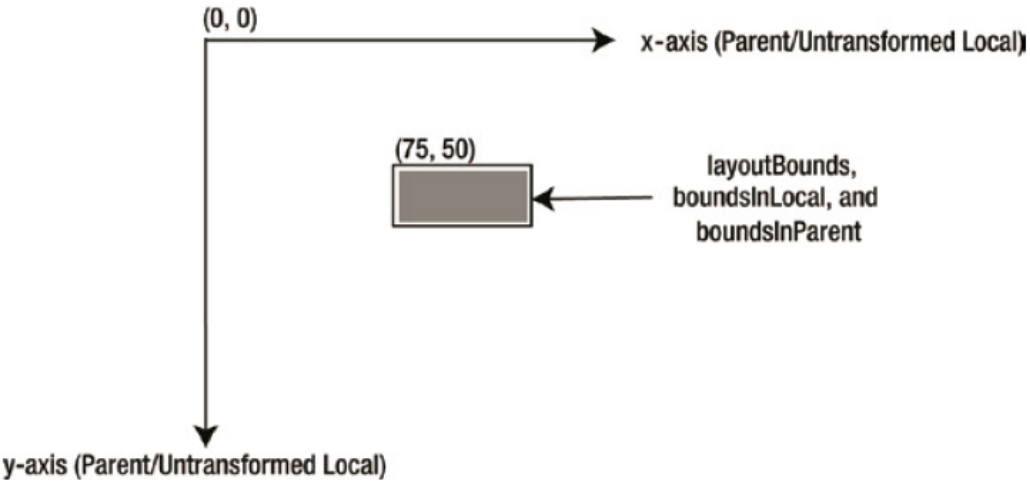

Figure 6-13. *A 50 by 20 rectangle placed at (75, 50) with no effects and transformations*

> layoutBounds[minX=75.0, minY=50.0, width=50.0, height=20.0] boundsInLocal[minX=75.0, minY=50.0, width=50.0, height=20.0] boundsInParent[minX=75.0, minY=50.0, width=50.0, height=20.0]

- third :

> 加特效

```
Rectangle r = new Rectangle(75, 50, 50, 20);
r.setEffect(new DropShadow());
```

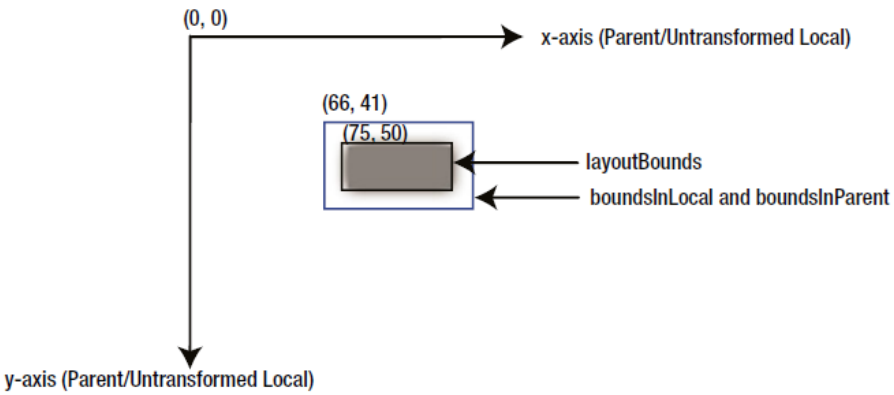

Figure 6-14. *A 50 by 20 rectangle placed at (75, 50) with a drop shadow and no transformations*

> layoutBounds[minX=75.0, minY=50.0, width=50.0, height=20.0] boundsInLocal[minX=66.0, minY=41.0, width=68.0, height=38.0] boundsInParent[minX=66.0, minY=41.0, width=68.0, height=38.0]

- fourth :

```
Rectangle r = new Rectangle(75, 50, 50, 20);
r.setEffect(new DropShadow());
r.getTransforms().add(new Translate(150, 75));
```

> 平移

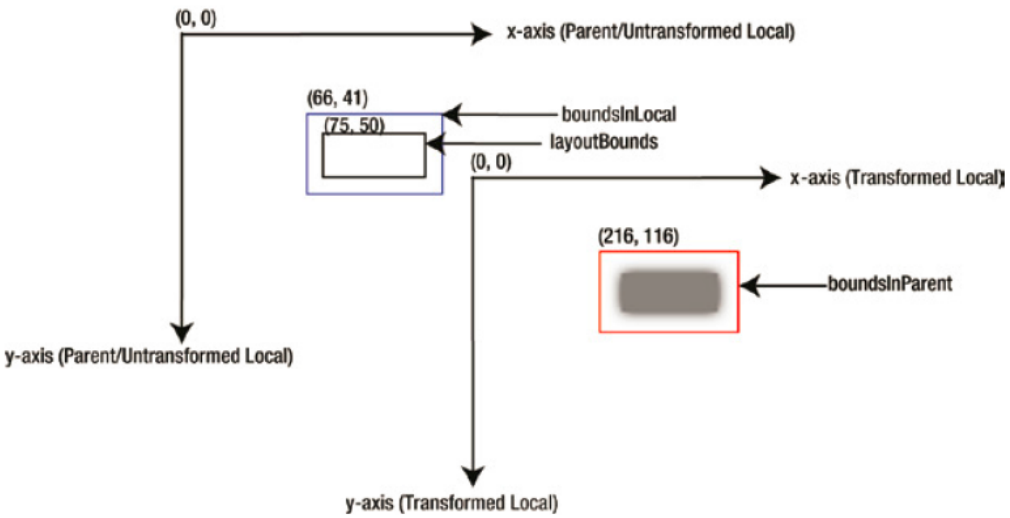

Figure 6-15. *A 50 by 20 rectangle placed at (75, 50) with a drop shadow and a (150, 75) translation*

> layoutBounds[minX=75.0, minY=50.0, width=50.0, height=20.0] boundsInLocal[minX=66.0, minY=41.0, width=68.0, height=38.0] boundsInParent[minX=216.0, minY=116.0, width=68.0, height=38.0]

- fifth :

  > 旋转

  ```
  Rectangle r = new Rectangle(75, 50, 50, 20);
  r.setEffect(new DropShadow());
  r.getTransforms().addAll(new Translate(150, 75), new Rotate(30));
  ```

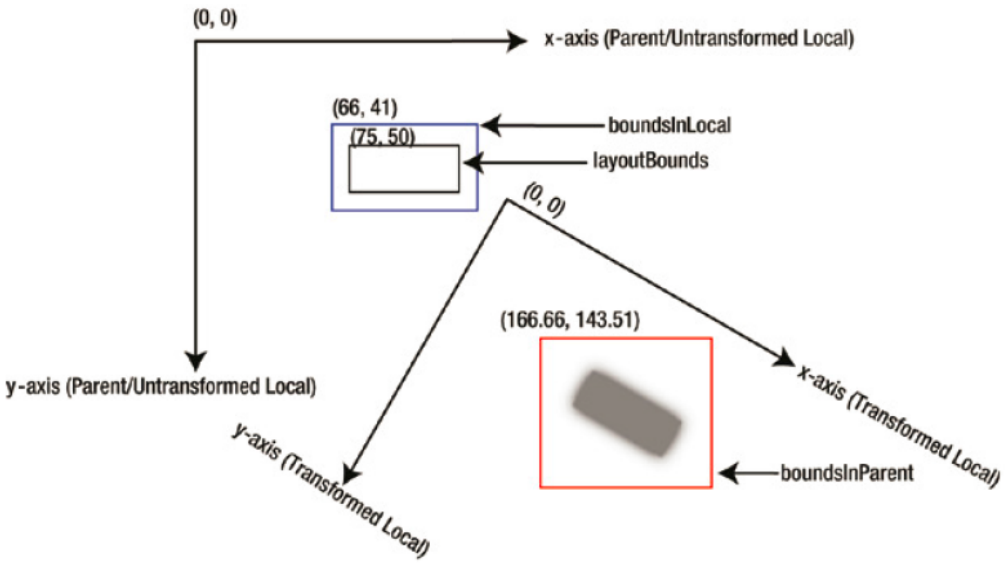

Figure 6-16. *A 50 by 20 rectangle placed at (75, 50) with a drop shadow, a (150, 75) translation, and a 30-degree clockwise rotation*

> layoutBounds[minX=75.0, minY=50.0, width=50.0, height=20.0] boundsInLocal[minX=66.0, minY=41.0, width=68.0, height=38.0] boundsInParent[minX=167.66, minY=143.51, width=77.89, height=66.91]

- sixth :

  > scale and shear : 缩放和裁剪

  ```
  Rectangle r = new Rectangle(75, 50, 50, 20);
  r.setEffect(new DropShadow());
  r.getTransforms().addAll(new Translate(150, 75), new Rotate(30),
  new Scale(1.2, 1.2), new Shear(0.30, 0.10));
  The resulting
  ```

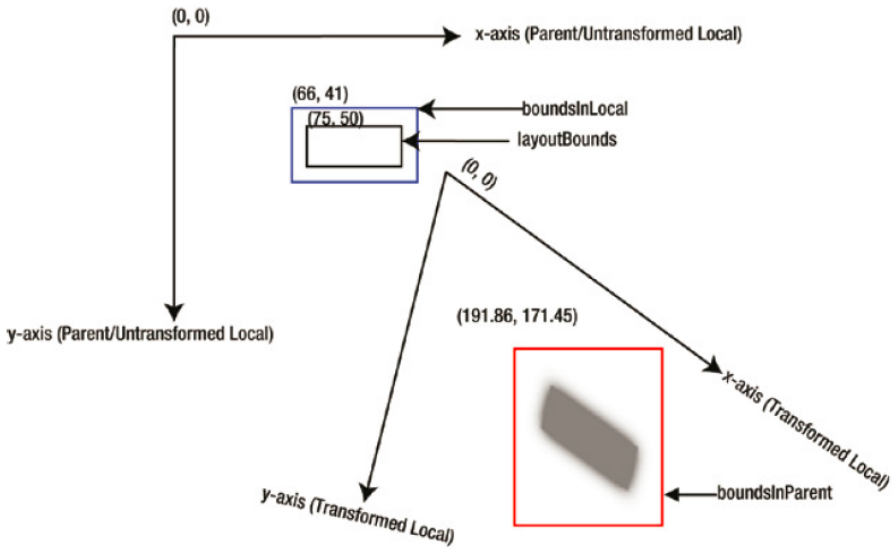

Figure 6-17. *A 50 by 20 rectangle placed at (75, 50) with a drop shadow, a (150, 75) translation, a 30-degree clockwise rotation, a 1.2 in x and y scales, and a 0.30 x shear and 0.10 y shear*

> layoutBounds[minX=75.0, minY=50.0, width=50.0, height=20.0] boundsInLocal[minX=66.0, minY=41.0, width=68.0, height=38.0] boundsInParent[minX=191.86, minY=171.45, width=77.54, height=94.20]

8. Positioning a Node Using layoutX and layoutY

translation

- `layoutX` and `layoutY` properties in Node class
- `translateX` and `translateY` properties in Node class
- `finalTranslationX = layoutX + translateX`
- `finalTranslationY = layoutY + translateY`

> 为什么要使用两种呢?
>
> layout系列提供了稳定的layout
>
> translate系列则提供了动态layout，例如在animation中的应用
>
> **the layoutX and layoutY properties do not specify the final position of a node**

- **relocate(double finalX,double finalY)** of Node class
  - 直接计算并设置到layoutX和layoutY
- wraning : 当你设定layoutX和layoutY时，可能遇到没有效果
  - 因为大多数parent node，比如Region的子类，例如VBox和HBox：use their own positioning policy and they will ignore the **layoutX** and **layoutY** values for their children
  - Pane 或者 Group则可以

9. Setting the Size of a Node

> every node can be resized

- A resizable node can be resized by its parent during a layout. A nonresizable node is not resized by its parent during a layout

  > 关键在于parent node是否能够在放置期间进行自动重置大小
  >
  > Button是放置时由parent设置大小
  >
  > Rectangle则不是
  >
  > 当然：resizable和nonresizable，想修改完全可以通过设置属性来实现

- resizable : Group,Text, Shapes
- nonresizable : Regions, Controls, WebView (由parent来设定大小)
- isResizable method : returns true if is a resizable node

```
package com.javafx.test;

import javafx.application.Application;
```

```
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
public class ResizableNodeTest extends Application
{
    public static void main(String[] args) {
        Application.launch(args);
}
    @Override
    public void start(Stage stage)
    {
        Button btn = new Button("A big button");
        Rectangle rect = new Rectangle(100, 50);
        rect.setFill(Color.WHITE);
        rect.setStrokeWidth(1);
        rect.setStroke(Color.BLACK);
        HBox root = new HBox();
        root.setSpacing(20);
        root.getChildren().addAll(btn, rect);
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.setTitle("Resizable Nodes");
        stage.show();
        System.out.println("btn.isResizable(): " + btn.isResizable());
        System.out.println("rect.isResizable(): " + rect.isResizable());
    }
}
```

> 当改变stage的width时，你就会发现，button开始resize

9.1 Resizable Nodes

- actual size of a resizable node
  - The sizing policy of the container in which the node is placed

    > 放置node的容器的size策略

  - The sizing range specified by the node itself

    > node本身的size range

- 每个容器都有对于孩子的resizing policy

  > 将在 第十章讨论

- 节点的size range :
  - Preferred size : 理想的宽度和高度，以显示其所有内容

    > 首选大小

- Minimum size： 够使用的最小size

  > 最小尺寸，显示省略号文本

- Maximum size： 最大size

  > 对于button，maximum和preferred一样

- 大多数resizable node都会自动根据内容和属性设置计算它们的 preferred,minimum,maximum大小，这些大小被称为内在大小。
  - Region and Control classes define two constants : 用于内在大小
    - USE_COMPUTED： double

      > 节点将自动根据内容和属性设置计算出大小

    - USE_PREF_SIZE : double

      > 如果和preffed大小一样，则设置minmum和maximum

- DoubleProperty properties to define :
  - preWidth
  - preHeight
  - minWidth
  - minHeight
  - maxWidth
  - maxHeight

  > 这些属性默认设置值：USE_COMPUTE_SIZE，意味着自动计算
  >
  > 当然你可以去设置

  ```
  // 设置预设理想和Min和Max
  // 50pixels
  Button btn = new Button("Close");
  btn.setPrefWidth(50);
  btn.setMinWidth(50);
  btn.setMaxWidth(50);
  ```

  ```
  // 内部计算
  Button btn = new Button("Close");
  btn.setMinWidth(Control.USE_PREF_SIZE);
  btn.setMaxWidth(Control.USE_PREF_SIZE);
  ```

  > 需要bind节点大小到一个表达式时，你需要bind the preWidth and preHeight properties

如何获取 preferred，minimum，maximum

- 不应该使用：getPrefWidth,...等
- 因为它们可能被设置为USE_COMPUTE_SIZE等标志值，而使用get则获取到这些标志值而不是真实的size
- 使用下面的方法 in Node class
  - prefWindth(double height)
  - double prefWidth(double height)

    - double prefHeight(double width)
    - double minWidth(double height)
    - double minHeight(double width)
    - double maxWidth(double height)
    - double maxHeight(double width)

> 对于JavaFX中的大多数节点，宽度和高度是独立的。然而，对于某些节点，高度取决于宽度，反之亦然。
>
> 当节点的宽度取决于它的高度，节点被认为具有vertical content bias (垂直内容偏向)
>
> 当节点的高度取决于他的宽度，节点被认为具有horizontal content bias（水平内容偏向）
>
> 然而节点不可能同时拥有 horizontal 和 vertical content biases

    - getContentBias method : returns the content bias of a node
        - return type : **javafx.geometry.Orientation** enum
            - HORIZONTAL

> Labeled class 的子类：Label, Button, CheckBox

            - VERICAL

> 有的content bias取决于它们的定位

        - 然而有的node没有content bias，例如：Text、ChoiceBox
            - 返回null

- 参数说明
    - 如果节点类型没有 content bias，你需要传递 -1

```
ChoiceBox choices = new ChoiceBox();
...
double prefWidth = choices.prefWidth(-1);
double prefHeight = choices.prefHeight(-1);
```

    - 如果有 content bias，你需要传递有 biased dimension
        - 例如 button

```
Button b = new Button("Hello JavaFX");
// Enable text wrapping for the button, which will change its
// content bias from null (default) to HORIZONTAL
b.setWrapText(true);
...
double prefWidth = b.prefWidth(-1); // 获取width
double prefHeight = b.prefHeight(prefWidth);// 根据width获取height
```

> 因为 button是HORIZONTAL content bias，因此需要传递width获取height
>
> 如果没有设置text wrap 为true（没有文本），则不需要传递width，只要传递-1就行了

- 介绍通用方式

```
Node node = get the reference of of the node;
```

```
...
double prefWidth = -1;
double prefHeight = -1;
Orientation contentBias = b.getContentBias();
if (contentBias == HORIZONTAL)
{
    prefWidth = node.prefWidth(-1);
    prefHeight = node.prefHeight(prefWidth);
} else if (contentBias == VERTICAL)
{
    prefHeight = node.prefHeight(-1);
    prefWidth = node.prefWidth(prefHeight);
} else
{
    // contentBias is null
    prefWidth = node.prefWidth(-1);
    prefHeight = node.prefHeight(-1);
}
```

> OK 知道了如何获取标志值和实际值
>
> get the specified values and the actual values for the preferred, minimum, and maximum sizes of a node

- **current size**
  - The Region and Control classes define two read-only properties, width and height, that hold the values for the current width and height of a node

其他方法

Table 6-2. Size-Related Methods of Resizable Nodes

Methods/Properties	Defining Class	Usage

Properties:
`prefWidth`
`prefHeight`
`minWidth`
`minHeight`
`maxWidth`
`maxHeight`
	`Region, Control`	They define the preferred, minimum, and maximum sizes. They are set to sentinel values by default. Use them to override the default values.

Methods:
`double prefWidth(double h)`
`double prefHeight(double w)`
`double minWidth(double h)`
`double minHeight(double w)`
`double maxWidth(double h)`
`double maxHeight(double w)`
	`Node`	Use them to get the actual sizes of nodes. Pass -1 as the argument if the node does not have a content bias. Pass the actual value of the other dimension as the argument if the node has a content bias. Note that there are no corresponding properties to these methods.

Properties:
`width`
`height`
	`Region, Control`	These are *read-only* properties that hold the current width and height of resizable nodes.

Methods:
`void setPrefSize(double w, double h)`
`void setMinSize(double w, double h)`
`void setMaxSize(double w, double h)`
	`Region, Control`	These are convenience methods to override the default computed width and height of nodes.

Methods: `void resize(double w, double h)`	`Node`	It resizes a node to the specified width and height. It is called by the parent of the node during a layout. You should not call this method directly in your code. If you need to set the size of a node, use the `setMinSize()`, `setPrefSize()`, or `setMaxSize()` methods instead. This method has no effect on a nonresizable node.
Methods: `void autosize()`	`Node`	For a resizable node, it sets the layout bounds to its current preferred width and height. It takes care of the content bias. This method has no effect on a nonresizable node.

9.2 Nonresizable Nodes

- 通过修改属性改变它们的大小
- 有些方法用于 nonresizable node是没有效果的
  - **resize** method : no effect
  - **preWidth,minWidth**, ...: return **layoutBounds** 相关
- 没有 content bias
- 传递 -1 去获取其他 dimension

10. Storing User Data in a Node

- 每个节点都维护(maintain) 一个observable到用户定义属性的映射 (key/value pairs)

- 用于store任何用户信息
  - **TextField** :

```
// adds a property "origunbalData" with a value "Advik" to a TextField node
TextField nameField = new TextField();
ObservableMap<Object,Object> props = nameField.getProperties();
props.put("originalData","Advik");
// reads the value of "originalData" property
if(props.containsKey("originalData"))
{
    String originalData = (String)props.get("originalData");
}else
{
    // 不存在
}
```

> getProperties : return `ObservableMap<Object, Object>`
>
> 这个方式比较直接，直接获取textField的整个map引用

- 好了，我们现在看其他方法
  - **setUserData** and **getUserData** : store a user-defined value as a property for a node
    - **setUserData** : 参数指定了 类似`ObservableMap<Object, Object>`功能，不过你只要提供value就行了，key由node内部来操作

```
nameField .setUserData("Saved"); // 设置用户数据
...
String userData = (String)nameField.getUserData(); // 获取用户数据
```

  - **hasProperties** method : return true if has properties

```
// JavaFX 2.2存在问题
TextField nameField = new TextField();
System.out.println(nameField.hasProperties());
ObservableMap<Object, Object> props = nameField.getProperties();
System.out.println(nameField.hasProperties());
// prints:
// false
// true
```

11. What Is a Managed Node?

- The Node class has a managed property
  - type : **BooleanProperty**
  - default : all nodes are managed
- The laying out of a managed node is managed by its parent

> 托管节点的布局由parent来管理
>
> parent计算所有管理节点的layouBounds并计算出自己的size

- 如果一个节点未被管理，应用程序只负责将其放置（计算其大小和大小位置）

  > 因此parent不放置未被管理的孩子
  >
  > 未受管节点的layoutBounds的更改不会触发parent或者scene的重置

- An unmanaged parent node acts as a layout root
  - 如果孩子调用 `Parent.requesLayout`, 只有unmanaged parent建立的分支才会被重置
- 什么时候使用 unmanaged node? (一般不使用)
  - 当你要去在container中显示一个node，而且不想让container计算layoutBounds时

    > 这也意味着你必须自己设置size 和 position

  - example

```
package com.javafx.test;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.value.ObservableValue;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;
public class MicroHelpApp extends Application
{
    // An instance variable to store the Text node reference
    private Text helpText = new Text();
    public static void main(String[] args)
    {
        Application.launch(args);
    }
    @Override
    public void start(Stage stage)
    {
        TextField fName = new TextField();
        TextField lName = new TextField();
        TextField salary = new TextField();
        Button closeBtn = new Button("Close");
        closeBtn.setOnAction(e -> Platform.exit());
        fName.getProperties().put("microHelpText", "Enter the first name");
        lName.getProperties().put("microHelpText", "Enter the last name");
        salary.getProperties().put("microHelpText",
        "Enter a salary greater than $2000.00.");

        // The help text node is unmanaged
```

```
        helpText.setManaged(false);
        helpText.setTextOrigin(VPos.TOP);
        helpText.setFill(Color.RED);
        helpText.setFont(Font.font(null, 9));
        helpText.setMouseTransparent(true);
        // Add all nodes to a GridPane
        GridPane root = new GridPane();
        root.add(new Label("First Name:"), 1, 1);
        root.add(fName, 2, 1);
        root.add(new Label("Last Name:"), 1, 2);
        root.add(lName, 2, 2);
        root.add(new Label("Salary:"), 1, 3);
        root.add(salary, 2, 3);
        root.add(closeBtn, 3, 3);
        root.add(helpText, 4, 3);
        Scene scene = new Scene(root, 300, 100);
        // Add a change listener to the scene, so you know when the focus owner
        // changes and display the micro help
        scene.focusOwnerProperty().addListener(
        (ObservableValue<? extends Node> value, Node oldNode, Node newNode)
        -> focusChanged(value, oldNode, newNode));
        stage.setScene(scene);
        stage.setTitle("Showing Micro Help");
        stage.show();
    }
    public void focusChanged(ObservableValue<? extends Node> value,
    Node oldNode, Node newNode)
    {
        // Focus has changed to a new node
        String microHelpText = (String)newNode.getProperties().get("microHelpText");
        if (microHelpText != null && microHelpText.trim().length() > 0)
        {
            helpText.setText(microHelpText);
            helpText.setVisible(true);
            // Position the help text node
            double x = newNode.getLayoutX() +
            newNode.getLayoutBounds().getMinX() -
            helpText.getLayoutBounds().getMinX();
            double y = newNode.getLayoutY() +
            newNode.getLayoutBounds().getMinY() +
            newNode.getLayoutBounds().getHeight() -
            helpText.getLayoutBounds().getMinX();
            //System.out.println("layoutX:" + newNode.getLayoutX());
            //System.out.println("layoutY:" + newNode.getLayoutY());
            //System.out.println("layoutBoundsMinX:" +
newNode.getLayoutBounds().getMinX());
            //System.out.println("layoutBoundsMinY:" +
newNode.getLayoutBounds().getMinY());
            //System.out.println("layoutBounds" + newNode.getLayoutBounds());
            //System.out.println("textBounds" + helpText.getLayoutBounds());
            helpText.setLayoutX(x);
            helpText.setLayoutY(y);

            helpText.setWrappingWidth(newNode.getLayoutBounds().getWidth());
```

```
        }
        else
        {
            helpText.setVisible(false);
        }
    }
}
```

- exmaple : bind visibleProperty

```
package com.javafx.test;

import javafx.application.Application;
import javafx.beans.binding.When;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class SlidingLeftNodeTest extends Application
{
    public static void main(String[] args)
    {
        Application.launch(args);
    }
    @Override
    public void start(Stage stage)
    {
        Button b1 = new Button("B1");
        Button b2 = new Button("B2");
        Button b3 = new Button("B3");
        Button visibleBtn = new Button("Make Invisible");
        // Add an action listener to the button to make b2 visible
        // if it is invisible and invisible if it is visible
        visibleBtn.setOnAction(e -> b2.setVisible(!b2.isVisible()));
        // Bind the text property of the button to the visible
        // property of the b2 button
        visibleBtn.textProperty().bind(new When(b2.visibleProperty())
        .then("Make Invisible")
        .otherwise("Make Visible"));
        // Bind the managed property of b2 to its visible property
        b2.managedProperty().bind(b2.visibleProperty());
        HBox root = new HBox();
        root.getChildren().addAll(visibleBtn, b1, b2, b3);
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.setTitle("Sliding to the Left");
        stage.show();
    }
}
```

> b2.managedProperty().bind(b2.visibleProperty());
>
> 当b2 变成 visible时，managed

> 当b2变成 invisible时，unmanaged
>
> HBox不会计算 unmanged大小，并且invisible的东西不会被显示

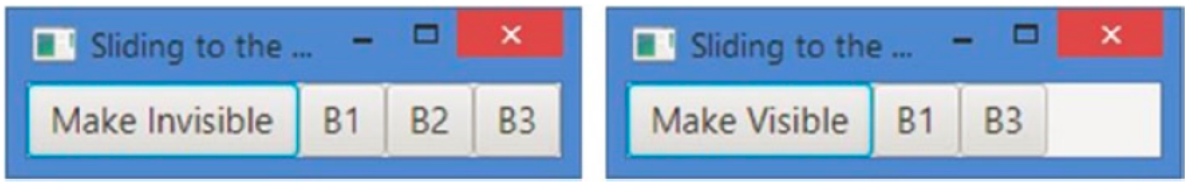

Figure 6-23. *Simulating the slide-left feature for B2 button*

12. Transforming Bounds between Coordinate Spaces

- **LocaltoParent** : transforms a Bounds or a point in the local coordinate space of a node to the coordinate space of its parent
- **LocaltoScene** : transforms ....local to scene
- **parentToLocal** : transforms...parent to local
- **sceneToLocal** :transforms...scene to local

  > 这些都有三种重载版本

- scene 的左上角坐标是参照stage左上角为原点的坐标系统
- stage的左上角坐标是参照screen左上角为原点的坐标系统

Summary

- 所有节点都有自己的坐标系统
- 五个坐标空间（Node，Parent，Scene，Stage，Screen）
- 节点的大小和位置称为 Bounds
- 几何图形（geomety）被 bounding box包裹
- javafx.geometry.Bounds : Bounds是一个抽象类
- javafx.geometry.BoundingBox : BoundingBox是Bounds类的实例类
- Bounds 被设计用于处理 3D space
  - 使用bounding box的最小深度，高度，宽度来封装左上角坐标
  - getMinX, getMinY, getMinZ 用于坐标
  - getWidth, getHeight, getDepth 用于获取三维
  - getMaxX, getMaxY, getMaxZ 用于获取右下角坐标
- Bounds在2D spcae上的情况
  - minX , minY 定义左上角坐标
  - maxX, maxY定义右下角坐标
  - Z坐标和深度值为0
  - isEmpty() : 三维中任何一个为负数，返回true
  - contains(): 检查是否包含其他Bounds，2D point或者3D point

    - interesects(): 检查是否相交其他Bounds,2D point或者3D point
- 三种类型的Bounds（只读）
    - layoutBounds
    - boundsInLocal
    - boundsInParent
- 关键点
    - 如何定义 (minX, minY)
    - 点的坐标是相对于坐标空间定义的
    - 节点几何(geometry)，笔画(stroke)，效果(effect)，剪辑(clip)和转换(transformations)的属性 被包含在特定类型的边界中
- stroke： 非零stroke被认为是节点几何的一部分，用于计算 bounds
- layoutBounds被认为是逻辑边界（不一定对应物理边界）
- boundsInLocal和boundsInParent被认为是物理边界
- 当几何发生变化时，所有类型的边界都会被重新计算

- **layoutBounds**
    - **节点**的 **未转换局部坐标空间(untransformed local)**中 **根据节点的几何属性** 计算得到
    - 不包括： effect，clip，transformations
    - 计算bounding box左上角的坐标根据节点的 resizable 行为有不同的规则
        - **resizable** : 例如 Region，Control，WebView，**左上角始终为 (0,0)**
        - **nonresizable** : 例如 Shape, Circle, Text, Group, 基于geometry属性计算出来
            - 对于Shape，你可以指定 左上角坐标（节点未转换坐标空间）存储于layoutBounds中
    - **layoutBounds的宽度和高度** （平台差异）
        - 有的节点允许你指定宽度和高度
        - 有的节点自动计算并覆盖你设置的宽度和高度
            - 例如 Button由文本和字体指定
    - 如何以layoutBounds的方式 得到deffect和transformations效果呢（足够大空间存放）
        - 放入 Group, 这样就会去计算 Group的 layoutBounds （包括所有children）
    - layoutBounds根据geometry计算出来时，你不能对其进行绑定
- **boundsInLocal**
    - **节点**的 **未转换局部坐标空间(untransformed local)**中
        - 计算在layoutBounds基础之上，如果有还包括effect，clip
        - 不包括 transformations
    - boundsInLocal属性包括按钮周围的投影效果。 请注意，由layoutBounds定义的边界框左上角的坐标为（0.0，0.0），对于boundsInLocal为（-9.0，-9.0）。由于节点的大小是基于运行程序的平台自动计算的，因此在不同平台上的输出可能有点不同
    - 什么时候使用 boundsInLocal
        - 当需要 包含 节点的效果和剪辑时

        - 假设你有一个带有反射的Text节点，并且你想垂直居中。
          如果使用Text节点的layoutBounds，它只会将文本部分居中
          该节点并不包含反射。 如果使用boundsInLocal，它将使文本与其反射中心
        - 另一个例子是检查具有效果的球的碰撞。 如果一个球在包含其效果的另一个球的边界内移动时发生两个球之间的碰撞，请使用boundsInLocal
        - 如果碰撞仅在它们与几何边界相交时才发生，请使用layoutBounds

- **boundsInParent**
    - 位于parent坐标空间中
    - 包括节点几何属性，效果，剪辑和转换。 它很少直接在代码中使用

- **Bounds of Group**
    - Group的layoutBounds，boundsInLocal和boundsInParent的计算与节点的计算不同。
      一个Group承担其Children的计算和。
    - 您可以将效果，剪辑和转换分别应用于组的每个孩子。
      您还可以将效果，剪辑和转换直接应用于组，并将其应用于其所有子节点
    - 组的layoutBounds是其所有子项的boundsInParent的并集
    - 组的boundsInLocal是通过取其layoutBounds和
      包括直接适用于本集团的效果和剪辑。
    - 组的boundsInParent通过将其boundsInLocal和包括直接应用于该组的变换包含在内来计算

- 对于resizable 节点，注意 node的坐标空间和local坐标空间差异 （ 推测
- 对于nonresizable节点，注意 node坐标空间和local坐标空间相同 （ 推测

- 使用**layoutX** and **layoutY** : 平移坐标系统
    - Node类有两个属性，分别是layoutX和layoutY，用于定义**坐标空间沿x轴和y轴的平移**
    - Node类具有translateX和translateY属性，可以做同样的事情

      ```
      finalTranslationX = layoutX + translateX
      finalTranslationY = layoutY + translateY
      ```

    - 使用layoutX和layoutY来定位一个稳定布局的节点。 例如，在动画过程中，使用translateX和translateY来为动态布局定位节点
    - 请务必记住，layoutX和layoutY属性不指定节点的最终位置。它们是应用于节点坐标空间的平移
    - Node类有一个方便的方法，relocate（double finalX，double finalY），将节点定位在（finalX，finalY）位置。 该方法会正确计算并设置layoutX和layoutY值，同时考虑到layoutBounds的minX和minY值。 为了避免错误和节点错位，我更喜欢使用relocate（）方法,而不是setLayoutX（）和setLayoutY（）方法
    - 对于**Region**子类：会忽略孩子的 layoutX和layoutY设置，可以使用Group来代替

      > HBox、VBox等等
      >
      > 平移坐标空间并不发挥作用

- **设置节点大小**
    - 每个节点都可以设置 宽度和高度

- **resizable** node : 节点在布局过程中可以由其父节点调整大小(例如 Button)
- **nonresizable** node： 父节点无法在布局时为他自动调整大小 （例如 Rectangle)
- 可调整大小的节点可以在布局过程中由其父级调整大小,在布局过程中，不可调整的节点不会由其父级调整大小.如果您想调整不可恢复的节点的大小，则需要修改影响其大小的属性。
- 例如，要调整矩形的大小，您需要更改其宽度和高度属性。 Regions，Controls和WebView是可调整大小的节点的示例。 组，文本和形状是不可修复节点的示例
- 你怎么知道一个节点是否是 **resizable**?
  - Node类中的**isResizable**（）方法为可调整大小的节点返回true; 它会为不可恢复的节点返回false
- **Resizable Node**
  - 实际大小由： parent规则（第十章介绍） 和 node自己的范围指定
  - node范围类型
    - Preferred size : 理想大小，足够大显示内容
    - Minimum size： 最小，足够大去显示图像和省略号文本
    - Maximum size : 最大, 甚至可以设置 Double.MAX_VALUE

    > 大多数可调整大小的节点根据其内容和属性设置自动计算 preferred, minimux, maximum, 这些尺寸被称为它们的固有尺寸

    - **Region**和**Control**类定义了两个常量，它们充当节点**固有尺寸**的标记值
      - USE_COMPUTED_SIZE : -1
        - 节点将根据其内容和属性设置自动计算该大小
      - USE_PREF_SIZE : Double.NEGATIVE_INFINITY
        - 用于设置最小和最大尺寸（如果它们与首选尺寸相同）
    - Region和Control类具有六个DoubleProperty类型的属性，用于定义宽度和高度的首选值，最小值和最大值
      - prefWidth
      - prefHeight
      - minWidth
      - minHeight
      - maxWidth
      - maxHeight
      - 默认被设置USE_COMPUTED_SIZE
      - 您可以设置其中一个属性来**覆盖**节点的固有大小。 例如，您可以将按钮的首选宽度，最小宽度和最大宽度设置为50像素，如下所示

        ```
        Button btn = new Button("Close");
        btn.setPrefWidth(50);
        btn.setMinWidth(50);
        btn.setMaxWidth(50);
        ```

      - 以下代码片段将按钮的最小和最大宽度设置为首选宽度，其中首选宽度本身是在内部计算的：

```
Button btn = new Button("Close");
btn.setMinWidth(Control.USE_PREF_SIZE);
btn.setMaxWidth(Control.USE_PREF_SIZE);
```

    - 在大多数情况下，节点的首选，最小和最大尺寸的内部计算值都很好。 只有在不满足应用程序需求的情况下，才可以使用这些属性来覆盖内部计算的大小。 如果您需要将节点的大小绑定到表达式，则需要绑定prefWidth和prefHeight属性

- 获取实际的size属性值
    - getPrefWidth，.... 都是获取标记值或者覆盖值，而非实际
    - preWidth(double), preHeight(double), ... 传递参数获取实际值
        - **horizontal content bias** : 高度依赖于宽度的变化，称为水平内容偏差
        - **vertical content bias** : 宽度依赖于高度的变化，称位垂直内容偏差
        - 获取节点的内容偏差： **getContentBias**,返回枚举
            - 如果没有content，例如Text，ChoiceBox，则返回**null**
        - **什么类拥有水平偏差**?
            - 所有Labeled类的子类： Label, Button, CheckBox, ... 当设置文本 wapping属性时
            - 有的节点由方向(Orientation)决定：FlowPane，方向为水平则为水平内容偏差
        - **具体操作**
            - 没有 content bias 时，传递 -1 获取
            - 有 content bias时 ，传递 相应的 属性获取
            - 如果按钮没有启用文本环绕属性(setWrapText(false))，则可以将-1传递给prefWidth（）和prefHeight（）方法，因为它不会有内容偏倚

```
Button b = new Button("Hello JavaFX");
// Enable text wrapping for the button, which will change its
// content bias from null (default) to HORIZONTAL
b.setWrapText(true);
...
double prefWidth = b.prefWidth(-1);
double prefHeight = b.prefHeight(prefWidth);
```

            - **通用方法**

```
Node node = get the reference of of the node;
...
double prefWidth = -1;
double prefHeight = -1;
Orientation contentBias = b.getContentBias(); // ***
if (contentBias == HORIZONTAL) {
prefWidth = node.prefWidth(-1);
prefHeight = node.prefHeight(prefWidth);
} else if (contentBias == VERTICAL) {
prefHeight = node.prefHeight(-1);
prefWidth = node.prefWidth(prefHeight);

} else {
```

```
// contentBias is null
prefWidth = node.prefWidth(-1);
prefHeight = node.prefHeight(-1);
}
```

        - Region和Control类定义了两个只读属性，width和 高度，它保存节点当前宽度和高度的值

    - **Noresizable**
        - 在布局过程中，不可修复的节点不会被parent调整大小。 但是，您可以通过更改其属性来更改其大小。
        - Node类中定义了几个与尺寸相关的方法。 这些方法在nonresiable节点上调用时不起作用，或者它们返回当前的大小
        - Node类中的prefWidth（double h），minWidth（double h）和maxWidth（double h）方法返回其layoutBounds宽度; 而prefHeight（double w），minHeight（double w）和maxHeight（double w）方法返回其layoutBounds高度
        - 不可修复的节点没有content bias, 将所有这些方法传递-1作为其他维度的参数

- 保存用户数据
    - **getProperties** method of Node : 返回 `ObservableMap<Object, Object>`

```
TextField nameField = new TextField();
...
ObservableMap<Object, Object> props = nameField.getProperties();
props.put("originalData", "Advik");




ObservableMap<Object, Object> props = nameField.getProperties();
if (props.containsKey("originalData")) {
String originalData = (String)props.get("originalData");
} else {
// originalData property is not set yet
}
```

    - Node类有两个便捷方法，**setUserData**（Object value）和**getUserData**（），用于将用户定义的值存储为节点的属性
        - key 由内部指定

```
nameField.setUserData("Saved"); // Set the user data
...
String userData = (String)nameField.getUserData(); // Get the user data
```

        - 由于它存储在由getProperties（）方法返回的相同ObservableMap中，因此可以通过遍历该映射中的值来间接获取它

- **Managed Node**
    - Node类具有一个managed属性，该属性的类型为BooleanProperty

- 默认情况下，所有节点都是被 managed , 即true。
- managed节点的布局由其父节点进行管理
- 父节点在计算其自身大小时会考虑其所有managed子项的layoutBounds
- 当管理的孩子的layoutBounds发生变化时，scene的相关部分将被重新输出

- **Unmanaged Node**
  - 应用程序全权负责布局（计算其大小和位置）
  - 父节点不会布置其unmanaged子项
  - unmanaged 节点的layoutBounds中的更改不会触发它上面的重新布局
  - 如果一个子节点调用Parent.requestLayout（）方法，只有以 unmanaged 父节点为根的分支才会被重新布局
  - 父节点在布局时也会布局 invisible 节点，但是会不管 unmanaged 节点 （可以将其进行绑定)
  - 有时，如果节点变得不可见，您可能需要使用由节点使用的空间。 假设你有一个带有几个按钮的HBox。 当其中一个按钮变为不可见时，您需要从右向左滑动所有按钮。 您可以在VBox中实现滑动效果。 在HBox和VBox（或任何其他具有相对定位的容器）中实现滑动效果很容易，方法是将节点的托管属性绑定到 可见的属性

    ```
    b2.managedProperty().bind(b2.visibleProperty());
    ```

- **坐标空间的转换**
  - 您可能需要将一个边界或一个点从一个坐标空间转换为另一个坐标空间。 **Node**类包含几种方法来支持这一点
  - Local to parent ：**localToParent**
  - Local to scene : **localToScene**
  - Parent to local : **parentToLocal**
  - Scene to local : **sceneToLocal**
  - 所有方法都有三个重载版本; 一个版本将Bounds作为参数并返回转换后的Bounds; 另一个版本将Point2D作为参数并返回转换后的Point2D; 另一个版本采用点的x和y坐标并返回转换的Point2D

```
double nodeMinX = newNode.getLayoutBounds().getMinX();
double nodeMinY = newNode.getLayoutBounds().getMinY();
Point2D nodeInScene = newNode.localToScene(nodeMinX, nodeMinY);
Point2D nodeInMarkerLocal = marker.sceneToLocal(nodeInScene);
Point2D nodeInMarkerParent = marker.localToParent(nodeInMarkerLocal);
// Position the circle approperiately
marker.relocate(nodeInMarkerParent.getX()
+ marker.getLayoutBounds().getMinX(),
nodeInMarkerParent.getY()
+ marker.getLayoutBounds().getMinY());
```