## 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}  }  }  } |  |

## 2.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*  }  } |

## 3.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"  }  }  }  } |  |

### 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

|  |  |
| --- | --- |
| 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** |

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

### 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**

## 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**

## 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？

|  |
| --- |
|  |

## 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*  }  }  } |  |

|  |
| --- |
|  |

## 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.QMl CheckBox