## 0.QML的运行

首先，有两种方法来运行qml文件1.使用qt安装目录下面的bin目录的qmlscene.exe来运行一个qml文件，好处是

该qml文件不需要Window元素作为根元素也是可以的。而且可以随时运行，不需要生成任何文件2.使用QtCreator运行，需要在一个完整的qml项目中运行，而且入口是main.qml

而且必须一window元素作为根元素，否则不可见，这个方法的好处是可以生成一个exe文件。

## 1.Rectangle元素：他的父元素可以是一个window元素

常见属性：x，y，width，height，anchors(有很多子属性：anchors.horizontalCenter，anchors.verticalCenter，anchors.centerIn，anchors.fill)

border(有很多子属性: border.color, border.width,还有一些事件处理方法)，roration：旋转角度，scale：缩放控件小于1是缩小，大于1是放大，

antialiasing：反锯齿，默认是true，radius：控制圆角的大小，gradient: 颜色渐变效果

实例：

|  |
| --- |
|  |

|  |  |
| --- | --- |
| import QtQuick 2.12  import QtQuick.Window 2.12  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle{  // x:20  // y:20  width: 300  height: 200  border.color: "pink"  border.width: 5  //rotation: 90  //scale: 0.5 //小于1是缩小，大于1是放大  //antialiasing: false  gradient: Gradient { //颜色渐变效果  GradientStop { position: 0.0; color: "lightsteelblue" }  GradientStop { position: 1.0; color: "blue" }  }  radius: 20  anchors.centerIn: *parent* //在父窗体中居中显示  //anchors.horizontalCenter: parent.horizontalCenter //在父窗体中水平居中显示  //anchors.verticalCenter: parent.verticalCenter //在父窗体中垂直居中显示  Image {  x:80  y:30  id: *img1*  focus: true  width: 100  height: 100  source: "gkiss.jpg"  Keys.onReturnPressed:*console*.log("enter pressed") //响应回车键按下, 接收该事件的对象必须处于聚焦状态focus:ture  //anchors.fill: parent  }  // MouseArea{ //鼠标点击事件  // anchors.fill: parent  // //onClicked: console.log("clicked")  // onClicked: {img1.width +=10;img1.height +=10}  // }  }  Rectangle{  id:*rec1*  Text {  id: *name1*  text: *qsTr*("变大")  anchors.centerIn: *parent*  }  x:210  y:350  width: 100  height: 50  color: 'purple'  MouseArea{ //鼠标点击事件  anchors.fill: *parent*  //onClicked: console.log("clicked")  onClicked: {*img1*.width +=10;*img1*.height +=10}  }  }  //anchors的上下左右属性是用来确定位置的，anchors.fill属性是用来填充的  Rectangle{  id:*rec2*  Text {  id: *name2*  anchors.centerIn: *parent*  text: *qsTr*("变小")  }  width: 100  height: 50  anchors.bottom: *rec1*.bottom  anchors.left: *rec1*.right  anchors.leftMargin: 20  color: 'blue'  MouseArea{ //鼠标点击事件  anchors.fill: *parent*  //onClicked: console.log("clicked")  onClicked: {*img1*.width -=10;*img1*.height -=10}  }  }  } |  |

#### .创建自定义Rectangle：

|  |  |
| --- | --- |
|  |  |
| //main.qml  import QtQuick 2.12  import QtQuick.Window 2.12  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  CustomRect{ //可以直接引用在同级目录下面的qml文件  width:300  height: 200  myTopMargin: 10  myBottomMargin: 10  anchors.centerIn: *parent*  }  } | //CustomRect.qml  import QtQuick 2.0  Rectangle {  id:*borderRect*  property int myTopMargin:0  property int myBottomMargin:0  color: "purple"  Rectangle{  id:*innerRect*  color: "green"  anchors.fill: *parent*  anchors.topMargin: *myTopMargin*  anchors.leftMargin: 5  anchors.rightMargin: 5  anchors.bottomMargin: *myBottomMargin*  }  } |

## 2.states 和 transitions 动画效果制作

states是一个列表，里面有一些状态，它是当我们设置一个控件颜色的state属性的时候被触发，实现预先定义好的功能

例如，在CustomRect.qml中添加states，并且设置state属性，效果如下

|  |  |
| --- | --- |
| //CustomRect.qml  import QtQuick 2.0  Rectangle {  id:*borderRect*  property int myTopMargin:0  property int myBottomMargin:0  color: "purple"  Rectangle{  id:*innerRect*  color: "green"  anchors.fill: *parent*  anchors.topMargin: *myTopMargin*  anchors.leftMargin: 5  anchors.rightMargin: 5  anchors.bottomMargin:*myBottomMargin*  state: "blue\_color" //这个名称必须是下面的其中之一  states: [  State {  name: "red\_color"  PropertyChanges {  target: *innerRect*;  color:"red"  }  },  State {  name: "blue\_color"  PropertyChanges {  target: *innerRect*;  color:"blue"  }  }  ]  }  } |  |

### 还可以通过鼠标点击来修改状态：

|  |  |
| --- | --- |
| //  import QtQuick 2.0  Rectangle {  id:*borderRect*  property int myTopMargin:0  property int myBottomMargin:0  color: "purple"  Rectangle{  id:*innerRect*  color: "green"  anchors.fill: *parent*  anchors.topMargin: *myTopMargin*  anchors.leftMargin: 5  anchors.rightMargin: 5  anchors.bottomMargin:*myBottomMargin*  //state: "red\_color"  states: [  State {  name: "red\_color"  PropertyChanges {  target: *innerRect*;  color:"red"  }  },  State {  name: "blue\_color"  PropertyChanges {  target: *innerRect*;  color:"blue"  }  }  ]  MouseArea{  anchors.fill: *parent*  onPressed: {  *innerRect*.state = "red\_color"  }  onReleased: {  *innerRect*.state = "blue\_color"  }  }  }  } |  |

### 可以使用State的when属性来指定上面事件可以触发状态改变

|  |  |
| --- | --- |
| //main.qml  import "./MyColors.js" as *ColorGen*  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Rectangle{  id:*rect*  width: 300  height: 200  color: "pink"  states: State{ //这里有多个状态的话用[],只有一个就直接用State元素  when:*mArea*.pressed  PropertyChanges {  target: *rect*  color: *it*.randClr()  }  }  MouseArea{  id:*mArea*  anchors.fill: *parent*  }  }  }  } | //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果：

|  |  |  |
| --- | --- | --- |
| 初始状态 | 在矩形里面按下鼠标不松会变色 | 松开鼠标会还原 |

### animation属性：可以实现一些动画效果，如颜色慢慢变成绿色，透明度由透明慢慢变为不透明

### PropertyAnimation，NumberAnimation

|  |  |
| --- | --- |
| import QtQuick 2.12  import QtQuick.Window 2.12  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  // CustomRect{ //可以直接引用在同级目录下面的qml文件  // width:300  // height: 200  // myTopMargin: 10  // myBottomMargin: 10  // anchors.centerIn: parent  // }  Rectangle{  id:*flashing*  width: 75;height: 75  color: "blue"  opacity: 1.0  MouseArea{  anchors.fill: *parent*  onClicked: {  *animateColor*.start()  *animateOpacity*.start()  }  }  PropertyAnimation  {  id: *animateColor*;  target:*flashing*;  properties:"color";  to:"green";  duration:2000  }  NumberAnimation {  id: *animateOpacity*  target:*flashing*  property: "opacity"  from:0.09  to:1.0  duration: 5000  easing.type: Easing.InOutQuad  }  }  } |  |

**注意，在PropertyAnimation的后面跟 on 再跟属性名称，里面的效果会在窗体启动的时候自动调用，没有on需要在MouseArea的onClicked事件中触发。**

### 还有SequentialAnimation按顺序的动画：先做a效果，再做b效果

|  |  |
| --- | --- |
| import QtQuick 2.12  import QtQuick.Window 2.12  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  // CustomRect{ //可以直接引用在同级目录下面的qml文件  // width:300  // height: 200  // myTopMargin: 10  // myBottomMargin: 10  // anchors.centerIn: parent  // }  Rectangle{  id:*flashing*  width: 75;height: 75  color: "blue"  opacity: 1.0  MouseArea{  anchors.fill: *parent*  onClicked: {  *animateColor*.start()  *animateOpacity*.start()  *animateWidth*.start()  *animateHeight*.start()  *animateX*.start()  }  }  //直接方法  PropertyAnimation{  id: *animateColor*;  target:*flashing*;  properties:"color";  to:"green";  duration:2000  }  NumberAnimation {  id: *animateOpacity*  target:*flashing*  property: "opacity"  from:0.09  to:1.0  duration: 5000  easing.type: Easing.InOutQuad  }  NumberAnimation {  id: *animateWidth*  target:*flashing*  property: "width"  from:75  to:200  duration: 3000  easing.type: Easing.InOutQuad  }  NumberAnimation {  id: *animateHeight*  target:*flashing*  property: "height"  from:75  to:200  duration: 3000  easing.type: Easing.InOutQuad  }  NumberAnimation {  id: *animateX*  target:*flashing*  property: "x"  from:0  to:100  duration: 3000  easing.type: Easing.InOutQuad  }  //后面有on的动画会自动执行  PropertyAnimation on width{  to:300  duration:2000  }  PropertyAnimation on height{  to:300  duration:2000  }  // PropertyAnimation on color{  // to:"yellow"  // duration:1000  // }  ColorAnimation on color{  to:"orange"  duration:1000  }  SequentialAnimation on color{  ColorAnimation {  to: "black"  duration: 1000  }  ColorAnimation {  to: "green"  duration: 1000  }  ColorAnimation {  to: "purple"  duration: 1000  }  ColorAnimation {  to: "deeppink"  duration: 1000  }  ColorAnimation {  to: "yellow"  duration: 1000  }  ColorAnimation {  to: "cyan"  duration: 1000  }  ColorAnimation {  to: "orange"  duration: 1000  }  ColorAnimation {  to: "red"  duration: 1000  }  }  }  } | **效果矩形会慢慢变大，颜色会按照**  SequentialAnimation的先后**顺序改变** |

### 小扩展：可以使得矩形按照左→右，上→下，右→左，下→上移动的SequentialAnimation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle{  id:*rect*  color: "deeppink"  width: 100  height: 100  SequentialAnimation{  running: true  //无限循环播放  loops：**Animation.Infinite**  NumberAnimation{  target: *rect*  property: "x"  to:300  duration: 1000  running: true  }  NumberAnimation{  target: *rect*  property: "y"  to:300  duration: 1000  running: true  }  NumberAnimation{  target: *rect*  property: "x"  to:0  duration: 1000  running: true  }  NumberAnimation{  target: *rect*  property: "y"  to:0  duration: 1000  running: true  }  }  }  } | |  |  | | --- | --- | |  |  | |  |  | |

**：修改颜色有多种方法：可以使用**ColorAnimation on color，SequentialAnimation on color，PropertyAnimation on color

### ParallelAnimation

|  |  |
| --- | --- |
| import "./MyColor.js" as *Mycolor*  import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  //并行动画  Rectangle {  id: *rect*  width: 100; height: 100  property var ranclr:*Mycolor*.randomColorFactory(*Qt*.rgba)  color: *ranclr*()  ParallelAnimation {  running: true  loops:Animation.Infinite  NumberAnimation { target: *rect*; property: "x"; to: 300; duration: 1000 }  NumberAnimation { target: *rect*; property: "y"; to: 300; duration: 1000 }  ColorAnimation { target:*rect*;property:"color";to:*rect*.ranclr(); duration: 500}  }  }  } | 矩形会变色和沿着对角线移动到（300,300处）  不断重复 |

### AnchorAnimation

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  //AnchorAnimation  Item {  id: *container*  width: 400; height: 200  Rectangle {  id: *myRect*  width: 100; height: 100  color: "red"  }    states: State {  name: "reanchored"  AnchorChanges { target: *myRect*; anchors.right: *container*.right }  }  transitions: Transition {  // smoothly reanchor myRect and move into new position  AnchorAnimation { loops:Animation.Infinite;duration: 1000 }  }  Component.onCompleted: *container*.state = "reanchored"  }  } | 矩形会一直移动到200处有返回然后继续。。。。 |

### RotationAnimation

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  // RotationAnimation  Image {  id: *img*  anchors.centerIn: *parent*  width: 200  height: 200  source: "./wmill.jpg"  RotationAnimation{  id:*rotate*  loops: Animation.Infinite  target: *img*  from:0  to:90  duration: 1000  //running: true //自动运行  }  MouseArea{  anchors.fill: *parent*  onClicked: *rotate*.running=true //鼠标点击启动运行  }  }  } | 初始状态    点击风车，它会转动 |

### OpacityAnimator

|  |
| --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle {  id: *opacityBox*  width: 50  height: 50  color: "lightsteelblue"  OpacityAnimator {  target: *opacityBox*;  from: 0;  to: 1;  duration: 3000  running: true  loops:Animation.Infinite  }  } |

### ParentAnimation

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item {  width: 200; height: 100  Rectangle {  id: *redRect*  width: 100; height: 100  color: "red"  }  Rectangle {  id: *blueRect*  x: *redRect*.width  width: 50; height: 50  color: "blue"  states: State {  name: "reparented"  ParentChange { target: *blueRect*; parent: *redRect*; x: 10; y: 10 }  }  transitions: Transition {  ParentAnimation {  NumberAnimation { properties: "x,y"; duration: 1000 }  }  }  MouseArea { anchors.fill: *parent*; onClicked: *blueRect*.state = "reparented" }  }  } | 初始状态    效果：点击一下蓝色矩形，它就会移动到红色矩形里面 |

### PathInterpolator,比较复杂

|  |  |
| --- | --- |
| Rectangle {  id: *rootRect*  width: 320  height: 480  color:"deeppink"  Canvas {  id: *canvas*  anchors.fill: *parent*  antialiasing: true  onPaint: {  var *context* = *canvas*.getContext("2d")  *context*.clearRect(0, 0, *width*, *height*)  *context*.strokeStyle = "black"  *context*.path = *motionPath*.path  *context*.stroke()  }  }  //! [0]  PathInterpolator {  id: *motionPath*  path: Path {  startX: 50; startY: 50  PathCubic {  x: *rootRect*.width - 50  y: *rootRect*.height - 50  control1X: *x*; control1Y: 50  control2X: 50; control2Y: *y*  }  onChanged: *canvas*.requestPaint()  }  SequentialAnimation on progress {  running: true  loops: -1  PauseAnimation { duration: 1000 }  NumberAnimation {  id: *progressAnim*  running: false  from: 0; to: 1  duration: 2000  //easing.type: Easing.InQuad  //easing.type: Easing.OutQuad  easing.type: Easing.InOutQuad  }  }  }  //! [0]  Rectangle {  id: *box*  width: 50; height: 50  border.width: 1  antialiasing: true  //bind our attributes to follow the path as progress changes  x: *motionPath*.x  y: *motionPath*.y  rotation: *motionPath*.angle  transform: Translate { x: -*box*.width/2; y: -*box*.height/2 }  Text {  anchors.centerIn: *parent*  text: "Box"  }  }  } |  |

### SpringAnimation

|  |  |
| --- | --- |
| Item {  width: 300; height: 300  Rectangle {  id: *rect*  width: 50; height: 50  color: "purple"  Behavior on x { SpringAnimation { spring: 2; damping: 0.2 } }  Behavior on y { SpringAnimation { spring: 2; damping: 0.2 } }  }  MouseArea {  anchors.fill: *parent*  onClicked: {  *rect*.x = *mouse*.x - *rect*.width/2  *rect*.y = *mouse*.y - *rect*.height/2  }  }  } | 初始状态    点击一次矩形，他会弹跳一下    点击三个角的其中一个，它会朝着对应的方向弹跳 |

### SmoothAnimation

|  |  |
| --- | --- |
| import "./MyColor.js" as *Mycolor*  import QtQuick 2.15  import QtQuick.Window 2.15  Window {  id:*win*  width: 800  height: 600  visible: true  title: *qsTr*("Hello World")  Rectangle {  width: 800; height: 600  color: "blue"  Rectangle {  width: 60; height: 60  x: *rect1*.x - 5; y: *rect1*.y - 5  color: "green"  Behavior on x { SmoothedAnimation { velocity: 200 } }  Behavior on y { SmoothedAnimation { velocity: 200 } }  }  Rectangle {  id: *rect1*  width: 50; height: 50  color: "red"  }  focus: true  Keys.onRightPressed: *rect1*.x = *rect1*.x + 100  Keys.onLeftPressed: *rect1*.x = *rect1*.x - 100  Keys.onUpPressed: *rect1*.y = *rect1*.y - 100  Keys.onDownPressed: *rect1*.y = *rect1*.y + 100  }  } | 初始状态    按下键盘的↑↓←→键，红色的正方形会朝着对应的方向弹跳，然后绿色的矩形会跟着往同一方向弹跳 |

### ScriptAction

|  |  |
| --- | --- |
| //main.qml  import "./MyColor.js" as *Mycolor*  import "./changeclr.js" as *Chgcl*  import QtQuick 2.15  import QtQuick.Window 2.15  Window {  id:*win*  width: 800  height:600  visible: true  title: *qsTr*("Hello World")  //ScriptAction  Rectangle {  id:*rootRect*  width: 300  height: 200  function *changeColor*(){  let *clrFactory* = *Mycolor*.randomColorFactory(*Qt*.rgba)  *rootRect*.color = *clrFactory*()  }  Rectangle {  width: 60; height: 60  color: "green"  SequentialAnimation {  id:*sAnim*  ScriptAction{  script:*Chgcl*.changeColor()//这里也可以是js语句  }  NumberAnimation{  target: *rootRect*  properties: "x,y"  to:*rootRect*.width  duration: 1000  }  ScriptAction{  script:*rootRect*.changeColor()  }  NumberAnimation{  target: *rootRect*  properties: "x,y"  to:0  duration: 1000  }  }  MouseArea{  anchors.fill: *parent*  onClicked: {  *sAnim*.running = true  }  }  }  }  } | //MyColor.js  //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |
| //changclr.js  function *changeColor*(){  let *clrFactory* = Mycolor.randomColorFactory(*Qt*.rgba)  rootRect.color = *clrFactory*()  } | 初始状态    点击一下绿色矩形会修改他的父控件（也是一个矩形）的颜色，并且两个矩形会沿着对角线移动下来由反弹回去      以后每点击一次小矩形两个矩形都会移动和反弹并且修改大矩形的颜色 |

### 注意：ScriptAction中的script可以直接给一个js语句，例如，动态修改透明度

如：将main.qml的代码修改一下

|  |  |
| --- | --- |
| import "./MyColor.js" as *Mycolor*  import "./changeclr.js" as *Chgcl*  import QtQuick 2.15  import QtQuick.Window 2.15  Window {  id:*win*  width: 800  height:600  visible: true  title: *qsTr*("Hello World")  //ScriptAction  Rectangle {  id:*rootRect*  width: 300  height: 200  function *changeColor*(){  let *clrFactory* = *Mycolor*.randomColorFactory(*Qt*.rgba)  *rootRect*.color = *clrFactory*()  }  Rectangle {  width: 60; height: 60  color: "green"  SequentialAnimation {  id:*sAnim*  ScriptAction{  //script:Chgcl.changeColor()  script:*rootRect*.opacity =(*rootRect*.opacity === 1 ? 0.1 : 1)  }  NumberAnimation{  target: *rootRect*  properties: "x,y"  to:*rootRect*.width  duration: 1000  }  ScriptAction{  script:*rootRect*.changeColor()  }  NumberAnimation{  target: *rootRect*  properties: "x,y"  to:0  duration: 1000  }  }  MouseArea{  anchors.fill: *parent*  onClicked: {  *sAnim*.running = true  }  }  }  } |  |

### transition是状态改变的渐变效果，如果没有它，状态是一下子就改变了，有了它可以设置渐变效果已经渐变时间

|  |
| --- |
| Rectangle {  width: 75; height: 75  id: *button*  state: "RELEASED"  MouseArea {  anchors.fill: *parent*  onPressed: *button*.state = "PRESSED"  onReleased: *button*.state = "RELEASED"  }  states: [  State {  name: "PRESSED"  PropertyChanges { target: *button*; color: "blue"}  },  State {  name: "RELEASED"  PropertyChanges { target: *button*; color: "lightsteelblue"}  }  ]  transitions: [  Transition {  from: "PRESSED"  to: "RELEASED"  ColorAnimation { target: *button*; duration: 3000}  },  Transition {  from: "RELEASED"  to: "PRESSED"  ColorAnimation { target: *button*; duration: 2000}  }  ]  }  效果，在矩形里面点击鼠标颜色会慢慢变成蓝色，松开鼠标颜色会慢慢变为浅蓝色 |

**还有behavior和其他但是在实际的应用在state和transition来触发动画效果是比较常用的方式**

### 状态机，SwipeView改变按钮的状态和功能

可以实现同一个按钮完成不同的功能

|  |  |
| --- | --- |
| //main.qml  import "./MyColors.js" as *ColorGen*  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  //状态机，通过SwipeView改变按钮的状态和功能  SwipeView{  id:*swv*  anchors.top: *parent*.top  anchors.left: *parent*.left  anchors.right: *parent*.right  anchors.bottom: *btn*.top  Repeater{  model:["home","about","help"]  Rectangle{  color: *it*.randClr()  }  }  }  PageIndicator{  count:*swv*.count  currentIndex: *swv*.currentIndex  anchors.horizontalCenter: *parent*.horizontalCenter  anchors.bottom: *swv*.bottom  }  Button{  id:*btn*  anchors.bottom: *parent*.bottom  states: [  State {  when:*swv*.currentIndex ===0  PropertyChanges {  target: *btn*  text:"home"  onClicked:*console*.log("home page selected...")  }  },  State {  when:*swv*.currentIndex ===1  PropertyChanges {  target: *btn*  text:"about"  onClicked:*console*.log("about page selected...")  }  },  State {  when:*swv*.currentIndex ===2  PropertyChanges {  target: *btn*  text:"help"  onClicked:*console*.log("help page selected...")  }  }  ]  }  }  } | //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |
| 初始状态 | |  |  |  | | --- | --- | --- | | 往右滑动 | 往右再滑动 | 往左，再往左 | |

每一个状态的按钮的文本和实现的功能不一样

## 3.Button

#### 实例，

button和设置它的样式，注意：QtQuick.Controls.Styles 1.4有一个叫做ButtonStyle的样式在2.15版本已经过时了，如果需要使用，必须和QtQuick.Controls 1.14一起使用

|  |  |
| --- | --- |
| //main.qml  import "./MyColors.js" as *ColorGen*  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Button{  id:*control*  text: "Button1"  background: Rectangle { //可以通过设置按钮是背景属性来设置按钮样式  implicitWidth: 100  implicitHeight: 25  border.width: *control*.activeFocus ? 2 : 1  border.color: "#888"  radius: 4  gradient: Gradient {  GradientStop { position: 0 ; color: *control*.pressed ? "#ccc" : "#eee" }  GradientStop { position: 1 ; color: *control*.pressed ? "#aaa" : "#ccc" }  }  }  }  }  } | //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果

|  |  |
| --- | --- |
| 点击前的样式 | 点击后的样式 |

#### Qt助手的自定义按钮样例

打开Qt助手，搜索**Custom，**会找很多自定义控件的例子，找到**Customizing Button，**将代码复制

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  Button {  id: *control*  x:10  text: *qsTr*("Button")  contentItem: Text {  text: *control*.text  font: *control*.font  opacity: *enabled* ? 1.0 : 0.3  color: *control*.down ? "#17a81a" : "#21be2b"  horizontalAlignment: Text.AlignHCenter  verticalAlignment: Text.AlignVCenter  elide: Text.ElideRight  }  background: Rectangle {  implicitWidth: 100  implicitHeight: 40  opacity: *enabled* ? 1 : 0.3  border.color: *control*.down ? "#17a81a" : "#21be2b"  border.width: 1  radius: 2  }  }  }  } | 点击之前    点击后    连续点击会有闪烁效果 |

#### 将官方自定义按钮该为自己的样式

**在上面的基础上做一点点修改，可以得到下面的效果**

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:ColorGen.randomColorFactory(*Qt*.rgba)  Button {  x:10  id: *control*  text: *qsTr*("Button")  contentItem: Text {  text: *control*.text  //font: control.font  font.bold: *control*.down ?true :false  font.pixelSize: *control*.down ?18:14  opacity: *enabled* ? 1.0 : 0.3  color: *control*.down ? "orange" : "#21be2b"  horizontalAlignment: Text.AlignHCenter  verticalAlignment: Text.AlignVCenter  elide: Text.ElideRight  }  background: Rectangle {  implicitWidth: 100  implicitHeight: 40  opacity: *enabled* ? 1 : 0.3  color: *control*.down ? "deeppink" :"transparent"  border.color: *control*.down ? "deeppink" : "#21be2b"  border.width: 1  radius: 2  }  }  }  } | 点击之前    鼠标按下不松，    鼠标松开会还原    连续点击有一点点类似动画的效果 |

#### 注意：如果你想直接设置按钮的文本颜色可以设置按钮的palette.buttonText属性。

|  |
| --- |
|  |

#### 如果将按钮的flat属性设置为true，那么它的背景就会隐藏，不过按下的时候会出现背景

#### 按钮的palette.dark属性的颜色会覆盖高亮色

|  |
| --- |
|  |

#### 按钮的palette.button属性可以设置按钮的背景色

|  |
| --- |
|  |

#### 自己封装的自定义按钮

其实，使用官方样例改一改就可以了

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  MyStyleButton{  text: "Custom Button"  labelColor: "yellow"  backColor: "lightsteelblue"  clickColor: "red"  }  }  } | //MyStyleButton.qml  import QtQuick 2.0  import QtQuick .Controls 2.15  Button{//按钮的palette.button属性修改按钮的背景色，palette.buttonText属性修改按钮的文本颜色  property string labelColor  property string clickColor  property string backColor  palette.buttonText: *labelColor*? *labelColor*: *palette*.buttonText  palette.button: *backColor* ? *backColor* : *palette*.button  palette.dark: *clickColor* ? *clickColor* : *palette*.dark  // palette.highlight: clickColor ? clickColor :palette.highlight  } |

## 4.Component和Loader

**Component是组件的意思，QML中所有的元素都是Component，它最常用的信号是onCompleted，组件加载完成时触发**

**还有一个on**[Destruction](qml-qtqml-component.html#destruction-signal) ，在组件被销毁的时候调用

**实例：**

|  |
| --- |
|  |

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Component.onCompleted: *console*.log("finished....")//window 本身就有一个Component  Component.onDestruction: *console*.log("deconstrution...")  Item {  id: *item1*  width: 200  height:200  Component{  id:*redSquare*  Rectangle{  color: "red"  width: 100  height: 100  }  }  // Loader{ //Component需要Loader来加载，否则不会显示  // sourceComponent: redSquare  // }  Loader{ //Component需要Loader来加载，否则不会显示,有多个Loader就加载多次  sourceComponent: *redSquare*; x:100  }  }  } |  |

### 注意:可以使用loader来加载其他qml文件，此时需要将sourceComponent修改为source属性值输入qml文件名称

**如：**

|  |  |
| --- | --- |
| **//main.qml**  import QtQuick 2.15  import QtQuick.Window 2.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Component.onCompleted: *console*.log("finished....")//window 本身就有一个Component  Component.onDestruction: *console*.log("deconstrution...")  Loader{  source: "./CustomRect.qml" //可以使用Loader直接加载qml文件里面的组件  }  } | **//CustomRect.qml**  import QtQuick 2.0  Rectangle {  id:*borderRect*  width: 300  height: 300  property int myTopMargin:0  property int myBottomMargin:0  color: "purple"  Rectangle{  id:*innerRect*  color: "green"  anchors.fill: *parent*  anchors.topMargin: *myTopMargin*  anchors.leftMargin: 5  anchors.rightMargin: 5  anchors.bottomMargin:*myBottomMargin*  //state: "red\_color"  states: [  State {  name: "red\_color"  PropertyChanges {  target: *innerRect*;  color:"red"  }  },  State {  name: "blue\_color"  PropertyChanges {  target: *innerRect*;  color:"blue"  }  }  ]  MouseArea{  anchors.fill: *parent*  onPressed: {  *innerRect*.state = "red\_color"  }  onReleased: {  *innerRect*.state = "blue\_color"  }  }  }  } |

**运行效果**

|  |
| --- |
|  |

### 可以在自定义的组件里面添加onCompleted和onDestruction信号处理函数，然后在主调文件这里添加一个按钮来触发这个事件

**只需要在按钮的点击事件里面将Loader的source文件设置为“”。**

|  |  |
| --- | --- |
| **//main.qml**  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  // Component.onCompleted: console.log("finished....")//window 本身就有一个Component  // Component.onDestruction: console.log("deconstrution...")  // Item {  // id: item1  // width: 200  // height:200  // Component{  // id:redSquare  // Rectangle{  // color: "red"  // width: 100  // height: 100  // }  // }  //// Loader{ //Component需要Loader来加载，否则不会显示  //// sourceComponent: redSquare  //// }  // Loader{ //Component需要Loader来加载，否则不会显示,有多个Loader就加载多次  // sourceComponent: redSquare; x:100  // }  // }  Loader{  id:*ldr*  source: "./CustomRect.qml" //可以使用Loader直接加载qml文件里面的组件  }  Button{ //需要import QtQuick.Controls 2.15  width: 100  height: 60  x:300  onClicked: {  *ldr*.source = ""//卸载组件，会引发组件的onDestruction信号处理函数  }  text: "卸载"  }  } | **//CustomRect.qml**  import QtQuick 2.0  Rectangle {  id:*borderRect*  width: 300  height: 300  property int myTopMargin:0  property int myBottomMargin:0  color: "purple"  Rectangle{  id:*innerRect*  color: "green"  anchors.fill: *parent*  anchors.topMargin: *myTopMargin*  anchors.leftMargin: 5  anchors.rightMargin: 5  anchors.bottomMargin:*myBottomMargin*  //state: "red\_color"  states: [  State {  name: "red\_color"  PropertyChanges {  target: *innerRect*;  color:"red"  }  },  State {  name: "blue\_color"  PropertyChanges {  target: *innerRect*;  color:"blue"  }  }  ]  MouseArea{  anchors.fill: *parent*  onPressed: {  *innerRect*.state = "red\_color"  }  onReleased: {  *innerRect*.state = "blue\_color"  }  }  Component.onCompleted: *console*.log("finished....")//window 本身就有一个Component  Component.onDestruction: *console*.log("destrution...")  }  } |

**效果**

|  |
| --- |
|  |

|  |
| --- |
|  |

### 注意:Loader有一个item属性可以获取它组件的组件，通过修改这个item的属性，就可以改变Loader加载的组件的属性

|  |
| --- |
|  |

**Loader还有一个异步属性，当需要加载的资源很大的时候非常有用**

### 此外，Loader还有一个setSource方法，也可以用来加载qml文件中文组件用的

当使用Loader的source属性来加载一个qml文件作为组件，而这个组件需要参数的时候，source属性就不管用了，这时候需要setSource方法，因为它可以传递参数

|  |  |
| --- | --- |
| **//CustomColor.js**  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } | **//DateTimeLabel.qml**  import QtQuick 2.0  import QtQuick.Controls 2.15  import "./CustomColor.js" as *ColorGen*  Label{  width: 100  height:40  id:*lblDate*  font.bold: true  font.pixelSize: 30  //required property color rndClr  required property var rndClr  color:*rndClr*()  Timer{  repeat: true  triggeredOnStart: true  running: true  interval: 1000  onTriggered: {  *lblDate*.text = *Qt*.formatDateTime(new *Date*(),'yyyy-MM-dd hh:mm:ss')  }  }  } |

|  |  |  |
| --- | --- | --- |
| **//main.qml**  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./CustomColor.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  //Loader扩展  Item {  id: *it*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Loader{id:*loader*}  Component.onCompleted: {  *loader*.setSource("./DateTimeLabel.qml",{rndClr:*it*.randClr})  }  }  } |  |  |

### 其实也可以不使用Loader来加载Component。

**此时需要使用Component的createObject(parent)方法来创建实例，这样子可以实现组件的重复使用**

**可以只写一个Component，然后使用多次**

**实例**

|  |  |
| --- | --- |
| **//CustomColor.js**  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } | **//main.qml**  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./CustomColor.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item {  id: *it*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Component{  id:*com*  Label{  width: 100  height:100  id:*lblDate*  font.bold: true  font.pixelSize: 30  color: *it*.randClr()  Timer{  repeat: true  triggeredOnStart: true  running: true  interval: 1000  onTriggered: {  *lblDate*.text = *Qt*.formatDateTime(new *Date*(),'yyyy-MM-dd hh:mm:ss')  }  }  }  }  Column{  id:*col*  width: 200  height: 60  Component.onCompleted: {  // com.createObject(this)  // com.createObject(this)  *batchCreate*(5)  }  function *batchCreate*(n){  for(var *i*=0;*i*<*n*;++*i*){  *com*.createObject(this)  }  }  }  }  // Item {  // id: it  // Label{  // width: 100  // height:100  // id:lblDate  // font.bold: true  // font.pixelSize: 30  // color: ColorGen.randomColorFactory(Qt.rgba)()  // Timer{  // repeat: true  // triggeredOnStart: true  // running: true  // interval: 1000  // onTriggered: {  // lblDate.text = Qt.formatDateTime(new Date(),'yyyy-MM-dd hh:mm:ss')  // }  // }  // }  // }  //Item{  // id:it  // anchors.fill: parent  // property var randColor : ColorGen.randomColorFactory(Qt.rgba)  // Component{  // id:dateTimeCom  // Label{  // width: 100  // height:100  // id:lblDate  // font.bold: true  // font.pixelSize: 30  // color: it.randColor()  // Timer{  // repeat: true  // triggeredOnStart: true  // running: true  // interval: 1000  // onTriggered: {  // lblDate.text = Qt.formatDate(new Date(),"yyyy-MM-dd hh:mm:ss")  // }  // }  // }  // }  // Column{  // Component.onCompleted: {  // dateTimeCom.createObject()  // }  // }  // }  } |

**效果：**

|  |
| --- |
|  |

## 5.如何在窗体中显示图像

#### 使用Image元素，显示gif动画，使用AnimatedImage元素

**当然也可以把这些元素放到Component里面，然后使用Loader来加载，这样子比较麻烦**

|  |
| --- |
|  |

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  // Component.onCompleted: console.log("finished....")//window 本身就有一个Component  // Component.onDestruction: console.log("deconstrution...")  // Item {  // id: item1  // width: 200  // height:200  // Component{  // id:redSquare  // Rectangle{  // color: "red"  // width: 100  // height: 100  // }  // }  //// Loader{ //Component需要Loader来加载，否则不会显示  //// sourceComponent: redSquare  //// }  // Loader{ //Component需要Loader来加载，否则不会显示,有多个Loader就加载多次  // sourceComponent: redSquare; x:100  // }  // }  // Loader{  // id:ldr  // source: "./CustomRect.qml" //可以使用Loader直接加载qml文件里面的组件  // asynchronous: true //异步加载。当组件非常大的时候有用  // onStatusChanged: console.log("Status=",status)  // }  // Button{ //需要import QtQuick.Controls 2.15  // width: 100  // height: 60  // x:300  // onClicked: {  // ldr.source = ""//卸载组件，会引发组件的onDestruction信号处理函数  // }  // text: "卸载"  // }  // Component{  // id:picbox  // Image{  // source:"./nicescn.jpg"  // }  // }  // Loader{  // sourceComponent: picbox  // }  // Image{  // source:"./nicescn.jpg"  // x:0  // y:0  // width:100  // height:100  // PropertyAnimation on width{  // to:350  // duration: 2000  // }  // PropertyAnimation on height{  // to:350  // duration: 2000  // }  // PropertyAnimation on x {  // to:150  // duration: 2000  // }  // PropertyAnimation on y {  // to:50  // duration: 2000  // }  // }  AnimatedImage{ //用来显示gif动画  id:*ami*  source:"./dogcr.gif"  x:0  y:0  width:100  height:100  PropertyAnimation on width{  to:350  duration: 2000  }  PropertyAnimation on height{  to:350  duration: 2000  }  PropertyAnimation on x {  to:150  duration: 2000  }  PropertyAnimation on y {  to:50  duration: 2000  }  }  Button{ //按钮控件没有color属性，如果想设置按钮的颜色，可以在按钮中嵌套一个矩形，用形填充整个按钮，修改矩形的颜色也就修改了按钮的颜色  id:*btn2*  x:100  y:370  onClicked: {  *ami*.paused = !*ami*.paused  }  Rectangle{  anchors.fill: *parent*  color:"yellow"  Text{  anchors.centerIn: *parent*  text:*qsTr*("play/stop")  color:"blue"  }  }  }  } |  |

## 6.MouseArea，可以接收鼠标点击。默认的接收按钮是左键，不过可以修改。

### 6.1基本使用

|  |
| --- |
|  |

**注意：在qml再MouseArea有onClicked事件，onPressed事件和onReleased事件**

**其中 onClicked = onPressed和onReleased事件的组合，需要如果只需要捕捉鼠标按下事件使用onPressed事件，**

**如果只需要捕捉鼠标弹起事件使用onReleased事件，onClicked只有当这两个事件都触发了才会触发**

### 6.2在MouseArea的按钮按下事件中判断是鼠标那个键按下

|  |
| --- |
| MouseArea{  width: 100  height: 100  x:500  y:350  Rectangle{  anchors.fill: *parent*  color: "green"  }  //设置MouseArea接收左中右键的按下和弹起事件  acceptedButtons: *Qt*.LeftButton|*Qt*.RightButton|*Qt*.MidButton  onPressed:{  if(*pressedButtons* & *Qt*.LeftButton)  {  *console*.log("left button Pressed....")  }  else if(*pressedButtons* & *Qt*.RightButton)  {  *console*.log("Right button Pressed....")  } else if(*pressedButtons* & *Qt*.MidButton){  *console*.log("Middle button Pressed....")  }  }  onReleased: *console*.log("Released...")  } |

### 也可以像下面这么写

|  |
| --- |
|  |

### mouseX 和 mouseY属性

|  |
| --- |
|  |

### pressAndHold属性触发onPressAndHold事件

|  |
| --- |
|  |

### 事件冒泡？

|  |
| --- |
|  |

**注意：一般来说，控件的x，y坐标都是相当于父控件的。如果需要使用全局坐标，可以使用一个叫做mapToGlobal的函数**

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./CustomColor.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  //MouseArea  Item {  id: *it*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Rectangle{  width: 100  height: 100  color:*it*.randClr()  MouseArea{  anchors.fill: *parent*  onClicked: {  *console*.log(*mouse*.x,*mouse*.y,*mapToGlobal*(*mouse*.x,*mouse*.y))  }  }  }  } | //CustomColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

**效果**

|  |
| --- |
|  |

### MouseArea有一个drag.target属性，可以将父控件的id赋值给它，这样子就可以拖动方控件了

实例，可以拖动以及点击会变色的文本

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./CustomColor.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  //MouseArea  Item {  id: *it*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Text {  id: *txt*  font.bold: true  font.pointSize: 16  color: *it*.randClr()  text: *qsTr*("拖我走")  MouseArea{  anchors.fill: *parent*  onClicked: {  *console*.log(*txt*.text)  *txt*.color = *it*.randClr()  }  drag.target: *txt*  }  }  }  } | //CustomColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果：

|  |  |  |
| --- | --- | --- |
| //初始状态 | //点击一下 | //拖动 |

### 默认情况下，MouseArea是没有事件穿透的功能的，

如果需要它有事件穿透，也就是所谓的冒泡，需要设置最里面的MouseArea的propagateComposedEvents属性为true，并且在它的点击事件中将accepted属性设置为false，否则它会把

这个事件吃掉，而不会发生事件穿透。

将上面的代码该一下

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./CustomColor.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  //MouseArea  Item {  id: *it*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Rectangle{  id:*rect*  width: 100  height: 100  color:*it*.randClr()  MouseArea{  anchors.fill: *parent*  onClicked: {  *console*.log(*mouse*.x,*mouse*.y,*mapToGlobal*(*mouse*.x,*mouse*.y))  }  drag.target: *rect*  }  Text {  id: *txt*  anchors.centerIn: *parent*  font.bold: true  font.pointSize: 16  color: *it*.randClr()  text: *qsTr*("拖我走")  MouseArea{  anchors.fill: *parent*  propagateComposedEvents: true //将事件穿透设置为true  onClicked: {  *console*.log(*txt*.text)  *txt*.color = *it*.randClr()  *mouse*.accepted = false //防止事件被吃掉了  }  drag.target: *txt*  }  }  }  }  } | //CustomColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果

|  |
| --- |
|  |

**注意，在需要使用鼠标拖动的控件里面不能使用anchors属性，否则会限制控件的拖动！！！！除非你就是想限制它在某个方向移动**

## 7.按钮控件QML Button：注意需要import QtQuick.Controls 2.15

### 有一个非常坑的地方：如果使用button的icon属性设置按钮的图片并不能显示，这是显示一个黑区域，

**设置图片需要用到**indicator属性

|  |
| --- |
| Button{  id:*btn1*  width: 100  height: 50  checkable: true  autoExclusive: true //保证统一时刻一组按钮只有一个被选中  onDownChanged: {  *console*.log("down:",*down*,"pressed",*pressed*)  }  indicator: Image{  source: "./icon1.png"  }  } |

|  |
| --- |
|  |

### 其实可以不用icon和indicator来设置图片样式等等，因为Button控件有一个background属性可以设置这些

|  |
| --- |
| Button{  id:*btn2*  x:120  width: 100  height: 50  checkable: true  autoExclusive: true  background: Rectangle{  anchors.fill: *btn2*  color:{  if(*btn2*.pressed){  return "gray"  } else{  return "lightgreen"  }  }  border.color: *btn2*.pressed? "red" :"yellow"  }  } |

### Button扩展

#### 1.CheckBox

注意：CheckBox有一个叫做autoExclusive，它是从Button继承而来，对button有用对CheckBox没有用，因为CheckBox是多选按钮

如果非得要将其设置为排他的话可以使用ButtonGroup

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  ButtonGroup{  exclusive: true  buttons: *row2*.children  }  Row{  id:*row2*  CheckBox{  checked: true  //tristate: true //设置按钮为三态按钮  text: *qsTr*("学习")  onCheckStateChanged: {  *console*.log("checkState",*checkState*)  }  }  CheckBox{  text: *qsTr*("吃饭")  }  CheckBox{  //checked: true  text: *qsTr*("打游戏")  }  }  } |  |

**可以利用几个CheckBox和一个ButtonGroup实现父CheckBox的三态的效果，如子项全部选中，父项就是选中，如果有一个没有选中，父项是半选中状态，**

**如果全部都没有选中，父项为非选中状态，另一方面，选中父项就会选中所有子项，取消选中父项就会取消选中所有子项**

**实例：**

|  |  |
| --- | --- |
| **//main.qml**  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Column{  ButtonGroup{  id:*childGroup*  exclusive: false  buttons: row2.children  checkState: *parentBox*.checkState  }  CheckBox{  id:*parentBox*  text: *qsTr*("Parent")  checkState: *childGroup*.checkState  }  CheckBox{  checked: true  text: *qsTr*("Child1")  leftPadding: *indicator*.width  ButtonGroup.group:*childGroup*  }  CheckBox{  text: *qsTr*("Child2")  leftPadding: *indicator*.width  ButtonGroup.group:*childGroup*  }  }  } |  |

|  |
| --- |
|  |

#### 2. DelayButton

用的不多，需要一直按住才会有进度条的效果

|  |
| --- |
| DelayButton{  width: 180  height: 60  delay: 3000  onProgressChanged: {  *console*.log("operation in progress...")  }  } |

#### 3.RadioButton

默认就有排他功能因为这是一个单选按钮，不过需要把它们放在同一个父容器中

|  |  |
| --- | --- |
| ColumnLayout{ //需要import QtQuick.layouts2.15  RadioButton{  text: *qsTr*("c++")  checked: true  }  RadioButton{  text: *qsTr*("java")  }  RadioButton{  text: *qsTr*("python")  }  } |  |

如果想知道用户点击了那个RadioButton，需要RadioButton所在的容器和ButtonGroup配合工作

如：

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ButtonGroup{  id:*btngrp*  buttons: *col*.children  onClicked: {  if(*rd1*.checked){  *console*.log("radio1 checked...")  } else if(*rd2*.checked){  *console*.log("radio2 checked...")  } else if(*rd3*.checked){  *console*.log("radio3 checked...")  }  }  }  Column{  id:*col*  RadioButton{  id:*rd1*  checked:true  text: *qsTr*("c++")  }  RadioButton{  id:*rd2*  checked:false  text: *qsTr*("java")  }  RadioButton{  id:*rd3*  checked:false  text: *qsTr*("python")  }  }  } |  |

#### 4.开关按钮Switch

**默认自动排他没有效果，如果有这个需要，必须使用ButtonGroup来实现**

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ButtonGroup{  id:*btngrp*  buttons: *col*.children  exclusive: true  }  Switch{  text: *qsTr*("wifi")  onPositionChanged: {  *console*.log("position:",*position*)  }  onVisualPositionChanged: {  *console*.log("Visual position:",*visualPosition*)  }  }  Switch{  text: *qsTr*("bluetooth")  }  }  } |  |

#### 5，TabButton，使用的比较多

**TabButton必须和TabBar一起使用。**

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  TabBar{  TabButton{  text: *qsTr*("Home")  width:50;height:50  }  TabButton{  width:50;height:50  text: *qsTr*("Discover")  }  TabButton{  width:50;height:50  text: *qsTr*("Activity")  }  }  } |  |

#### 6.RoundButton

**跟一般的按钮没有什么大的区别，只不过它默认是圆的，可以修改radius属性将其变为圆角矩形**

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  RoundButton{  text: *qsTr*("hello")  onPressed: {  *console*.log("hello pressed...")  }  radius: 10  }  } |  |

#### 7.ToolButton

**需要在ToolBar里面使用，可以修改文本颜色**

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ToolBar{  RowLayout{  anchors.fill: *parent*  ToolButton{  palette.buttonText: "red"  text: *qsTr*("<")  onClicked: {  *console*.log("<<<<<<<<<<<<<<<<<<<")  }  }  Label{  palette.windowText: "green"  text: *qsTr*("title")  elide: Label.ElideRight  horizontalAlignment: *Qt*.AlignHCenter  verticalAlignment: *Qt*.AlignVCenter  Layout.fillWidth: true  }  ToolButton{  palette.buttonText: "purple"  text: *qsTr*("\*")  onClicked: {  *console*.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  }  }  }  }  } |  |

#### 8.Button的contentItem属性，它可以是文本，Rectangle等等

**如果是Text，可以用它来设置按钮的字体大小，颜色，斜体，粗体**

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  Button{  id:*btn*  text: *qsTr*("MyBtn")  width: 100  height: 60  background:Rectangle{  anchors.fill: *parent*  color: "steelblue"  }  contentItem: Text { //contentItem还可以是Rectangle,主要是修改按钮内容的属性  id: *name*  text: *btn*.text  color: "purple"  font.pixelSize:18  font.bold: true  font.italic: true  horizontalAlignment: *Qt*.AlignHCenter  verticalAlignment: *Qt*.AlignVCenter  }  }} |  |

**还可以这样子：将contentItem改为Rectangle，在rectangle里面放置文本和图片**

|  |  |
| --- | --- |
| **//main.qml**  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  Button{  id:*btn*  text: *qsTr*("MyBtn")  width: 100  height: 60  // background:Rectangle{  // anchors.fill: parent  // color: "steelblue"  // }  contentItem: Rectangle { //contentItem还可以是Rectangle,主要是修改按钮内容的属性  anchors.fill: *parent*  color: "lime"  Text {  id: *txt*  text: *btn*.text  color: "purple"  font.pixelSize:18  font.bold: true  font.italic: true  y:*parent*.height/3  }  Image {  id: *img*  source: "./icon1.png"  width: 30  height: 20  anchors.left: *txt*.right  anchors.bottom: *txt*.bottom  }  }  }  } |  |

## 8.Property的使用

### 8.1 可以在一个元素中使用property关键字来设置自定义属性

格式：property 类型 属性名称：值，如：property int myTopMargin:0，类型是默认支持的类型可以是string，real（代替double和float），color等等其他属性也可以

定义好后，就可以像使用内置的属性一样使用

|  |
| --- |
| Rectangle{  x:350  width: 100  height: 50  property string strColor: "purple"  property color kcolor: *strColor*  color:*kcolor*  } |

|  |
| --- |
|  |

注意：这些设置好的属性只能作为属性值，而不是属性名，如kcolor就只能放在冒号的右边，否则会报错

#### 可以使用property设置Component属性来实现动态加载组件的功能，如

property Component myComponent 注意不要赋值，这个需要外部传入，注意需要Loader。

在使用的地方创建一个组件，起一个id号，将这个id号传入需要的地方，见main.qml

|  |  |  |  |
| --- | --- | --- | --- |
|  | //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Component{  id:*com*  Rectangle{  width: 100  height: 50  color: "yellow"  }  }  MyRect{  myComponent: *com*    }  } | //MyRect.qml 首字母要大写，否则报错  import QtQuick 2.0  Rectangle{  property Component myComponent  property var myVar  property list<Rectangle> rectList  width: 300  height: 200  color: "pink"  Loader{  sourceComponent: *myComponent*  }  } |  |

### 8.2property的特殊用法

**注意：这个类型可以是上面的类型，还可以是一个很特殊的类型var，还可以是一个泛型list，如：property list<Rectangle> rectList**

**此外，可以在property关键字前面加readonly修饰符来设置一个只读属性。还可以在property前面添加required修饰符，表示这个属性必须设置，否则程序无法运行**

**如：**required property Component myComponent

property关键字还有一个作用：给一个组件的子组件起一个别名以供外部使用（默认情况下外部不能够直接使用一个控件的子控件）

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Component{  id:*com*  Rectangle{  //这里不要设置,否则下面设置无效  //width: 100  // height: 50  //color: "yellow"  }  }  MyRect{  myComponent: *com*  Component.onCompleted: {  *console*.log("width:",*rectWidth*)  //innerColor = "orange"  *inner*.color = "orange"  *inner*.width = 150  *inner*.height = 100  }  }  } | //MyRect.qml 首字母要大写，否则报错  Rectangle{  id:*borderRect*  width: 300  height: 200  required property Component myComponent  property var myVar  property list<Rectangle> rectList  readonly property int rectWidth: *width*  property alias inner: *innerRect*  //property alias innerColor: innerRect.color //可以只给子组件的一个属性起别名，这样子外部就只能够操作这个属性  color: "pink"  Rectangle{  id:*innerRect*  Loader{  sourceComponent: *myComponent*  }  }  } |

## 9.Text元素

text元素设置宽高没有作用，需要设置的是contentWidth和contentHeight

text元素有一个elide属性也就是省略属性，当设置为true时，可以将过长的文本用省略号代替，实例1（截屏）

|  |
| --- |
|  |

实例2，可以设置Text的样式，有Normal ,Outline,Raised,Sunken四种样式

|  |  |
| --- | --- |
| Text {  id: *txt1*  x:50  y:20  color: "steelblue"  font.pointSize: 24  font.bold: true  font.letterSpacing: 10  font.underline: true  font.family: "华文行楷"  text: *qsTr*("i love girls\ni love girls\ni love girls")  Component.onCompleted: {  *console*.log("contentWidth:",*contentWidth*)  *console*.log("contentHeight:",*contentHeight*)  *console*.log("lineCount:",*lineCount*)  *console*.log("lineHeight:",*lineHeight*)  }  }  Row {  x:10  y:140  Text { font.pointSize: 24; text: "Normal" }  Text { font.pointSize: 24; text: "Raised"; style: Text.Raised; styleColor: "lime" }  Text { font.pointSize: 24; text: "Outline";style: Text.Outline; styleColor: "red" }  Text { font.pointSize: 24; text: "Sunken"; style: Text.Sunken; styleColor: "yellow" }  } |  |

还可以设置Text组件的文本格式，有**Text.AutoText,Text.PlainText,Text.StyledText,Text.RichText,Text.MarkdownTex**t五种，默认是AutoText

**Text.StyledText 支持如下标签**

**<b></b> - bold**

**<del></del> - strike out (removed content)**

**<s></s> - strike out (no longer accurate or no longer relevant content)**

**<strong></strong> - bold**

**<i></i> - italic**

**<br> - new line**

**<p> - paragraph**

**<u> - underlined text**

**<font color="color\_name" size="1-7"></font>**

**<h1> to <h6> - headers**

**<a href=""> - anchor**

**<img src="" align="top,middle,bottom" width="" height=""> - inline images**

**<ol type="">, <ul type=""> and <li> - ordered and unordered lists**

**<pre></pre> - preformatted**

**&gt; &lt; &amp;**

实例3

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  Column{  Text {  font.pointSize: 24  text: "<b>Hello</b> <i>World!</i> Auto"  }  Text {  font.pointSize: 24  textFormat: Text.RichText  text: "<b>Hello</b> <i>World!</i> Rich"  }  Text {  font.pointSize: 24  textFormat: Text.StyledText  text: "<b>Hello</b> <i>World!</i> Styled"  }  Text {  font.pointSize: 24  textFormat: Text.PlainText  text: "<b>Hello</b> <i>World!</i> Plain"  }  Text {  font.pointSize: 24  textFormat: Text.MarkdownText  // text: "<b>Hello</b> <i>World!</i>"  text: "\*\*Hello\*\* \*World!\*"  }  }  } |  |

**还可以设置wrapMode。wrapMode的值可以是Text.NoWrap (default) ,Text.WordWrap ,Text.WrapAnywhere ,Text.Wrap默认值是noWrap，可以设置它自动换行**

最常用的是**Text.WordWrap或者Text.NoWrap**

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")    Rectangle{  id:*rec*  width: 180  height: 100  border.color: pink  color: "green"  Text {  font.pointSize: 16  text: "Hello Hello Hello Hello Hello Hello "  wrapMode: Text.WordWrap //以一个词为分割单位  anchors.fill: *parent*  }  }  } |  |

**linkActivated，可以处理超链接**

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")    Text {  textFormat: Text.RichText  text: "See the <a href=\"http://qt-project.org\">Qt Project website</a>."  onLinkActivated: *console*.log(*link* + " link activated")  onLinkHovered: *console*.log("hovered",*link*)  onHoveredLinkChanged: *console*.log("hover link changed",*hoveredLink*) //的鼠标从超链接处移开时触发  }  } |  |

如果想要实现点击超链接后打开指定的网页，可以使用*Qt*.openUrlExternally(*link*)

如：

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")    Text {  textFormat: Text.RichText  text: "See the <a href=\"http://qt-project.org\">Qt Project website</a>."  onLinkActivated: *Qt*.openUrlExternally(*link*) //打开链接对应的页面  onLinkHovered: *console*.log("hovered",*link*)  onHoveredLinkChanged: *console*.log("hover link changed",*hoveredLink*) //的鼠标从超链接处移开时触发  }  } | 点击链接后，打开了网页 |

如果需要点击链接的时候同时改变链接的颜色，需要使用Text.StyledText格式，修改linkColor属性的颜色，还可以给鼠标在链接悬停的状态设置另一个颜色：

使用Text.RichText需要配合css

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")    Text {  //textFormat: Text.RichText  textFormat: Text.StyledText  text: "See the <a href=\"http://qt-project.org\">Qt Project website</a>."  onLinkActivated:{  *Qt*.openUrlExternally(*link*)  *linkColor*= "purple"  }  onLinkHovered: onLinkHovered: *linkColor*="orange" onHoveredLinkChanged: *console*.log("hover link changed",*hoveredLink*) //的鼠标从超链接处移开时触发  }  } | 点击前    悬停状态：    点击后：打开网页，链接变色 |

注意qml中有一个叫做属性绑定的概念，如label.text: "space pressed :" + *spacePresses* + " times"

也就是直接将属性值和一个数值通过+号连接起来

这种绑定有一个不好的地方，就是如果此时你修改了这个属性值，绑定就失效了。

比较好的方法是使用赋值方式替代绑定方式，就像下列中的写法2

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  //写法1  // Text {  // id: label  // x: 24;y: 24  // property int spacePresses: 0  // text: "space pressed :" + spacePresses + "times" //这是属性绑定  // onTextChanged: console.log("text changed to ",text)  // focus: true  // Keys.onSpacePressed: increment()  // // Keys.onEscapePressed: label.text=''//注意如果此时将文本的值修改，那么将会销毁上面的协议，按空格键将不起作用  // Keys.onEscapePressed: spacePresses =0  // //Keys.onEnterPressed: spacePresses = 100 //没有作用  // function increment(){  // spacePresses++  // }  // }  //写法2，较好  Text {  id: *label*  x: 24;y: 24  property int spacePresses: 0  text:"space pressed :0 times"  onTextChanged: *console*.log("text changed to ",*text*)  focus: true  Keys.onSpacePressed: *increment*()  Keys.onEscapePressed: *label*.text="space pressed :0 times"  function *increment*(){  *spacePresses*++  *label*.text = "space pressed :" + *spacePresses* + " times" ;//使用赋值来修改属性值比直接绑定属性值好，也就是"="代替":"  }  }  } | 按空格键    按esc键 |

## 10、popup元素

popup元素的默认他的visible属性是false，也就是默认不显示。如果需要显示可以有几种方法1.直接将他的visible属性设置为true。2.在window的Component.onCompleted事件中调用

popup.open()方法。3.在按钮的点击事件中调用popup.open()方法或者popup.visible = true.

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Component.onCompleted: {  //popup.open()  *popup*.visible = true  }  Popup{  id:*popup*  width:300  height: 200  //visible: true  }  } |  |

popup的外观很像Rectangle。

也可以调用popup.close()方法来关闭一个popup

#### 注意：1.在qml中如果父控件的visible属性为false，即使子控件的visible属性为true，父子控件都不会显示，但是popup是例外的：

**实例，对比两种情况**

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle{  width: 200  height:100  color: "pink"    Popup{  id:*popup*  width:80  height: 40  visible: true  }  }  } | import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle{  width: 200  height:100  color: "pink"  visible: false  Popup{  id:*popup*  width:80  height: 40  visible: true  }  }  } |
| **效果** | **效果** |

#### 2.一般来说，当两个qml元素互相遮挡的时候，z值大的会覆盖z值小的元素，但是popup又是一个例外，popup总是在最上面，即使你将他的z值设置为负数，也没有用

**实例**

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle{  width: 200  height:100  color: "pink"  //z:0  }  Popup{  id:*popup*  width:200  x:100  height: 100  visible: true  //z:-1  }  **}** | import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle{  width: 200  height:100  color: "pink"  z:10  }  Popup{  id:*popup*  width:200  x:100  height: 100  visible: true  z:-1  } |
|  |  |

#### 实际上，popup的z值只有当两个popup互相遮挡的时候才有意义，z值大的popup覆盖z小的popup。其他情况z不起作用，都是popup覆盖其他元素

#### Popup还有modal，focus和closePolicy等等属性

|  |
| --- |
| Popup{  id:*popup*  width:200  x:100  height: 100  visible: true  z:-1  modal: true  focus: true//设置为可以获取输入焦点  closePolicy: Popup.CloseOnEscape|Popup.CloseOnReleaseOutsideParent  } |

#### 还可以将Popup的closePolicy设置为：NoAutoClose，此时按下esc键或者点击父窗体Popup不会关闭

#### 还有一个dims属性，控制非模态方式显示Popup时父窗体的颜色

#### enter和exit，可以在这两个属性在设置过渡效果，让透明度从0到1和透明度从1到0

例子，利用按钮控制Popup的显示隐藏并且显示过渡效果

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Button{  anchors.centerIn: *parent*  width: 100  height: 50  background:Rectangle {  color: "orange"  Text {  id: *txt*  text: *qsTr*("show/hide")  anchors.centerIn: *parent*  }  }  onClicked: *popup*.visible = !*popup*.visible  }  Popup{  id:*popup*  width:200  x:100  height: 100  visible: true  z:-1  focus: true  closePolicy: Popup.CloseOnEscape|Popup.CloseOnReleaseOutsideParent  enter: Transition {  NumberAnimation { property: "opacity"; from: 0.0; to: 1.0 ;duration: 1000}  }  exit: Transition {  NumberAnimation { property: "opacity"; from: 1.0; to: 0.0; duration: 1000 }  }  }  } | 点击按钮之前：    点击一次，隐藏    再点击一次，又显示，一直点击一直交替。。。。 |

#### Popup也有contentItem，可以在里面设置文本，背景等等

如：

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  Button{  anchors.centerIn: *parent*  width: 100  height: 50  background:Rectangle {  color: "orange"  Text {  id: *txt*  text: *qsTr*("show/hide")  anchors.centerIn: *parent*  }  }  onClicked: *popup*.visible = !*popup*.visible  }  Popup{  id:*popup*  width:200  x:100  height: 100  visible: true  z:-1  focus: true  contentItem: Rectangle{  anchors.fill: *parent*  color: "deeppink"  Text {  id: *txt1*  text: *qsTr*("i am popup")  anchors.centerIn: *parent*  }  }  closePolicy: Popup.CloseOnEscape|Popup.CloseOnReleaseOutsideParent  enter: Transition {  NumberAnimation { property: "opacity"; from: 0.0; to: 1.0;duration: 1000}  }  exit: Transition {  NumberAnimation { property: "opacity"; from: 1.0; to: 0.0 ;duration: 500}  }  }  } |  |

overlay属性参考qt assit

## 11.Repeater

#### 用于绘制n个完全相同的元素，以下是基本使用，model可以是常数，也可以是list，下面是常数，表示一共绘制第三个控件

|  |  |
| --- | --- |
| Row{  id:row  Repeater{  model: 4  Rectangle{  width: 120  height: 60  color: "deeppink"  border.color: "yellow"  }  }  } |  |

#### 注意：Repeater可以绘制多个控件，它不是控件的父控件，上面的代码中，父控件其实是它的容器Row

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Row{  id:*row*  objectName: "Row Container"  spacing: 2  //复习一下Repeater  Repeater{  model:["hello","hi","hey"]  Button{  text: modelData  onClicked: {  *console*.log(*parent*.objectName)  }  }  }  } |  |

#### Repeater 有一个index属性，可以利用这个属性乘于一个常数来调整控件的位置

|  |  |
| --- | --- |
| Repeater{  model: 4  Rectangle{  y:index\*70  width: 120  height: 60  color: "deeppink"  border.color: "yellow"  }  } |  |

#### 如果将model改为一个数组，在数组里面放置不同的控件名称可以一次创建几个不同的控件，数组的长度是创建的个数，数组元素是创建的类型

|  |  |
| --- | --- |
| Repeater{  model: ["Button","Rectangle","MouseArea"] //数量还是3，他会计算数组的长度  Rectangle{  y:index\*70  width: 120  height: 60  color: "deeppink"  border.color: "yellow"  Text {  text: modelData  }  }  } |  |

## 12.ListView

#### ListView是一个使用频率非常高的组件，最简单的用法

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ListView{  width: 100  height: 100  model:3  delegate: Text{  text:index  }  }  } |  |

#### delegate可以是任何类型如Rectangle，spacing属性可以控制列表项的间隔

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ListView{  width: 100  height: 100  model:3 //model可以是数字  delegate: Rectangle{  width: 100  height: 30  spacing: 5 //设置一个列表项的间隔  border.color: "deeppink"  color:"yellow"  Text {  text: "Rect"+index  }  }  }  } |  |

#### model还可以是一个数组

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ListView{  width: 100  height: 100  spacing: 5 //设置一个列表项的间隔  model: ["Button","Rectangle","MouseArea"]  delegate: Rectangle{  x:10  width: 100  height: 30  border.color: "deeppink"  color:"yellow"  Text {  text:modelData  anchors.centerIn: *parent*  }  }  } |  |

#### ListView的delegate还可以是ItemDelegate类型，该类型继承自AbstractButton，可以响应点击，另外还可以使用外部的js文件来随机设置颜色

|  |  |
| --- | --- |
| //effect.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  *console*.log(*r*,*g*,*b*)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  } | //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  anchors.fill: *parent*  ListView{  id:*lv*  anchors.fill: *parent*  model:["fried chicken","roasted chicken","stew chicked"]  delegate: ItemDelegate{  width:*lv*.width  text: modelData  background: Rectangle{  color:*getColor*() //注意这里不能直接用js代码，响应定义一个js函数然后调用  function *getColor*(){  var *clr* = *Effect*.getRndColor()  return *Qt*.rgba(*clr*.red,*clr*.green,*clr*.blue)  }  }  onClicked: {  *console*.log(modelData)  }  }  }  }  } |

效果：

|  |  |
| --- | --- |
|  | 点击对应的项会输出他的文本 |

#### 还可以放置滚动条

|  |
| --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  anchors.fill: *parent*  ListView{  id:*lv*  anchors.fill: *parent*  model:["fried chicken","roasted chicken","stew chicked","chicken soup","stew pork","roasted pork","pork soup","ox tail","stew peas"]  delegate: ItemDelegate{  width:*lv*.width  text: modelData  background: Rectangle{  color:*getColor*()  function *getColor*(){  var *clr* = *Effect*.getRndColor()  return *Qt*.rgba(*clr*.red,*clr*.green,*clr*.blue)  }  }  onClicked: {  *console*.log(modelData)  }  }  ScrollBar.vertical: ScrollBar{}  }  }  } |

效果：

|  |
| --- |
|  |

#### 小扩展：在js文件中封装随机颜色工厂

|  |
| --- |
| function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256) //调用上面的函数  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()//调用上面的函数  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

|  |
| --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  //property var clrFaceoty:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFaceoty:*Effect*.randomColorFactory(*Qt*.rgba)//注意：js代码返回的函数只能够赋值给var类型的属性变量  anchors.fill: *parent*  model:["fried chicken","roasted chicken","stew chicked","chicken soup","stew pork","roasted pork","pork soup","ox tail","stew peas"]  delegate: ItemDelegate{  width:*lv*.width  text: modelData  background: Rectangle{  color: *lv*.clrFaceoty() //在有需要的地方以调用函数的方式来调用一个属性变量  }  onClicked: {  *console*.log(modelData)  }  }  ScrollBar.vertical: ScrollBar{}  }  }  } |

#### 注意上面的代码是有点问题的，不能动态添加项，

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  //property var clrFaceoty:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFaceoty:*Effect*.randomColorFactory(*Qt*.rgba)//注意：js代码返回的函数只能够赋值给var类型的属性变量  anchors.fill: *parent*  model:["fried chicken","roasted chicken","stew chicked","chicken soup","stew pork","roasted pork","pork soup","ox tail","stew peas"]  delegate: ItemDelegate{  width:*lv*.width  text: modelData  background: Rectangle{  color: *lv*.clrFaceoty() //在有需要的地方以调用函数的方式来调用一个属性变量  }  onClicked: {  *console*.log(modelData)  }  }  Component.onCompleted: {  *Lv.model*.push("stew beef") //直接使用不会添加  *lv.model*.forEach(function(item){  *console*.log(*item*)  })  }  ScrollBar.vertical: ScrollBar{}  }  }  } |  |

#### 解决办法：先用一个变量接收ListView的model，用这个model变量来添加新项，然后再将model变量赋给ListView的model

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  // property var clrFactory:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFactory:*Effect*.randomColorFactory(*Qt*.rgba) //接收js返回的函数可以在有需要的地方调用  anchors.fill: *parent*  model:["fried chicken","roasted chicken","stew chicked","chicken soup","stew pork","roasted pork","pork soup","ox tail","stew peas"]  delegate: ItemDelegate{  width:*lv*.width  text: modelData  background: Rectangle{  color: *lv*.clrFactory()  //color: Effect.randomColorFactory(Qt.rgba)() //也行，不过写法比较怪  }  onClicked: {  *console*.log(modelData)  }  }  Component.onCompleted: {  var *lvmodel* = *lv*.model //直接使用lv.model是不行的，要先用一个变量接收ListView的model  *lvmodel*.push("stew beef")  *lvmodel*.forEach(function(item){  *console*.log(*item*)  })  *lv*.model = *lvmodel* //再把model设置给ListView的model  }  ScrollBar.vertical: ScrollBar{}  }  }  } |  |

#### 如果ListView的model是一个js的对象数组，delegate的text属性直接使用是没有任何文本显示的

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  // property var clrFactory:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFactory:*Effect*.randomColorFactory(*Qt*.rgba) //接收js返回的函数可以在有需要的地方调用  anchors.fill: *parent*  model:[  {  name:"fried chicked",  price: 3  },  {  name:"steam fish",  price: 6  },  {  name:"ox tail",  price: 8  }  ]  delegate: ItemDelegate{  width:*lv*.width  text: modelData  background: Rectangle{  color: *lv*.clrFactory()    }  onClicked: {  *console*.log(modelData)  }  }  ScrollBar.vertical: ScrollBar{}  }  }  } |  |

#### 解决办法：需要指明文本怎么显示

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  // property var clrFactory:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFactory:*Effect*.randomColorFactory(*Qt*.rgba) //接收js返回的函数可以在有需要的地方调用  anchors.fill: *parent*  model:[  {  name:"fried chicked",  price: 3  },  {  name:"steam fish",  price: 6  },  {  name:"ox tail",  price: 8  }  ]  delegate: ItemDelegate{  width:*lv*.width  text: modelData.name +": $" +modelData.price  background: Rectangle{  color: *lv*.clrFactory()  }  onClicked: {  *console*.log(modelData)  }  }  ScrollBar.vertical: ScrollBar{}  }  }  } |  |

#### 也可以实现点击显示列表项内容的功能

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  // property var clrFactory:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFactory:*Effect*.randomColorFactory(*Qt*.rgba) //接收js返回的函数可以在有需要的地方调用  anchors.fill: *parent*  model:[  {  name:"fried chicked",  price: 3  },  {  name:"steam fish",  price: 6  },  {  name:"ox tail",  price: 8  }  ]  delegate: ItemDelegate{  width:*lv*.width  text: modelData.name +": $" +modelData.price  background: Rectangle{  color: *lv*.clrFactory()  }  onClicked: {  *console*.log(modelData.name + ":$" +modelData.price)  // console.log(JSON.stringify(modelData)) //也是可以的，但是做显示用的话不是很好  }  }  Component.onCompleted: {  var *lvmodel* = *lv*.model  *lvmodel*.push({name:"stew peas",price:6})  *lv*.model = *lvmodel*  }  ScrollBar.vertical: ScrollBar{}  }  }  } |  |

#### 小技巧：如何在ListView选中的item后面添加一个“√”

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  // property var clrFactory:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFactory:*Effect*.randomColorFactory(*Qt*.rgba) //接收js返回的函数可以在有需要的地方调用  anchors.fill: *parent*  model:[  {  name:"fried chicked",  price: 3  },  {  name:"steam fish",  price: 6  },  {  name:"ox tail",  price: 8  }  ]  delegate: ItemDelegate{  width:*lv*.width  //在选中的项显示"√"  text: modelData.name +": $" +modelData.price +(*lv*.currentIndex === index? "√" : "")  background: Rectangle{  color: *lv*.clrFactory()  }  onClicked: {  *console*.log(modelData.name + ":$" +modelData.price)  *lv*.currentIndex = index //需要在点击是时候改变索引值  // console.log(JSON.stringify(modelData)) //也是可以的，但是不是很好  }  }  Component.onCompleted: {  var *lvmodel* = *lv*.model  *lvmodel*.push({name:"stew peas",price:6})  *lv*.model = *lvmodel*  }  ScrollBar.vertical: ScrollBar{}  }  } } | 选择另外一项 |

#### 可以将上面的model改为ListModel，不过需要作适当的修改，js代码相同所以就没有摘抄

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  // property var clrFactory:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFactory:*Effect*.randomColorFactory(*Qt*.rgba) //接收js返回的函数可以在有需要的地方调用  anchors.fill: *parent*  model:ListModel{  ListElement{  name:"fried chicked"  price: 3  }  ListElement{  name:"steam fish"  price: 6  }  ListElement{  name:"ox tail"  price: 8  }  }  //ItemDelegate继承自AbstractButton,能够响应鼠标点击，如果使用Rectangle的话，需要在里面放置一个MouseArea来响应鼠标点击  delegate: ItemDelegate{  width:*lv*.width  //在选中的项显示"√"  //注意：使用ListModel的时候，是可以直接获取ListElement的属性的  text: name +": $" +price +(*lv*.currentIndex === index? "√" : "")  background: Rectangle{  color: *lv*.clrFactory()  }  onClicked: {  *console*.log(index + "."+name + ":$" +price)  //var item = lv.model.get(index) //ListModel有一个get方法，可以获取index对应的项  //console.log(JSON.stringify(item))  *lv*.currentIndex = index //需要在点击是时候改变索引值  // console.log(JSON.stringify(modelData)) //也是可以的，但是不是很好  }  }  Component.onCompleted: {  var *lvmodel* = *lv*.model  *lvmodel*.append({name:"stew peas",price:6})  *lv*.model = *lvmodel*  }  ScrollBar.vertical: ScrollBar{}  }  }  } |  |

#### 当然也可以将ListModel放到例外一个qml文件，在ListView的model属性中加载该文件

|  |  |  |
| --- | --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ListView{  width: 200  height: 100  model: ContactModel{}//一个qml文件就是一个组件  delegate: Text { //绘制数据  text: name+ " " +number + " "+ email  }  }  } | //ContactModel.qml //数据文件  import QtQuick 2.15  import QtQuick.Controls 2.15  ListModel {  ListElement {  name: "Bill Smith"  number: "555 3264"  email:"Bill123@gmail.com"  }  ListElement {  name: "John Brown"  number: "555 8426"  email:"Johnb@yahoo.com"  }  ListElement {  name: "Sam Wise"  number: "555 0473"  email:"Sam123@outlook.com"  }  } |  |

#### 可以设置ListView的高亮highlight属性，可以通过键盘的上下方向键来移动列表项，被选中的会高亮显示，注意必须就列表项的颜色设置为透明

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ListView{  width: 100  height: 100  spacing: 5 //设置一个列表项的间隔  highlight: Rectangle{color:"orange";radius: 8} //必须要将元素的颜色设置为透明否则效果不好  focus: true  model: ["Button","Rectangle","MouseArea"]  delegate: Rectangle{  x:10  width: 100  height: 30  //border.color: "deeppink"  color:"transparent"  Text {  text:modelData  anchors.centerIn: *parent*  }  }  }  } |  |

#### ListView有一个currentIndex属性它代表当前选中的对象，可以index属性一起使用配合MouseArea实现点击列表项高亮显示的效果

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ListView{  id:*lv*  width: 100  height: 100  spacing: 5 //设置一个列表项的间隔  highlight: Rectangle{x:10;color:"orange";radius: 8} //必须要将元素的颜色设置为透明否则效果不好  focus: true  model: ["Button","Rectangle","MouseArea"]  delegate: Rectangle{  x:10  width: 100  height: 30  //border.color: "deeppink"  color:"transparent"  Text {  text:modelData  anchors.centerIn: *parent*  }  MouseArea{ //实现点击改变高亮的效果  anchors.fill: *parent*  onClicked: {*lv*.currentIndex = index}  }  }  }  } |  |

#### 还可以为ListView添加header和footer

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  Rectangle{  //anchors.centerIn: parent  border.color:"yellow"  ListView{  id:*lv*  width: 100  height: 100  spacing: 5 //设置一个列表项的间隔  header:Rectangle{  x:10;  color:"lightsteelblue";  width:*parent*.width;  height: 10  }  footer:Rectangle{  x:10;  color:"lightsteelblue";  width:*parent*.width;  height: 10  }  highlight: Rectangle{x:10;color:"orange";radius: 8} //必须要将元素的颜色设置为透明否则效果不好  focus: true  model: ["Button","Rectangle","MouseArea"]  delegate: Rectangle{  x:10  width: 100  height: 30  //border.color: "deeppink"  color:"transparent"  border.color: "yellow"  Text {  text:modelData  anchors.centerIn: *parent*  }  MouseArea{ //实现点击改变高亮的效果  anchors.fill: *parent*  onClicked: {*lv*.currentIndex = index}  }  }  }  }  } |  |

#### 利用ListView的section属性属性列表项分类显示，比较复杂,必须按照分类排列子项，否则不会放置在一起

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ListModel{  id:*animalsModel*  ListElement{  name:"Parrot"  size:"Small"  }  ListElement{  name:"Guinea pig"  size:"Small"  }  ListElement{  name:"Dog"  size:"Medium"  }  ListElement{  name:"Cat"  size:"Medium"  }  ListElement{  name:"Cow"  size:"Large"  }  ListElement{  name:"Elephant"  size:"Large"  }  }  // The delegate for each section header  Component {  id: *sectionHeading*  Rectangle {  width: 300  height: *childrenRect*.height  color: "lightsteelblue"  required property string section  Text {  text: *parent*.section  font.bold: true  font.pixelSize: 20  }  }  }  ListView {  id: *view*  anchors.top: *parent*.top  width: 300  height: 300  model: *animalsModel*  delegate: Text {  required property string name  text: *name*  font.pixelSize: 18  }  section.property: "size"  section.criteria: ViewSection.FullString  section.delegate: *sectionHeading*  }  } |  |

#### 分组实例2.

|  |  |
| --- | --- |
| //effect.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } | //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./effect.js" as *Effect*  Window {  width: 320  height: 200  visible: true  title: *qsTr*("Hello World")  Item{  id: *itemRoot*  anchors.fill: *parent*  // property var clrFactory:Effect.randomColorFactory(Qt.rgba)  ListView{  id:*lv*  property var clrFactory:*Effect*.randomColorFactory(*Qt*.rgba) //接收js返回的函数可以在有需要的地方调用  anchors.fill: *parent*  model:ListModel{  ListElement{  name:"fried chicked"  price: 3  category:"Cheap Food"  }  ListElement{  name:"stew chicked"  price: 3  category:"Cheap Food"  }  ListElement{  name:"steam fish"  price: 6  category:"Better Food"  }  ListElement{  name:"fried fish"  price: 6  category:"Better Food"  }  ListElement{  name:"ox tail"  price: 8  category:"Hi-Grade Food"  }  ListElement{  name:"pig tail"  price: 8  category:"Hi-Grade Food"  }  }  //ItemDelegate继承自AbstractButton,能够响应鼠标点击，如果使用Rectangle的话，需要在里面放置一个MouseArea来响应鼠标点击  delegate: ItemDelegate{  width:*lv*.width  required property string name  required property int index  required property int price  //在选中的项显示"√"  font.bold: ListView.isCurrentItem //选中的就一粗体显示  font.pixelSize: 25  text: *name* +": $" +*price* +(*lv*.currentIndex === *index*? "√" : "")//注意：使用ListModel的时候，是可以直接获取ListElement的属性的  //text: name  background: Rectangle{  color: *lv*.currentIndex === index? *lv*.clrFactory(): "lightgrey"  }  onClicked: {  *console*.log(*index*+"."+*name* + ":$" +*price*)  //var item = lv.model.get(index) //ListModel有一个get方法，可以获取index对应的项  //console.log(JSON.stringify(item))  *lv*.currentIndex = *index* //需要在点击是时候改变索引值  // console.log(JSON.stringify(modelData)) //也是可以的，但是不是很好  }  }  Component.onCompleted: {  var *lvmodel* = *lv*.model  *lvmodel*.append({name:"stew peas",price:6,category:"Better Food"}) //append方法是ListModel的方法，不是js数组的方法  *lv*.model = *lvmodel*  }  ScrollBar.vertical: ScrollBar{}  section.property: "category"  section.criteria: ViewSection.FullString  section.delegate: Rectangle{  width:*lv*.width  height:*childrenRect*.height  color:"steelblue"  required property string section  Text {  id: *txt*  text: *parent*.section  font.pixelSize: 30  }  }  }  }  } |

## 13 ComboBox

#### 最简单的ComboBox实例如下，model是一个数字

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ComboBox{  model:3  }  } |  |

#### 简单实例2、

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  ComboBox{  id:*combo*  model:["cpp","java","javaScript","python"]  onActivated: { //在这个事件中获取选中项的索引，文本等  *console*.log("index",*index*)  //获取文本  *console*.log("currentValue:"+*textAt*(*index*))  }  }  } | 选择任意一项，会输出它的索引和文本 |

#### model也可以是一个数组

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ComboBox{  model: model:["Button","Rectangl","MouseArea"]  }  } |  |

#### 使用ListModel的ComboBox

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  ComboBox{  id:*combo*  // model:["cpp","java","javaScript","python"]  //注意：当将ComboBox的model属性设置为一个ListModel而且ListElement中有多个角色的时候，必须支持textRole,否则报错  textRole:"text"  //textRole:"value"  model: ListModel{  ListElement{  text:"cpp"  value:100  }  ListElement{  text:"java"  value:90  }  ListElement{  text:"javaScript"  value:95  }  ListElement{  text:"python"  value:100  }  }  onActivated: { //在这个事件中获取选中项的索引，文本等  //如果model是一个数组，可以这样子获取，是一个ListModel也行  // console.log("index",index)  // //获取文本  // console.log("currentValue:"+textAt(index))  //如果model是一个ListModel还可以这样子来获取,这个方法比较好  let *selected* = *model*.get(*index*)  *console*.log(*selected*.text + ":"+*selected*.value)//可以获取ListElement的所有属性  }  }  } |  |

#### 可以编辑的ComboBox

需要将editable属性设置为true，而且需要在onAccepted事件中添加代码

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  //可以编辑的ComboBox  ComboBox{  id:*cbx*  editable: true  model:ListModel{  id:*listMod*  ListElement{text:"apple"}  ListElement{text:"banana"}  ListElement{text:"pineapple"}  }  onAccepted: {//这个的作用是将编辑框中的内容添加到ComboBox的列表项，  if(*find*(*editText*)===-1) //先在列表项在查找文本，没有才添加不要重复添加  {  *listMod*.append({text:*editText*})  }  }  }  } | 编辑前    编辑并且按回车确认后： |

#### currentIndex可以修改当前选中项，currentValue是当前列表项的值是一个只读属性

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  //可以编辑的ComboBox  ComboBox{  id:*cbx*  editable: true  currentIndex:1  model:ListModel{  id:*listMod*  ListElement{text:"apple"}  ListElement{text:"banana"}  ListElement{text:"pineapple"}  }  Component.onCompleted: *console*.log(*currentValue*)//currentValue是一个只读属性  onAccepted: {//这个的作用是将编辑框中的内容添加到ComboBox的列表项，  if(*find*(*editText*)===-1) //先在列表项在查找文本，没有才添加不要重复添加  {  *listMod*.append({text:*editText*})  }  }  }  } |  |

思考：currentValue和currentText有什么区别？

当ComboBox的列表项只有一个值并且是string的情况下两者是一样的。如果有两个值，

currentValue是指当前项的valueRole指定的子项的值，currentText是指当前项的textRole指定的子项的值

实例1

|  |  |
| --- | --- |
|  | 结果： |

实例2

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ComboBox {  textRole: "text"  valueRole: "value"  Component.onCompleted: {  *console*.log("currentText:",*currentText*)  *console*.log("currentValue:",*currentValue*)  }  model: [  { value: 100, text: *qsTr*("Enter") },  { value: 200, text: *qsTr*("Shift") },  { value: 300, text: *qsTr*("Control") }  ]  }  } |  |

实例3

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ComboBox {  textRole: "text"  valueRole: "title"  onCurrentTextChanged: {  *console*.log("currentText:",*currentText*)  }  onCurrentValueChanged: {  *console*.log("currentValue:",*currentValue*)  }  model: [  { value: 100, text: *qsTr*("Enter"),title:"first" },  { value: 200, text: *qsTr*("Shift") ,title:"second"},  { value: 300, text: *qsTr*("Control"),title:"third" }  ]  } |  |

#### ComboBox有一个displayText属性，可以修改列表项显示的内容：

实例4

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ComboBox {  textRole: "text"  valueRole: "title"  displayText: *currentText* +":" +*currentValue*  onCurrentTextChanged: {  *console*.log("currentText:",*currentText*)  }  onCurrentValueChanged: {  *console*.log("currentValue:",*currentValue*)  }  model: [  { value: 100, text: *qsTr*("Enter"),title:"first" },  { value: 200, text: *qsTr*("Shift") ,title:"second"},  { value: 300, text: *qsTr*("Control"),title:"third" }  ]  }  } |  |

#### ComboBox的验证器：可以限制输入，只允许输入符合条件的数值

如：整数验证器，只允许输入0-9的数字

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ComboBox {  model: 10  editable: true  validator: IntValidator { //整数验证器，只允许输入指定范围的数字  top: 9  bottom: 0  }  }  }  //注意，当你将top修改为20，你会发现即使你输入了99，也能接受，这是为什么？  //这是因为ComboBox有一个叫做acceptableInput的属性引起的。怎么解决？ |  |

正则表达式验证器

|  |
| --- |
|  |

#### 自定义绘制ComboBox

|  |  |  |
| --- | --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ComboBox {  id: *control*  model: ["First", "Second", "Third"]  delegate: ItemDelegate {//控制model中每一项的绘制  width: *control*.width  contentItem: Text {  text: modelData  // color: "#21be2b"  color: "orange" //修改列表项的颜色，可以使用%取模的方法实现一个ComboBox的列表项有不同颜色的效果  font: *control*.font  elide: Text.ElideRight  verticalAlignment: Text.AlignVCenter  }  highlighted: *control*.highlightedIndex === index  }  indicator: Canvas { //控制ComboBox右边的小箭头的绘制，可以不要  id: *canvas*  x: *control*.width - *width* - *control*.rightPadding  y: *control*.topPadding + (*control*.availableHeight - *height*) / 2  width: 12  height: 8  contextType: "2d"  Connections {  target: *control*  function onPressedChanged() { *canvas*.requestPaint(); }  }  onPaint: {  *context*.reset();  *context*.moveTo(0, 0);  *context*.lineTo(*width*, 0);  *context*.lineTo(*width* / 2, *height*);  *context*.closePath();  *context*.fillStyle = *control*.pressed ? "#17a81a" : "#21be2b";  *context*.fill();  }  }  contentItem: Text {  leftPadding: 0  rightPadding: *control*.indicator.width + *control*.spacing  text: *control*.displayText  font: *control*.font  color: *control*.pressed ? "#17a81a" : "#21be2b"  verticalAlignment: Text.AlignVCenter  elide: Text.ElideRight  }  background: Rectangle {//只会改变ComboBox的编辑框所在区域的背景  implicitWidth: 120  implicitHeight: 40  border.color: *control*.pressed ? "#17a81a" : "#21be2b"  border.width: *control*.visualFocus ? 2 : 1  radius: 2  }  popup: Popup {//修改ComboBox列表取的显示属性  y: *control*.height - 1 //控制ComboBox的下拉控件的显示位置当设置的太大，ComboBox会倒过来显示  width: *control*.width  implicitHeight: *contentItem*.implicitHeight  padding: 1  contentItem: ListView {  clip: true  implicitHeight: *contentHeight*  model: *control*.popup.visible ? *control*.delegateModel : null  currentIndex: *control*.highlightedIndex  ScrollIndicator.vertical: ScrollIndicator { } //滚动条，可以改为ScrollBar  }  background: Rectangle {  border.color: "#21be2b"  radius: 2  }  }  }  } | |  | | --- | |  | |

|  |
| --- |
|  |

|  |
| --- |
|  |

#### 实现阴影效果

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  ComboBox {  id: *control*  model: ["First", "Second", "Third"]  delegate: ItemDelegate {//控制model中每一项的绘制  width: *control*.width  contentItem: Text {  text: modelData  // color: "#21be2b"  color: "orange" //修改列表项的颜色，可以使用%取模的方法实现一个ComboBox的列表项有不同颜色的效果  font: *control*.font  elide: Text.ElideRight  verticalAlignment: Text.AlignVCenter  layer.enabled: true  layer.effect: DropShadow{//增加阴影效果  horizontalOffset: 3  verticalOffset: 3  radius: 8.0  //radius: 0.0  samples: 17  //color: "#80000000"  color: "lightsteelblue"  }  }  highlighted: *control*.highlightedIndex === index  }  indicator: Canvas { //控制ComboBox右边的小箭头的绘制，可以不要  id: *canvas*  x: *control*.width - *width* - *control*.rightPadding  y: *control*.topPadding + (*control*.availableHeight - *height*) / 2  width: 12  height: 8  contextType: "2d"  Connections {  target: *control*  function onPressedChanged() { *canvas*.requestPaint(); }  }  onPaint: {  *context*.reset();  *context*.moveTo(0, 0);  *context*.lineTo(*width*, 0);  *context*.lineTo(*width* / 2, *height*);  *context*.closePath();  *context*.fillStyle = *control*.pressed ? "#17a81a" : "#21be2b";  *context*.fill();  }  }  contentItem: Text {  leftPadding: 0  rightPadding: *control*.indicator.width + *control*.spacing  text: *control*.displayText  font: *control*.font  color: *control*.pressed ? "#17a81a" : "#21be2b"  verticalAlignment: Text.AlignVCenter  elide: Text.ElideRight  }  background: Rectangle {//只会改变ComboBox的编辑框所在区域的背景  implicitWidth: 120  implicitHeight: 40  border.color: *control*.pressed ? "#17a81a" : "#21be2b"  border.width: *control*.visualFocus ? 2 : 1  radius: 2  }  popup: Popup {//修改ComboBox列表取的显示属性  y: *control*.height - 1 //控制ComboBox的下拉控件的显示位置当设置的太大，ComboBox会倒过来显示  width: *control*.width  implicitHeight: *contentItem*.implicitHeight  padding: 1  contentItem: ListView {  clip: true  implicitHeight: *contentHeight*  model: *control*.popup.visible ? *control*.delegateModel : null  currentIndex: *control*.highlightedIndex  ScrollIndicator.vertical: ScrollIndicator { } //滚动条，可以改为ScrollBar  }  background: Rectangle {  border.color: "#21be2b"  radius: 2  layer.enabled: true  layer.effect: DropShadow{//增加阴影效果  horizontalOffset: 3  verticalOffset: 3  radius: 8.0  //radius: 0.0  samples: 17  //color: "#80000000"  color: "lightgray"  }  }  }  }  } |  |

## 14 FocusScope

#### FocusScope简单使用

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  import QtGraphicalEffects 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  FocusScope{ //没有border属性,没有background属性  id:*fss*  TextInput{  id:*input*  focus: true  text: "focus " + *focus*+" "+"activeFocus " +*activeFocus*  }  }  FocusScope{ //没有border属性,没有background属性  id:*fss2*  y:100  TextInput{  id:*input2*  focus: true  text: "focus " + *focus*+" "+"activeFocus " +*activeFocus*  }  }  } |  |

#### 解决有多个控件同时有输入焦点或者焦点丢失的方法

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import QtQuick.Layouts 1.15  import QtGraphicalEffects 1.15  //Component 和Loader  Window {  width: 640  height: 480  visible: true  title: *qsTr*("My QML Demo")  FocusScope{ //没有border属性,没有background属性  id:*fss*  Button{  id:*btn1*  width: 100  height:50  focus: true  background: Rectangle{  anchors.fill: *parent*  color: *btn1*.activeFocus? "green":"gray"  }  }  }  FocusScope{ //没有border属性,没有background属性  id:*fss2*  y:100  Button{  id:*btn2*  width: 100  height:50  focus: false  focusPolicy:*Qt*.NoFocus  background: Rectangle{  anchors.fill: *parent*  color: *btn2*.activeFocus? "green":"gray"  }  onClicked: {  *btn2*.forceActiveFocus()//解决多焦点  }  }  }  } |  |

## 15、StackView

#### StackView是用来切换窗口页面的

实例1，初始化

|  |  |
| --- | --- |
| //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } | //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  StackView{  id:*stv*  anchors.fill: *parent*  initialItem:Component{  Rectangle{  color: *it*.randClr()  }  }  }  }  } |

效果：

|  |
| --- |
|  |

#### StackView切换页面，使用push方法如果需要从新页面切换回去，需要使用pop方法

实例，js代码相同。

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Component{  id:*page2*  Rectangle{  /\*color:"deeppink"\*/  Text{  text: "page2"  }  MouseArea{  anchors.fill: *parent*  onClicked: { //实现点击鼠标切换StackView的页面  //stv.push(page2)  *stv*.pop() //返回之前的页面  }  }  }  }  StackView{  id:*stv*  anchors.fill: *parent*  initialItem:Component{  Rectangle{  color: *it*.randClr()  Text{  text: "page1"  }  MouseArea{  anchors.fill: *parent*  onClicked: { //实现点击鼠标切换StackView的页面  //stv.push(page2)  *stv*.push(*page2*,{color:"purple"})  }  }  }  }  }  }  } | 初始状态    点击页面后    再点击有切换回来    一直点击会一直切换。。。 |

#### StackView加载自定义组件

例，StackView加载我们上面的日期标签自定义控件，

##### 方法一，使用Component

需要注意的是，label的面积很小，不容易捕捉，可以用一个矩形把label包起来，放置MouseArea，在MouseArea的点击事件中实现页面的回退，由于其他文件没有变化，只贴出main.qml文件

|  |
| --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Component{  id:*page2*  Rectangle{  /\*color:"deeppink"\*/  Text{  text:"page2"  }  MouseArea{  anchors.fill: *parent*  onClicked: { //实现点击鼠标切换StackView的页面  //stv.push(page2)  *stv*.pop() //返回之前的页面  }  }  }  }  Component{  id:*labelPg*  Rectangle{  id:*frame*  color:"lightsteelblue"  DateTimeLabel{  rndClr: *it*.randClr  }  MouseArea{  anchors.fill: *parent*  onClicked: { //实现点击鼠标切换StackView的页面  //stv.push(page2)  *stv*.pop() //返回之前的页面  }  }  }  }  StackView{  id:*stv*  anchors.fill: *parent*  initialItem:Component{  Rectangle{  color: *it*.randClr()  Text{  text: "page1"  }  MouseArea{  anchors.fill: *parent*  onClicked: { //实现点击鼠标切换StackView的页面  //stv.push(page2)  //stv.push(page2,{color:"purple"})  *stv*.push(*labelPg*)  }  }  }  }  }  }  } |

##### 方法二，使用url加载自定义组件，注意在组件文件里面一定不能使用required关键字，否则没有效果，这种方法不好，因为修改了组件文件，可能没有通用性

而且需要为这个组件添加MouseArea，在MouseArea的点击事件槽函数中调用StackView的pop方法来返回，在同一个目录下面的使用qml文件可以互访子组件

|  |  |
| --- | --- |
| //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } | import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  StackView{  id:*stv*  anchors.fill: *parent*  initialItem:Component{  Rectangle{  color: *it*.randClr()  Text{  text: "page1"  }  MouseArea{  anchors.fill: *parent*  onClicked: { //实现点击鼠标切换StackView的页面  //stv.push(page2)  //stv.push(page2,{color:"purple"})  //stv.push(labelPg)  *stv*.push("./DateTimeLabel.qml",{rndClr:*it*.randClr})  }  }  }  }  }  }  } |
| //DataTimeLabel.qml  import QtQuick 2.0  import QtQuick.Controls 2.15  Label{  width: 100  height:40  id:*lblDate*  font.bold: true  font.pixelSize: 30  //required property color rndClr  //required property var rndClr  property var rndClr //如果想用StackView来通过URL加载该组件，一定不能够使用required关键字  color:*rndClr*()  Timer{  repeat: true  triggeredOnStart: true  running: true  interval: 1000  onTriggered: {  *lblDate*.text = *Qt*.formatDateTime(new *Date*(),'yyyy-MM-dd hh:mm:ss')  }  }  MouseArea{  anchors.fill: *parent*  onClicked: *stv*.pop()  }  } | 点击一下， |

## 16、StackLayout

是一个可以有多个Item的布局，不过同一时刻只有一个Item可见，可以在MouseArea的点击事件中通过修改currentIndex的值来改变显示的Item

#### 简单使用，实例1

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  StackLayout{  id:*stl*  anchors.fill: *parent*  currentIndex: 1  Rectangle{  id:*pg1*  color:'teal'  implicitWidth: 200  implicitHeight: 200  Text{  text: "page 0"  }  MouseArea{  anchors.fill: *parent*  onClicked: *stl*.currentIndex=1  }  }  Rectangle{  id:*pg2*  Text{  text: "page 1"  }  color:'plum'  implicitWidth: 300  implicitHeight: 200  MouseArea{  anchors.fill: *parent*  onClicked: *stl*.currentIndex=0  }  }  }  } | //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果：

|  |  |
| --- | --- |
| 点击前 | 点击后    再点击有回到1，。。。一直点击一直交替切换 |

#### 实例2.可以通过按钮来切换

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Row{  id:*row*  spacing: 2  Button{  id:*btn1*  text: "page4"  onClicked: *stl*.currentIndex = 3  }  Button{  id:*btn2*  text: "page3"  onClicked: *stl*.currentIndex = 2  }  Button{  id:*btn3*  text: "page2"  onClicked: *stl*.currentIndex = 1  }  Button{  id:*btn4*  text: "page1"  onClicked: *stl*.currentIndex = 0  }  }  StackLayout{  id:*stl*  anchors.top: *row*.bottom  anchors.bottom: *parent*.bottom  anchors.right: *parent*.right  anchors.left: *parent*.left  currentIndex: 1  Rectangle{  id:*pg1*  color:*it*.randClr()  implicitWidth: 200  implicitHeight: 200  Text{  text: "page 0"  }  MouseArea{  anchors.fill: *parent*  onClicked: *stl*.currentIndex=1  }  }  Rectangle{  id:*pg2*  Text{  text: "page 1"  }  color:*it*.randClr()  implicitWidth: 300  implicitHeight: 200  MouseArea{  anchors.fill: *parent*  onClicked: *stl*.currentIndex=0  }  }  Rectangle{  id:*pg3*  color:*it*.randClr()  implicitWidth: 200  implicitHeight: 200  Text{  text: "page 2"  }  MouseArea{  anchors.fill: *parent*  onClicked: *stl*.currentIndex=1  }  }  Rectangle{  id:*pg4*  Text{  text: "page 3"  }  color:*it*.randClr()  implicitWidth: 300  implicitHeight: 200  MouseArea{  anchors.fill: *parent*  onClicked: *stl*.currentIndex=0  }  }  }  } | // MyColors.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果：

|  |  |
| --- | --- |
|  |  |

**注意：StackLayout和StackView是不同的实现，StackView只有当有需要的时候才动态添加页面，StackLayout是创建的时候就有一定数量的控件，只不过只显示一个，可以通过它的**

**currentIndex属性来切换需要显示的控件**

#### \* 此外，可以对上面的代码进行简单封装，使他更容易扩展

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Row{  id:*row*  spacing: 2  Component{  id:*btnCom*  Button{  required property string btnText  required property var layout  required property int viewIndex  text: *btnText*  onClicked: *layout*.currentIndex = *viewIndex*  }  }  Component.onCompleted: {  let *menus* = ["home","help","new","login"]  *menus*.forEach(function(item,index){  *btnCom*.createObject(*row*,{layout:*stl*,btnText:*item*,viewIndex:*index*})  })  }  }  StackLayout{  id:*stl*  anchors.top: *row*.bottom  anchors.bottom: *parent*.bottom  anchors.right: *parent*.right  anchors.left: *parent*.left  currentIndex: 1  Component{  id:*pgCom*  Rectangle{  id:*pg1*  required property var pageClr  required property int pageWidth  required property int pageHeight  required property string pgText  color:*pageClr*()  implicitWidth: *pageWidth*  implicitHeight: *pageHeight*  Text{  text: pgText  }  }  }  Component.onCompleted: {  for(var *i*=0;*i*<4;++*i*){  *pgCom*.createObject(*stl*,{pageClr:*it*.randClr,pageWidth:300,pageHeight:300,pgText:"page"+*i*})  }  }  }  } | //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果是一样的：

|  |  |
| --- | --- |
|  |  |

## 17、SwipeView

#### 简单使用

实例

|  |
| --- |
| 项目结构 |

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("ListView扩展")  SwipeView {  id: *view*  currentIndex: 1  anchors.fill: *parent*  Item {  id: *firstPage*  // Rectangle{  // anchors.fill: parent  // color: "lime"  // }  Image{  source: "img/congq.jpg"  anchors.fill: *parent*  }  }  Item {  id: *secondPage*  // Rectangle{  // anchors.fill: parent  // color: "blue"  // }  Image{  source: "img/jcc.jpg"  anchors.fill: *parent*  }  }  Item {  id: *thirdPage*  // Rectangle{  // anchors.fill: parent  // color: "orange"  // }  Image{  source: "img/ctwheatfield.jpg"  anchors.fill: *parent*  }  }  }  PageIndicator {  id: *indicator*  count: *view*.count  currentIndex: *view*.currentIndex  anchors.bottom: *view*.bottom  anchors.horizontalCenter: *parent*.horizontalCenter  }  } | 默认显示第二页因为设置了index=1也就是第二个  向右移动显示第一个页面    以此类推。。。 |

#### 注意：SwipeView有一个不好的地方就是如果直接给它设置宽高，它就会失效！！也就是它只支持anchors.xx的设置方法

#### 扩展、为SwipeView动态添加子控件

所需图片

|  |  |
| --- | --- |
| // images/beach.jpg | // images/splitlegs.jpg |
| // images/trees.jpg |  |

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Component{  id:*imgCom*  Image{  id:*img*  required property string imgPath  source:*imgPath*  }  }  SwipeView{  id:*swv*  Component.onCompleted: {  let *paths* =["images/beach.jpg","images/splitlegs.jpg","images/trees.jpg"]  *paths*.forEach(function(item){  *imgCom*.createObject(*swv*,{imgPath:*item*})  })  *swv*.currentIndex=1  }  }  PageIndicator{  count:*swv*.count  currentIndex: *swv*.currentIndex  anchors.horizontalCenter: *parent*.horizontalCenter  anchors.bottom: *parent*.bottom  anchors.bottomMargin: 10  }  }  } | //MyColors.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果

|  |  |
| --- | --- |
| 启动状态 | 滑动 |

#### TabBar与SwipeView联动

图片同上

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./MyColors.js" as *ColorGen*  import QtQuick.Layouts 1.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  TabBar{  id:*header*  anchors.left: *parent*.left  anchors.right: *parent*.right  currentIndex: *swv*.currentIndex  Repeater{  model:["Home","Help","About"]  TabButton{  text: modelData  onClicked: *swv*.currentIndex = index  }  }  }  Component{  id:*imgCom*  Image{  id:*img*  required property string imgPath  source:*imgPath*  }  }  SwipeView{  id:*swv*  anchors.top: *header*.bottom  anchors.bottom: *parent*.bottom  Component.onCompleted: {  Let *paths*=["images/beach.jpg","images/splitlegs.jpg","images/trees.jpg"]  *paths*.forEach(function(item){  *imgCom*.createObject(*swv*,{imgPath:*item*})  })  *swv*.currentIndex=1  }  }  PageIndicator{  count:*swv*.count  currentIndex: *swv*.currentIndex  anchors.horizontalCenter: *parent*.horizontalCenter  anchors.bottom: *parent*.bottom  anchors.bottomMargin: 10  }  }  } | //MyColors.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

效果

|  |  |  |
| --- | --- | --- |
|  |  |  |

# 18、Dialog

Qml中Dialog有几种，其中版本1.3的是通用对话框，2.15的是Popup的子类

#### 1.3版本Dialog，

注意需要给1.3版本的起一个别名，否则会和2.15的冲突

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  import QtQuick.Dialogs 1.3 as *Dlg*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Dialog Demo")  Item{  id:*it*  anchors.fill: *parent*  Dlg.Dialog{ //1.3版本的  id:*dlg2*  title: "hi"  Text {  id: *txt*  text: *qsTr*("hi Dialog 1.3!!!")  }  standardButtons: Dialog.Ok  onAccepted: *console*.log(*txt*.text)  }  Button{  text: "hello"  onClicked: *dlg2*.visible = true  }  }  } | 初始状态    点击按钮，弹出对话框    点击OK，生成对话框的文本 |

#### 2.15版本的Dialog

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Dialog Demo")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Dialog{ //2.15版本的Dialog继承自Popup  id:*dlg*  title:"say hello"  x:*parent*.width/2-120  y:*parent*.height/2-70  Text {  id: *txt*  text: *qsTr*("hello from qml dialog!!!")  }  modal:true  standardButtons: Dialog.Ok | Dialog.Cancel  onAccepted: *console*.log(*txt*.text)  }  Button{  text: "hello"  onClicked: *dlg*.visible = true  //onClicked: dlg2.visible = true  }  }  } | 点击按钮    点击OK会输出对话框的文本 |

#### ColorDialog，FileDialog，FontDialog都是1.3版本的，

**如果需要使用必须import QtQuick.Dialogs 1.3 as *Dlg***

*然后使用*Dlg.ColorDialog{} 这样子来调用

|  |  |
| --- | --- |
| import "./MyColors.js" as *ColorGen*  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  import QtQuick.Dialogs 1.3 as *Dlg*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Dialog Demo")  Item{  id:*it*  anchors.fill: *parent*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Dlg.ColorDialog{  id:*clrDlg*  title: "choose color"  onAccepted: *console*.log("color chose: "+*clrDlg*.color)  }  Dlg.FileDialog{  }  Dlg.FontDialog{  }  Row{    Button{  text: "color"  onClicked: *clrDlg*.visible = true  //onClicked: dlg2.visible = true  }  }  }  } | 点击按钮    可以拖动滑块然后点击选择运算点击OK按钮获取颜色 |

#### FontDialog

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Layouts 1.15  import QtQuick.Controls 2.15  import QtQuick.Dialogs 1.3 as *Dlg*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Dialog Demo")  Item{  id:*it*  anchors.fill: *parent*  Dlg.FontDialog{  id:*fontDlg*  title:"choose font"  onAccepted: *console*.log("Font chosen: ",*fontDlg*.font )  }  Button{  text: "font"  //onClicked: dlg.visible = true  onClicked: *fontDlg*.visible = true  }  }  } | 点击按钮    点击OK可以获取选中的字体 |

# 19、Behavior

#### 1.简单使用

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle {  id: *rect*  width: 100; height: 100  color: "red"  Behavior on width {  NumberAnimation { duration: 1000 }  }  MouseArea {  anchors.fill: *parent*  onClicked: *rect*.width = 50  }  }  } | 效果：点击一下正方形，他的长度就会变为50这个过程持续1秒钟 |

#### 扩展，可以利用js代码实现更好的效果

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle {  id: *rect*  width: 100; height: 100  color: "red"  Behavior on width {  NumberAnimation { duration: 1000 }  }  MouseArea {  anchors.fill: *parent*  onClicked: {  if(*rect*.width===100)  *rect*.width = 50  else  *rect*.width=100  }  }  }  } | 效果  初始化状态    第一次点击，将矩形的宽度变为50    再点击一次，宽度又会变为100 |

#### 还可以做一个透明度的动画

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle {  id: *rect*  width: 100; height: 100  color: "deeppink"  Behavior on opacity {  NumberAnimation { duration: 1000 }  }  MouseArea {  anchors.fill: *parent*  onClicked: {  if(*rect*.opacity===1)  *rect*.opacity = 0.3  else  *rect*.opacity=1  }  }  }  } | 初始状态    用鼠标点击一下矩形会变为半透明    再点击一次有回来 |

#### 扩展2、ColorAnimation

|  |  |
| --- | --- |
| //main.qml  import "./MyColors.js" as *ColorGen*  import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  // Text {  // id: txt  // anchors.horizontalCenter: parent.horizontalCenter  // property int counter  // font.pointSize: 20  // text: "Blinking:" +counter  // FadeBehavior on text{}  // Timer{  // running: true  // repeat: true  // interval: 1000  // onTriggered: ++txt.counter  // }  // }  // Rectangle {  // id: rect  // width: 100; height: 100  // color: "deeppink"  //// Behavior on width {  //// NumberAnimation { duration: 1000 }  //// }  // Behavior on opacity {  // NumberAnimation { duration: 1000 }  // }  // MouseArea {  // anchors.fill: parent  // onClicked: {  // if(rect.opacity===1)  // rect.opacity = 0.3  // else  // rect.opacity=1  // }  // }  // }  Rectangle {  id: *rect*  property var rndClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  width: 50  height: 50  color: "deeppink"  Behavior on x{  NumberAnimation{duration: 1000}  }  Behavior on color{  ColorAnimation {  duration: 1000  }  }  MouseArea {  anchors.fill: *parent*  onClicked: {  *rect*.color=*rect*.rndClr()  if(*rect*.x===0)  *rect*.x=300  else  *rect*.x=0  }  }  }  } | //MyColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } |

扩展3. OpacityAnimator，也是可以有动画效果，不过只对透明度起作用

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Rectangle {  id: *opacityBox*  width: 50  height: 50  color: "deeppink"  OpacityAnimator {  id:*animt*  target: *opacityBox*;  from: 0;  to: 1;  duration: 1000  running: true  }  Timer{  interval: 1000  repeat: true  running: true  onTriggered: *animt*.running = !*animt*.running  }  }  } | 效果利用Timer不断触发OpacityAnimator      矩形的透明度会值改变 |

#### 2.可以将一个Behavior单独保存到一个qml文件，

然后在有需要的地方加载这个文件

|  |  |
| --- | --- |
| //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Text {  id: *txt*  anchors.horizontalCenter: *parent*.horizontalCenter  property int counter  font.pointSize: 20  text: "Blinking:" +*counter*  FadeBehavior on text{}  Timer{  running: true  repeat: true  interval: 1000  onTriggered: ++*txt*.counter  }  }  } | // **FadeBehavior.qml**  import QtQuick 2.15  Behavior{  id:*fBeh*  property Item fadeTarget:*targetProperty*.object  SequentialAnimation{  NumberAnimation {  target: *fBeh*.fadeTarget  property: "opacity"  duration: 200  to:0  easing.type: Easing.OutQuad  }  NumberAnimation {  target: *fBeh*.fadeTarget  property: "opacity"  duration: 200  to:1  easing.type: Easing.InQuad  }  }  } |

效果：文字闪烁效果

# 20 qml自定义组件

### 将4.Component和Loader的最后一个例子的label组件独立写到一个qml文件中，就是一个自定义组件

|  |  |  |
| --- | --- | --- |
| //CustomColor.js  function *getColor*(){  let *colors*=["blue","deeppink","steelblue","lime","orange","yellow"];  var *index* = *Math*.round(*Math*.random()\*10)%6; //Math.random()产生0-1之间的随机数，需要\*10变为1-10之间  //console.log(index)  return *colors*[*index*];  }  function *getRangedRnd*(max){  return *Math*.floor(*Math*.random() \* *max*)  }  function *getRndColor*(){  let *r* = *getRangedRnd*(256)  let *g* = *getRangedRnd*(256)  let *b* = *getRangedRnd*(256)  //console.log(r,g,b)  return {red: *r*/255,green: *g*/255,blue: *b*/255} //注意这个对象是所有参数的值必须在0-1之间，否则没有效果  }  //封装随机颜色生成器  function *randomColorFactory*(rgbFunction){  return function(){//返回一个函数  let *clr* =*getRndColor*()  return *rgbFunction*(*clr*.red,*clr*.green,*clr*.blue)//函数又有返回值  }  } | //DateTimeLable.qml  import QtQuick 2.0  import QtQuick.Controls 2.15  Label{  width: 100  height:100  id:*lblDate*  font.bold: true  font.pixelSize: 30    Timer{  repeat: true  triggeredOnStart: true  running: true  interval: 1000  onTriggered: {  *lblDate*.text = *Qt*.formatDateTime(new *Date*(),'yyyy-MM-dd hh:mm:ss')  }  }  } | //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import QtQuick.Controls 2.15  import "./CustomColor.js" as *ColorGen*  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  Item {  id: *it*  property var randClr:*ColorGen*.randomColorFactory(*Qt*.rgba)  Component{  id:*com*  DateTimeLabel{  color:*it*.randClr()  }  }  Column{  id:*col*  width: 200  height: 60  Component.onCompleted: {  // com.createObject(this)  // com.createObject(this)  *batchCreate*(5)  }  function *batchCreate*(n){  for(var *i*=0;*i*<*n*;++*i*){  *com*.createObject(this)  }  }  }  }  } |

效果：

|  |
| --- |
|  |