

使用MATLAB App Designer开发项目——简单教程和示例

**——简单教程和示例**



2021.12

# 目录

[目录 1](#_Toc90565261)

[前言 2](#_Toc90565262)

[1 MATLAB App Designer简介 3](#_Toc90565263)

[1.1 一些必须熟悉的名词 3](#_Toc90565264)

[1.2 应用程序是如何运行的？ 5](#_Toc90565265)

[1.3 常用组件 6](#_Toc90565266)

[1.4 一个简单的例子：3分钟学习教程 10](#_Toc90565267)

[2 App Designer使用入门 15](#_Toc90565268)

[2.1 明确开发需求，并转化为具体对象 15](#_Toc90565269)

[2.2 为App定义属性、实现方法 15](#_Toc90565270)

[2.3 遇到不会实现的功能时，如何解决？ 17](#_Toc90565271)

[2.4 测试与维护：使App尽善尽美 17](#_Toc90565272)

[2.5 编写使用说明，打包、发布 18](#_Toc90565273)

[3 开发项目：基于Hall传感器阵列和NI采集卡的位移测量App 20](#_Toc90565274)

[3.1 初步设计：模块划分和App创建 20](#_Toc90565275)

[3.2 为App定义方法和属性 23](#_Toc90565276)

[3.3 关键功能实现：NI采集卡数据的获取和显示 27](#_Toc90565277)

[3.4 关键功能实现：位移测量结果的解算和显示 35](#_Toc90565278)

[3.5 编写使用说明并发布 37](#_Toc90565279)

[附录 38](#_Toc90565280)

[A MATLAB帮助文件 38](#_Toc90565281)

# 前言

MATLAB App Designer是基于MATLAB平台开发软件的工具，相比常规的软件开发更加便捷、容易上手，可以让开发者更加关注如何实现功能算法本身。本文档希望通过对App Designer的介绍和开发演示，使读者了解使用App Designer开发项目的简单流程。以下是作者在实践过程中的经验总结，希望能对初学者提供一些帮助，提高工具的学习效率。

无论是采用何种工具框架开发软件，其目的都是要满足用户的使用需求。通常，我们可以围绕具体目标，逐步细化项目，然后依次完成每个部分的实现。这样的好处是方便软件各功能的调试和调整，能够让自己在开发过程中保持一个清晰的逻辑框架。在正式上手使用App Designer开发前，建议先以草图的形式明确软件各个模块的设计要素，包括输入、输出、执行的具体功能，以减少开发过程中发现缺漏而不得不推翻重来的情况。当一项功能不知道该如何实现时，可以参照一些成熟的例程和帮助文档，学习借鉴其设计方案，MATLAB自己提供的一些例程已经涵盖了大部分的常用功能。

总而言之，App Designer的学习成本并不高，如果读者了解面向对象编程的基本知识，便能够很快明白其运作模式，并在短时间内完成软件的开发；即便读者没有编程基础，只要掌握了其各个模块的工作流程，结合一些例程也能够很快完成自己想要实现的功能。本文对App Designer中的模块和关键方法使用了中括号【】来强调，以便于读者在回顾时快速定位到相应的内容。希望这篇文档能够切实地帮到各位读者！

若读者对文档有任何编写和内容上的疑问和建议，都可以通过如下方式联系作者：

Email: hgj15@outlook.com

Github:

# MATLAB App Designer简介

## 一些必须熟悉的名词

——本节致力于解决疑问：“这是什么？”

MATLAB App Designer是在MATLAB的Guide的基础上发展而来的，其功能同样是图形界面应用程序的开发，但是在模块编辑、程序编写上更加完善，界面更加美观。本节将对一些基本名词和概念做具体的介绍，若读者已经有所了解，可以根据需要跳过这些内容。本文将以MATLAB R2019a为参考，但其中的大部分内容同样适用于其他版本的App Designer。

在介绍App Designer的工作界面之前，可以先对MATLAB中的一些App进行了解。这些App实际上都可以通过App Designer开发实现，因此读者在使用过程中，也不妨尝试思考其具体实现方式，这或多或少会对自己的项目设计提供一些参照。

App Designer的工作界面是独立于MATLAB的一个单独窗口。在这个窗口中，可以进行软件的界面设计、代码实现、调试、发布等功能。需要注意的是，如果在App Designer中调试软件，在运行过程中的变量会存储在MATLAB的工作区中。因此，在调试过程中，有时也需要在MATLAB主窗口切换以观察或分析程序运行过程中的一些变量是否存在异常。

**基本界面：**

在MATLAB中通过【APP】栏下的【设计App】，或者在命令行输入App Designer，可以打开App Designer设计工具。App Designer的引导界面提供了打开或新建工程文件的选项，以及一些使用的示例和教程。引导界面的组成非常容易理解，在此不再赘述。

打开或者新建一个项目后，App Designer将切换到工作界面，如【fig#】所示。

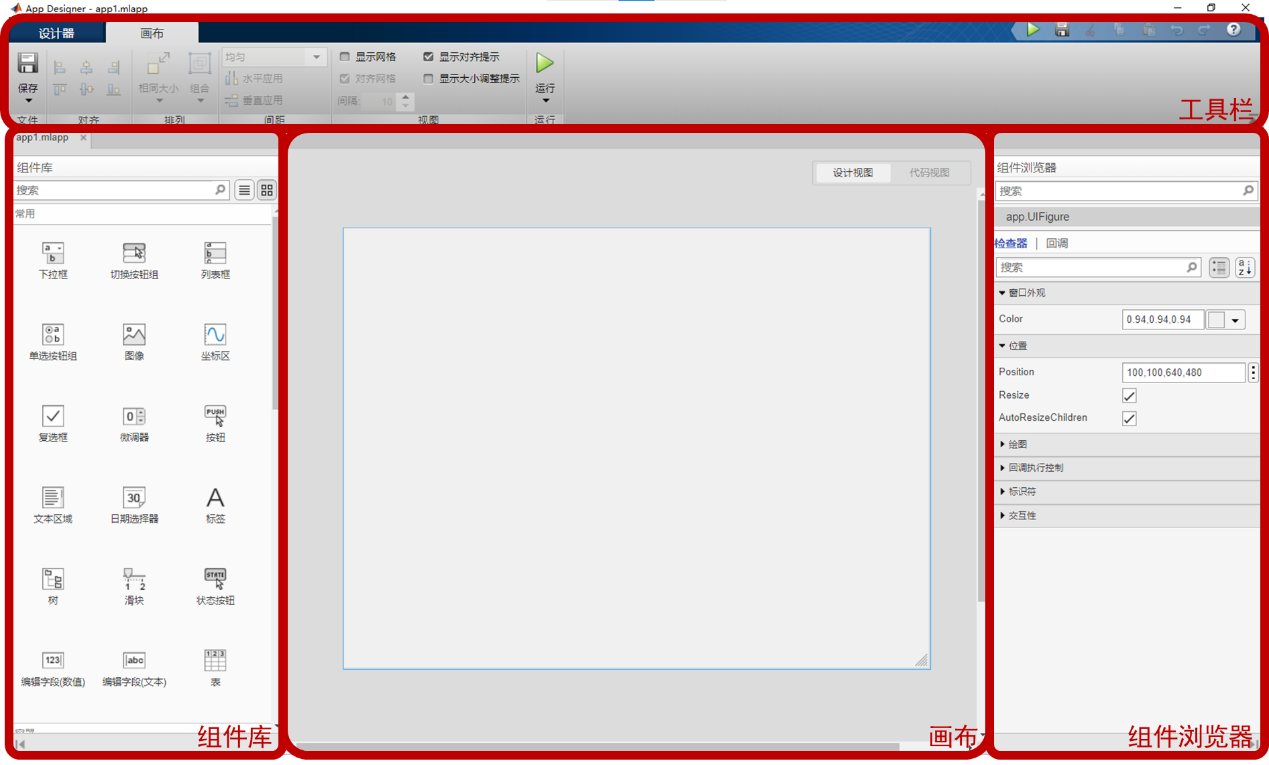


Fig. ‑ App Designer主界面：设计视图

工作界面可以视为多个模块组合成的应用编辑器，其中每个项目文件均以标签页的形式被打开。以下将对各个模块进行简要的介绍，以使读者有一个初步的概念：

**【工具栏】模块：**

包括【设计工具】和【画布】两个子栏目。前者用于实现如保存、发布等文件级的操作，后者用于组件布局和外观调整。该模块涉及的都是一些常见的编辑功能，类似于PowerPoint和MATLAB主界面，读者很容易理解。

**【画布】模块：**

最主要的显示模块，包含【设计视图】和【代码视图】两个部分，后者如【fig#】所示。顾名思义，前者展示了应用的外观，后者则是应用的实现代码。用户可以随时在二者间进行切换。

**【组件库】模块：**

设计视图下，展示了可以使用的组件。选择组件拖放到画布中，App Designer就会生成和该组件有关的代码段。

**【组件浏览器】模块：**

包含两个模块，上方子模块显示了组件的层次结构，可以用来检查组件之间的从属关系是否合理；下方子模块显示了组件的详细属性，如尺寸、位置、颜色等。许多组件属性可以在此处结合设计视图调整，免去了繁琐的代码编辑的工作，*非常实用*。

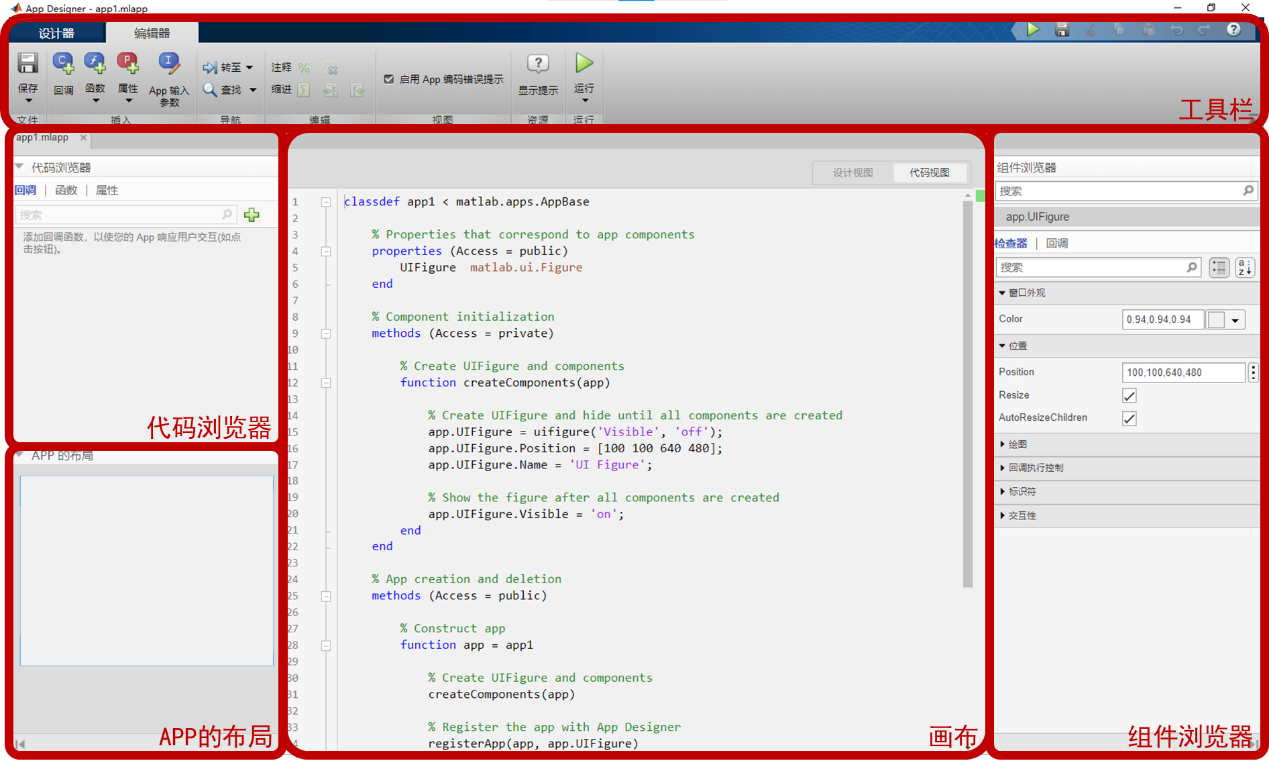


Fig. ‑ App Designer主界面：代码视图

**【代码浏览器】模块：**

代码视图下，显示了应用包含的所有回调、函数、属性（变量）。在回调/函数/属性很多时，可以在此模块快速查找，*非常实用*。保持良好的命名规范有助于提高查找效率。

**【APP的布局】模块：**

代码视图下，设计视图的缩略图。可以在缩略图上对组件进行选择操作。

除了以上一些与操作界面相关的名词外，还有一些在开发过程中经常被提及的名词，也需要读者对其具备较为明确的概念：

**类：**

类是面向对象编程中的一个概念。类是一个用于创建对象的可扩展的程序代码模板，包含两类成员：属性（成员变量）和行为（成员函数）。在MATLAB中，数据类型double和int等都属于类这一概念，App Designer中的程序主体app和描述了交互过程的事件event则是根据特定的类创建的对象。反过来，也可以将类理解为对一系列具备共同点的对象所建立的抽象模型。

**回调函数（Callback; call-after function）：**

回调函数简称回调，具体而言是作为参数传递给其他代码的可执行代码（的引用）。其他函数调用并执行回调函数完毕后，会返回继续运行自身。应用程序接收用户操作指令并做出响应的过程中，将频繁涉及回调函数的调用。通俗地说，回调函数解决了App在接收到用户操作后“该怎么做”的问题，是图形界面应用程序中不可或缺的组成部分。

## 应用程序是如何运行的？

——本节致力于解决疑问：“这是怎么实现的？”

上文已经对App Designer的操作界面做了基本介绍，在软件开发过程中，还应该明确其*动态部分*，即调试或运行过程中，App Designer是如何使我们编写好的软件工作的。

App Designer实现的应用是一种图形界面应用程序，和MATLAB一般的脚本、函数文件的区别在于多出了通过图形界面与用户交互进行这一功能。这类应用通常会遵循如下的运行模式：

因此，这类应用在开始运行后，通常不会主动结束并退出，而是进入等待状态，一旦检测到用户进行操作，就会做出响应。其中，“检测到用户进行操作”这一行为并不完全由应用程序实现，其完整的过程为：鼠标、键盘等硬件检测到操作信号后，和计算机操作系统交互，再由操作系统发出消息至应用程序的消息处理函数。消息处理函数的底层实现由编辑器完成，也不需要开发者实现。因此，最终呈现在开发者面前的实际上是一些函数接口，通常是一些比较容易理解的参数。开发者只需要根据函数接口的具体说明，将所需的参数提取出来（调用），即完成了交互的接收部分。同理，交互的响应输出和上述过程相反，开发者将数据按函数接口的要求正确输出即可，图像的显示、声音的播放等等交由后续的驱动和硬件完成。

上面的介绍，旨在让读者了解应用程序的大致运行流程，和交互过程中，用户实现的应用程序在全流程中负责实现哪些功能。接下来，我们在具体到App Designer中，对细节上的实现做一些说明，这些内容在调试中可能会经常遇到，因此对其有一定的了解能够帮助开发者快速定位到问题的所在。

在App Designer中，应用程序的运行离不开两个重要的对象：app和event。在断点调试中，我们经常会在工作区看到这两个对象，而我们所需要的交互信息通常都是其成员。如果我们对这两个对象进行稍微深入一些的探索，可以看到它们的父类（superclass）最后分别归属到两个不同的基类（base class）：AppBase和event.EventData。通过查看帮助文档可知，AppBase类包含了应用需要的方法（method），它*描述了应用能做什么*，它的子类在其基础上拓展，能做更多更具体的事情；而event.EventData类则是传递给侦听程序的所有数据对象的基类，它*记录了事件的发生*，它的子类在其基础上拓展，能对发生的事件做出更详细的描述。简而言之，对象app包含了应用自身固有的“静态”信息，对象event则包含了交互触发的事件的“动态”信息。当我们调用参数时，根据参数的来源，可以直接在app或event中对其进行访问。这实际上就是App Designer中的交互。

粗略查看AppBase的方法，其中包含了runStartupFcn, notify, delete等等成员，实际上就是上文介绍的图形界面应用程序的运行模式的具体实现：初始化、通知侦听程序事件正在发生、销毁对象等等。当然，我们的开发通常不需要直接涉及这些内容的改动。尽管如此，由App Designer开发的App是AppBase的子类，因此我们在开发过程中创建的许多回调和函数都继承自这些方法。相信读者对上述内容有了初步的了解后，对应用程序的组成和运行中一些主要成员所扮演的角色，已经有了大概的认识。这些内容虽然看似与实际编程工作相关性不大，但是在应用程序的开发和调试中实际上发挥着指导性的作用，希望读者能够多加注意。

## 常用组件

——本节致力于解决疑问：“这有些什么？”

上文已经对App Designer开发的应用程序的运行流程做了一个较为概括的说明，本节将进一步对应用程序中的主要元素——组件库组件展开介绍。本节涉及常用组件的介绍，以记忆性内容为主，建议读者通过实践熟悉掌握。

组件库中的组件随着MATLAB版本的更新，也在不断增删调整。本节将对常规开发中最常用的组件归类介绍，并结合使用经验给出一些使用的建议。通过常用组件的组合使用，已经可以开发出许多MATLAB的工具箱App。本文未介绍的组件，以及组件更加详细的使用方法，可以参阅帮助手册进一步了解。

**UI图窗（UIFigure）：**

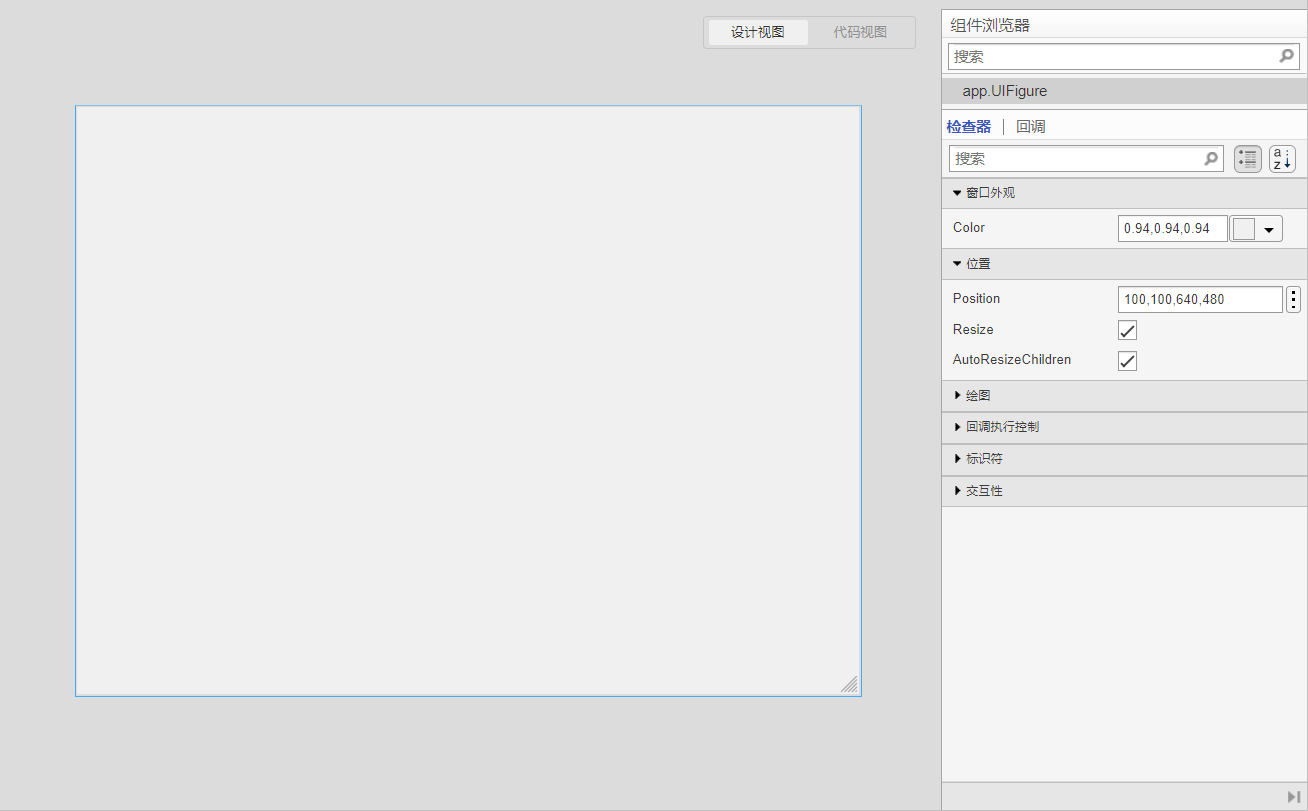


Fig. ‑ UI图窗（UIFigure）

即应用程序的主窗口。新建的空白应用程序仅有UI图窗这一个组件，如果直接运行，将显示一个空白的窗口。UI图窗类似于使用MATLAB绘图时产生的图窗，但是前者是由uifigure函数创建的，后者则是由figure函数创建的。

对单个App，仅有一个可以编辑属性的UI图窗。在App中新增的组件都是UI图窗的子组件。如果希望App有多个图窗，例如，点击主窗口的某个按钮，弹出一个新的对话框窗口，则需要创建多个UI图窗。为了在App Designer中实现这一点，需要创建多个App，然后将这些App关联起来，并设置其主次关系。具体的实现方法可以参阅帮助文档【Creating Multiwindow Apps in App Designer】，本文不再进一步介绍。UI图窗是App中最基础的成员。

UI图窗包含了App主窗口的基本属性，如主窗口的大小、在屏幕上的创建位置、主窗口的名称等。

**面板（Panel）**

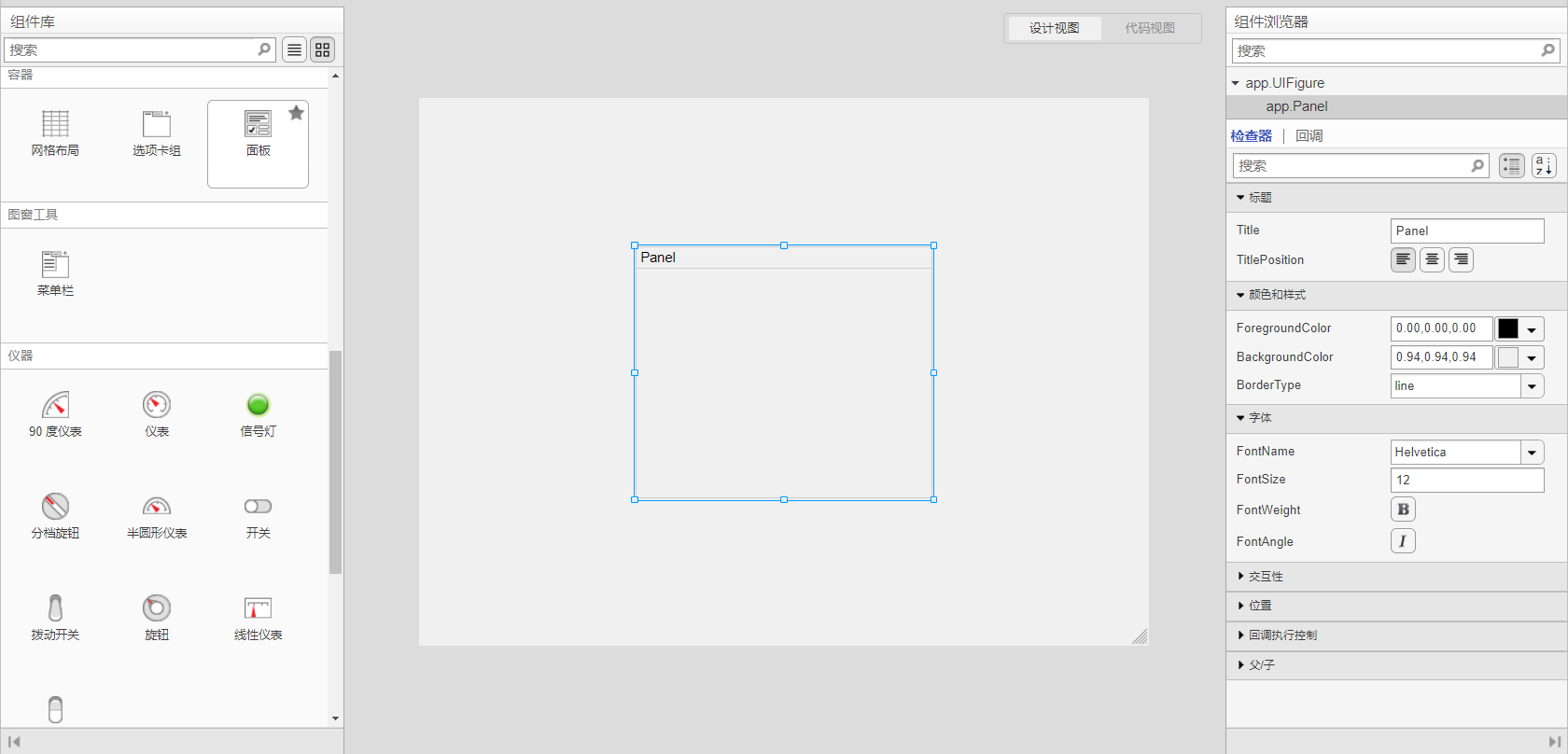
****

Fig. ‑ 面板（Panel）

容器栏下的面板是最为常用的组件之一。面板自身通常并不用于实现信息交互等功能，但是利用面板，可以对App中的各个组件分组，使App的层次结构更加清晰。此外，移动面板不会影响在同一面板中的各个组件的相对位置，每个面板都可以各自独立调整内部的组件布局，这对外观设计而言也是非常便利的。

建议读者在开发App时，先确定App需具备的各项功能的层次结构，将各个组件归类后，灵活使用面板来进行布局调整，以提高设计效率和程序易用性。

**标签/图像（Label/Image）**

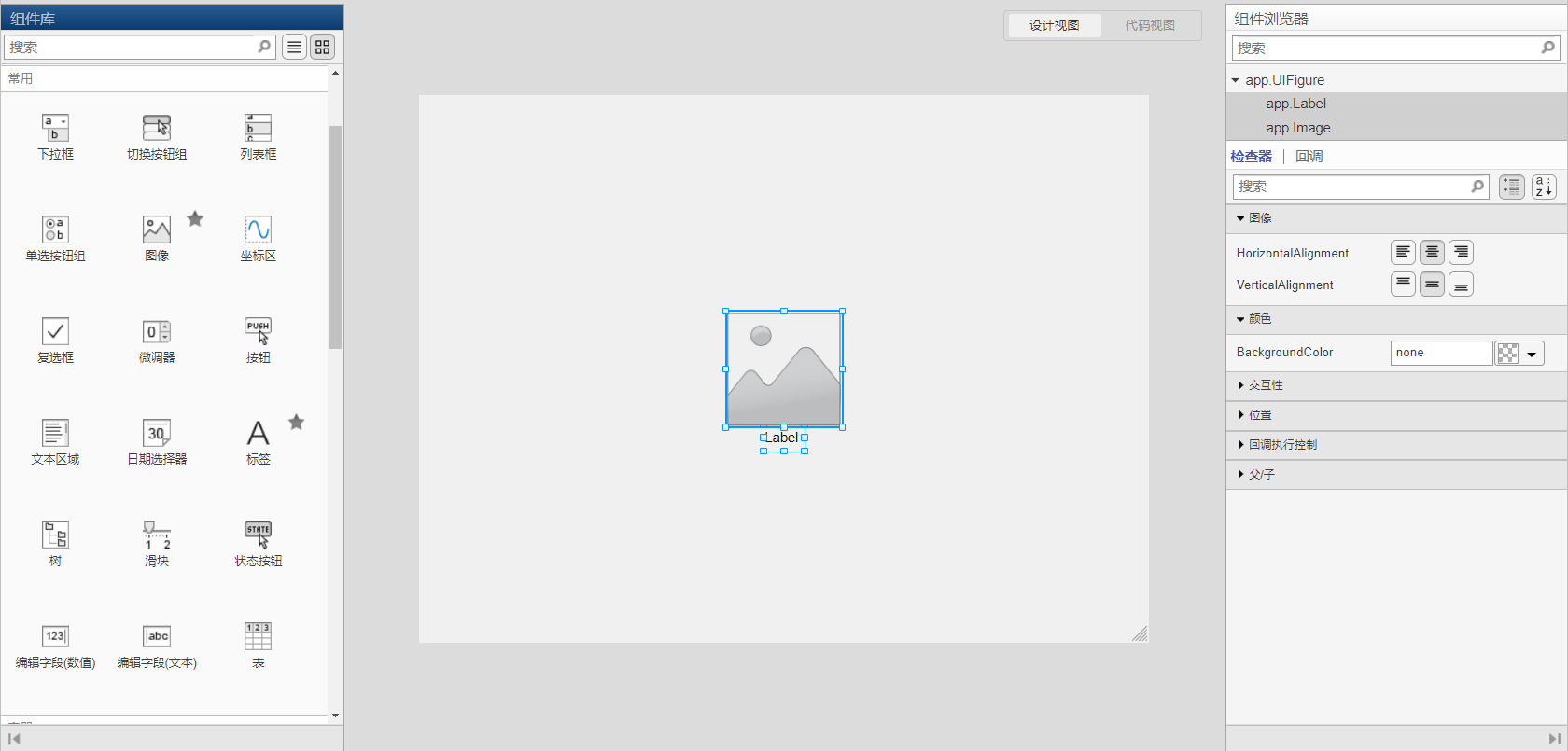


Fig. ‑ 标签/图像（Label/Image）

这类组件通常用于不需要频繁更新的静态信息的展示，例如：输入框输入内容的提示，指示图标，logo等等。组件库中的一些组件实际上已经包括了标签，如列表框、微调器等。适当地设置标签和图像为App的功能添加辅助性的说明能提高App的易用性。当然，这类组件也可以用于交互，例如，可以通过在回调函数中修改标签的Text属性和图像的ImageSource来改变组件的显示内容实现和用户的交互。

**按钮（Button）&类似组件：切换/单选按钮组（ButtonGroup），下拉框（DropDown），列表框（ListBox），滑块（Slider）**

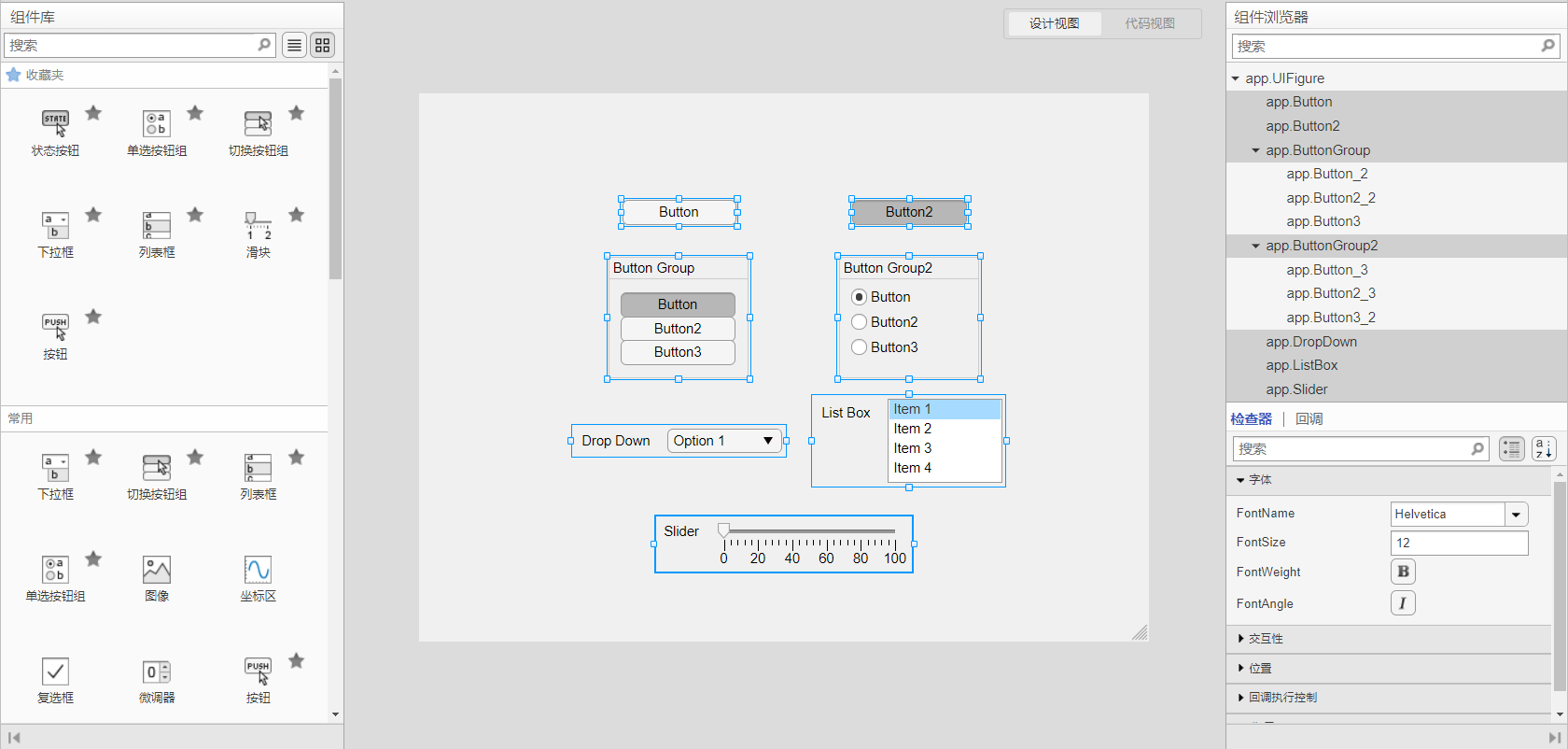


Fig. ‑ 按钮（Button）及类似组件

按钮是最常用图形交互接口。在App Designer中，用户通过点击并松开按钮，触发按钮被按下这一事件，被App侦测到后执行对应的回调函数，实现和用户的交互。许多用于交互的组件都和常规按钮类似，它们主要区别在于触发事件的形式、回调函数的参数和内容不同，开发者可以根据需要选择合适的交互组件。例如，状态按钮相较于常规按钮多出了一个状态值（Value），用户点击按钮即可触发按钮值改变这一事件，程序进而执行状态值改变的回调函数与用户交互；下拉框则有两个回调函数，分别在侦测到下拉框被展开、下拉框选项变更两个事件时被执行。无论组件的属性和回调函数有多复杂，都遵循在应用程序是如何运行节中介绍的应用程序的运行流程。通过举一反三，这类组件的使用方法很快就能够掌握。

**编辑字段（EditField）&类似组件：微调器（Spinner）**



Fig. ‑ 编辑字段（EditField）及类似组件

常用的数据交互接口。用户在文本框中输入数据或文本内容，就触发了内容变化事件。这类组件的回调函数中通常会根据应用程序获取文本框中的内容执行相应的功能。App Designer提供了两类编辑字段，分别适用于数值类型输入和文本类型输入。前者用于数值的输入，由于输入参数被限定为数字，因此便于处理；后者可接受的输入内容更加宽泛，接收的输入内容通常是一个char类型的数组，在编写其回调函数时应当注意是否需要预先处理。当然，这类组件也可以兼用于输出的显示。

**坐标区（UIAxes）：**

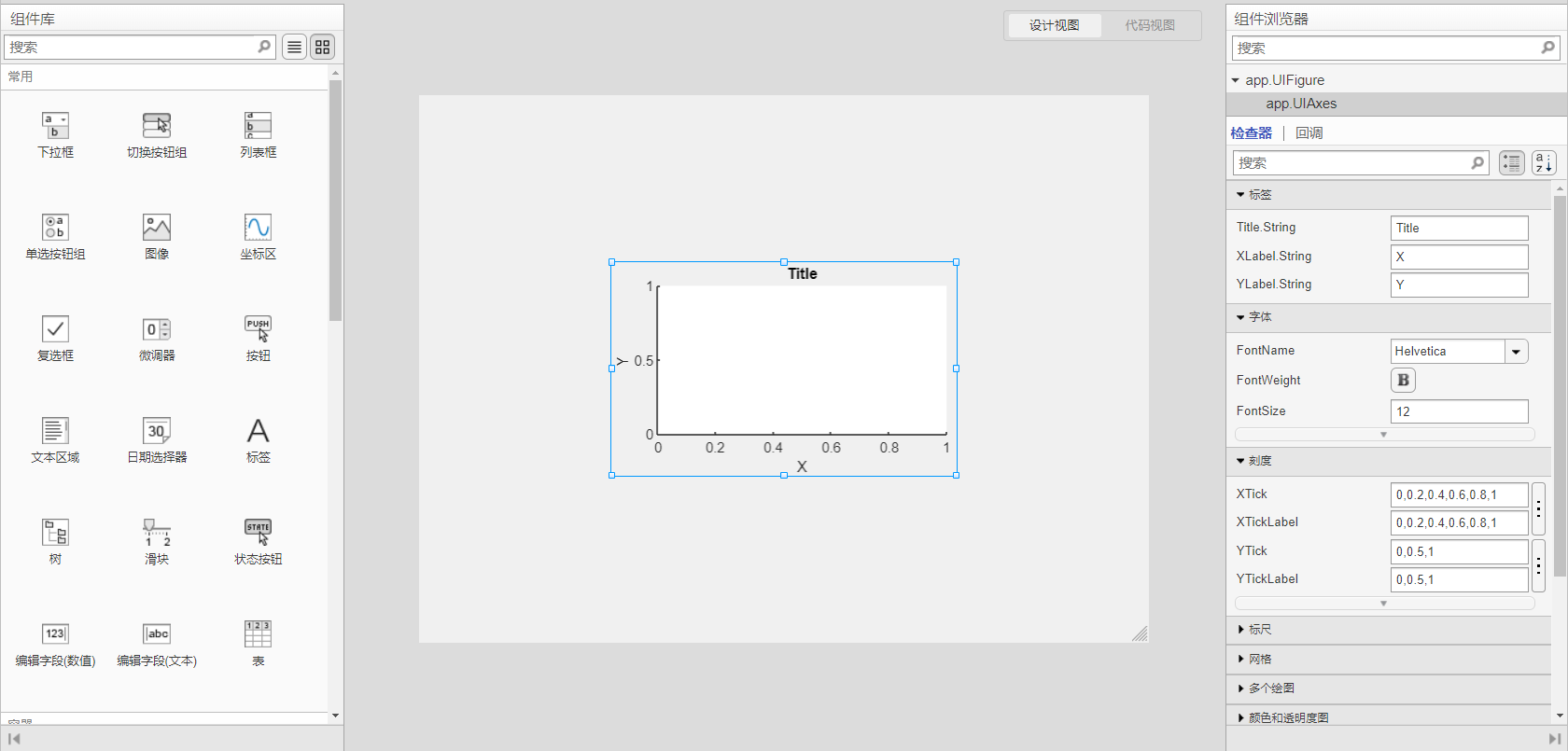


Fig. ‑ 坐标区（UIAxes）

常用的绘图显示组件。需要频繁更新的动态内容，例如实时变化的信号波形、图像等，都可以在坐标区组件中绘制。数据可视化结果和Latex数学公式等纯文本难以表达的内容，也可以在坐标区中使用MATLAB的一些绘制命令实现。坐标区可简单理解为在App的UI图窗中定义了一个绘图坐标系，在该坐标系中可以运行MATLAB中的各种绘图命令，只需要指定是在哪一个绘图坐标区绘制，就可以在多个绘图坐标区中正确展示不同的内容。

**网格布局（GridLayout）：**

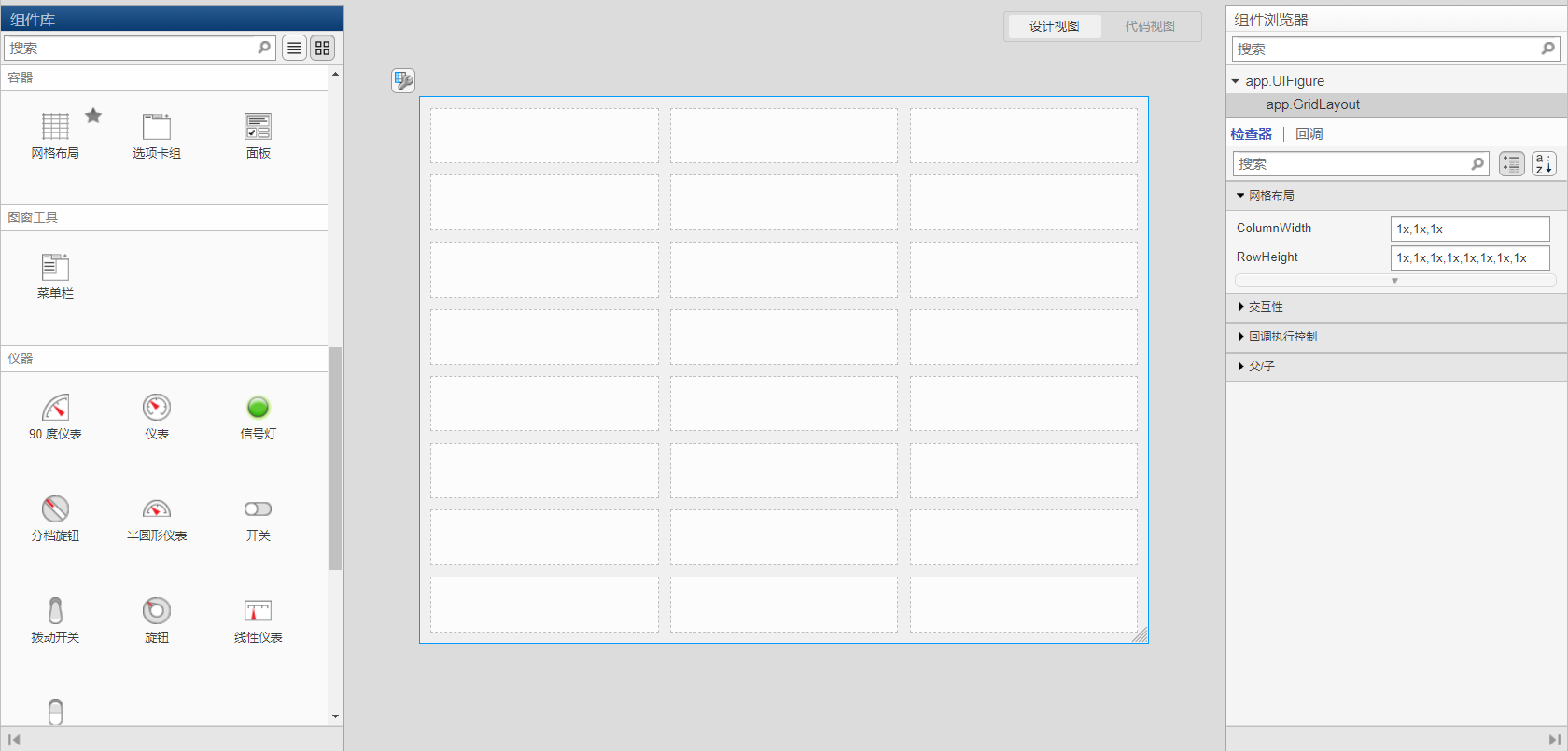


Fig. ‑ 网格布局（GridLayout）

网格布局是MATLAB R2018b中推出的组件，主要用于排版设计。为窗口或面板添加了网格布局之后，设计区域将会被网格化，区域内的所有组件的大小和位置都被网格限制。此时，我们可以通过调整网格的大小、网格间距、边缘距离等属性来调整内部组件的大小和位置。这样虽然导致单个组件的调整灵活性降低了，但是方便组件位置的统一布局调整，无需再对组件逐个进行对齐操作。读者可以根据开发规模决定是否需要采用网格布局。

注意：网格布局的操作具有不可逆性，移除窗口或面板的网格布局会同时移除内部组件；R2018a和更早版本的MATLAB将无法识别网格布局，导致App无法正确运行。如果读者需要开发能在这些版本使用的App，请不要使用该组件。

完成本章的内容后，对一个App Designer的开发项目，读者应当能够对其各个组成部分有一个清晰的认识，并能够大概了解应用程序运行过程中，各个部分是如何和用户交互的。在这一前提下，读者可以有的放矢地学习MATLAB中提供的开发例子，对想要实现的一项功能，快速、准确地定位到相关的组件和代码，高效掌握具体的实现方案。在本章的最后，我们将以App Designer引导界面提供的3分钟学习教程作为一个简单的例子，展示App Designer的典型开发流程。

## 一个简单的例子：3分钟学习教程

本节将按照App Designer的3分钟学习教程开发一个简单的示例App。读者可以在App Designer中实际体验这一过程。

点击引导界面上的【启动教程】，App Designer为我们新建了一个工程tutorialApp.mlapp，如【fig#】所示。



Fig. ‑ App Designer引导界面的3分钟学习教程

STEP1 提示文本给出了设计目标：构建一个App，用滑块控制绘图，即App有一个滑块组件，用户拖动滑块到不同位置，绘制结果会发生变化。显然，我们至少需要两个组件：滑块和坐标区。

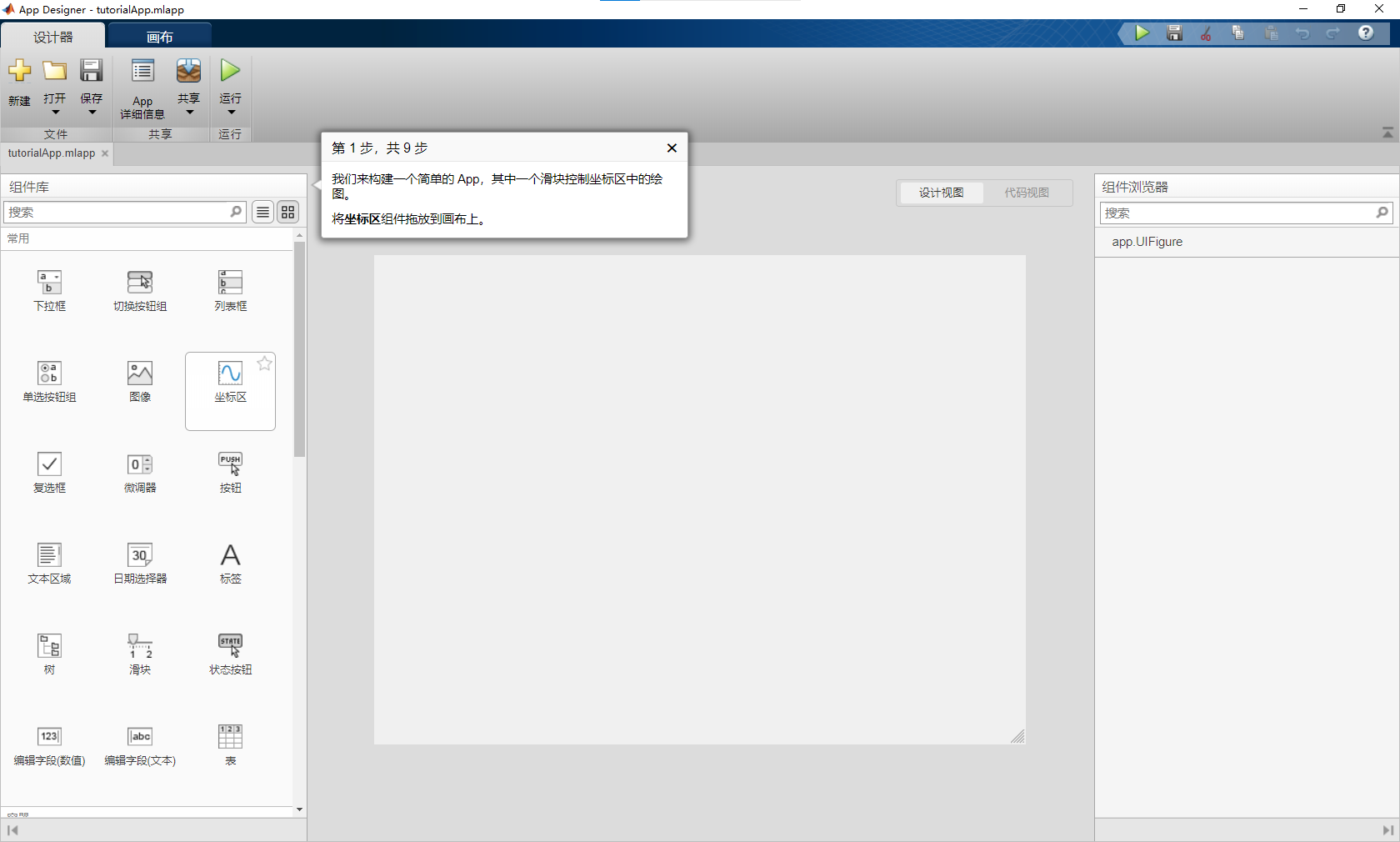


Fig. ‑ 教程第一步：创建组件

STEP2 完成App所需元素的创建工作。

STEP3 根据要实现的功能，修改组件的属性。到目前为止的工作是用户交互界面的外观设计。

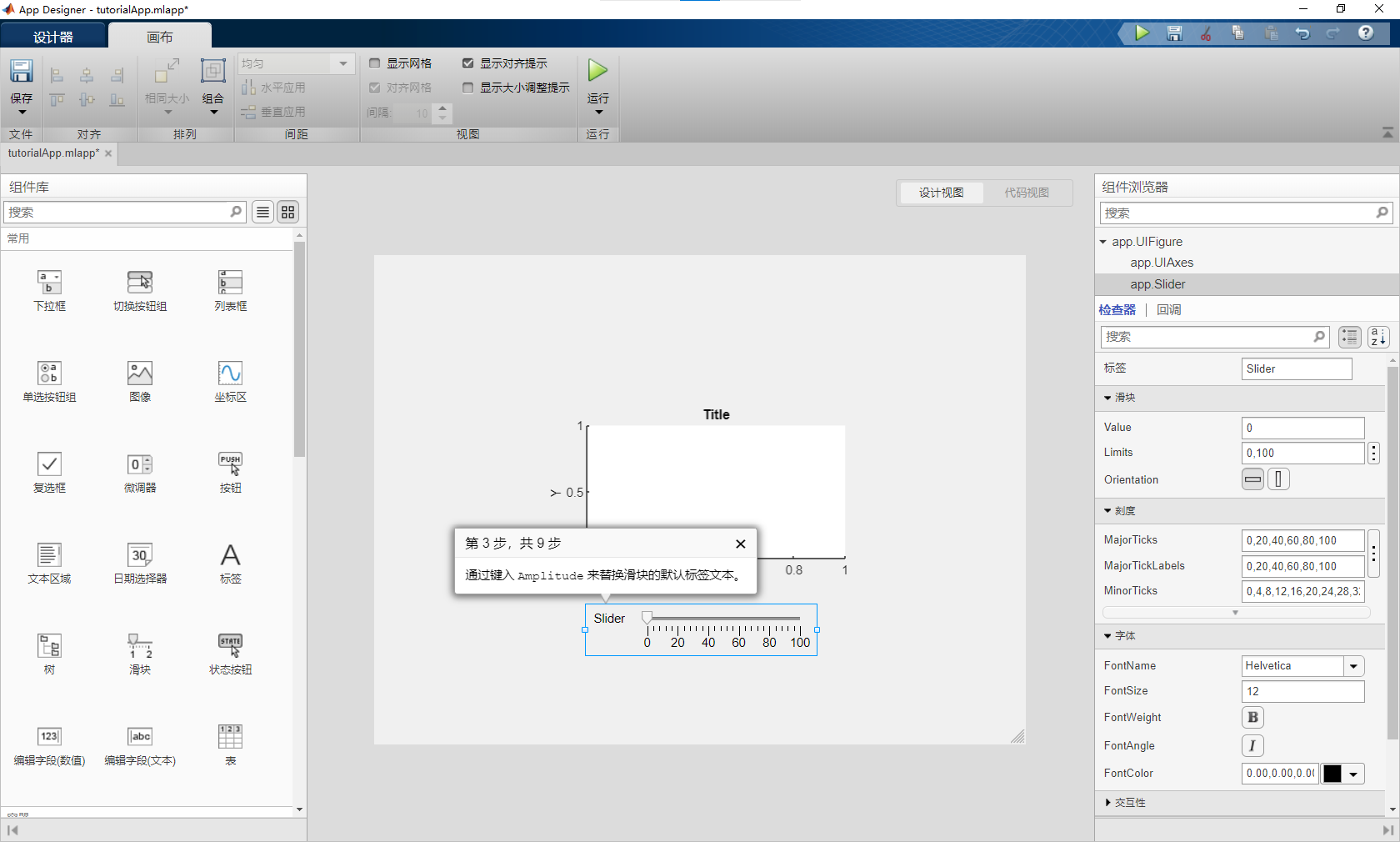


Fig. ‑ 教程第三步：修改组件属性

STEP4 现在开始实现交互本身。

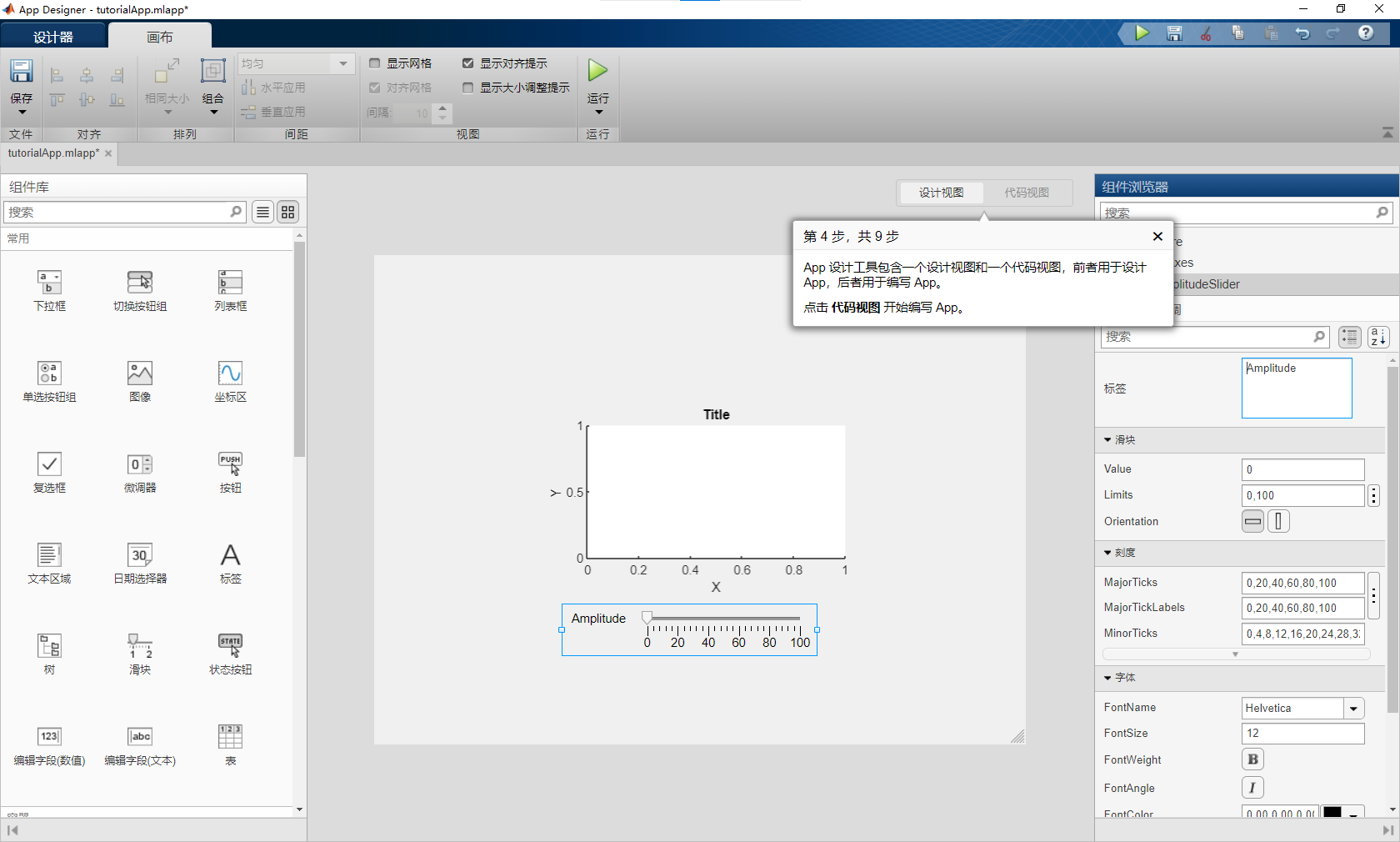


Fig. ‑ 教程第四步：切换到代码视图

STEP5 回顾上文App的运行流程，我们需要实现滑块位置变化时，改变绘图内容的回调函数。

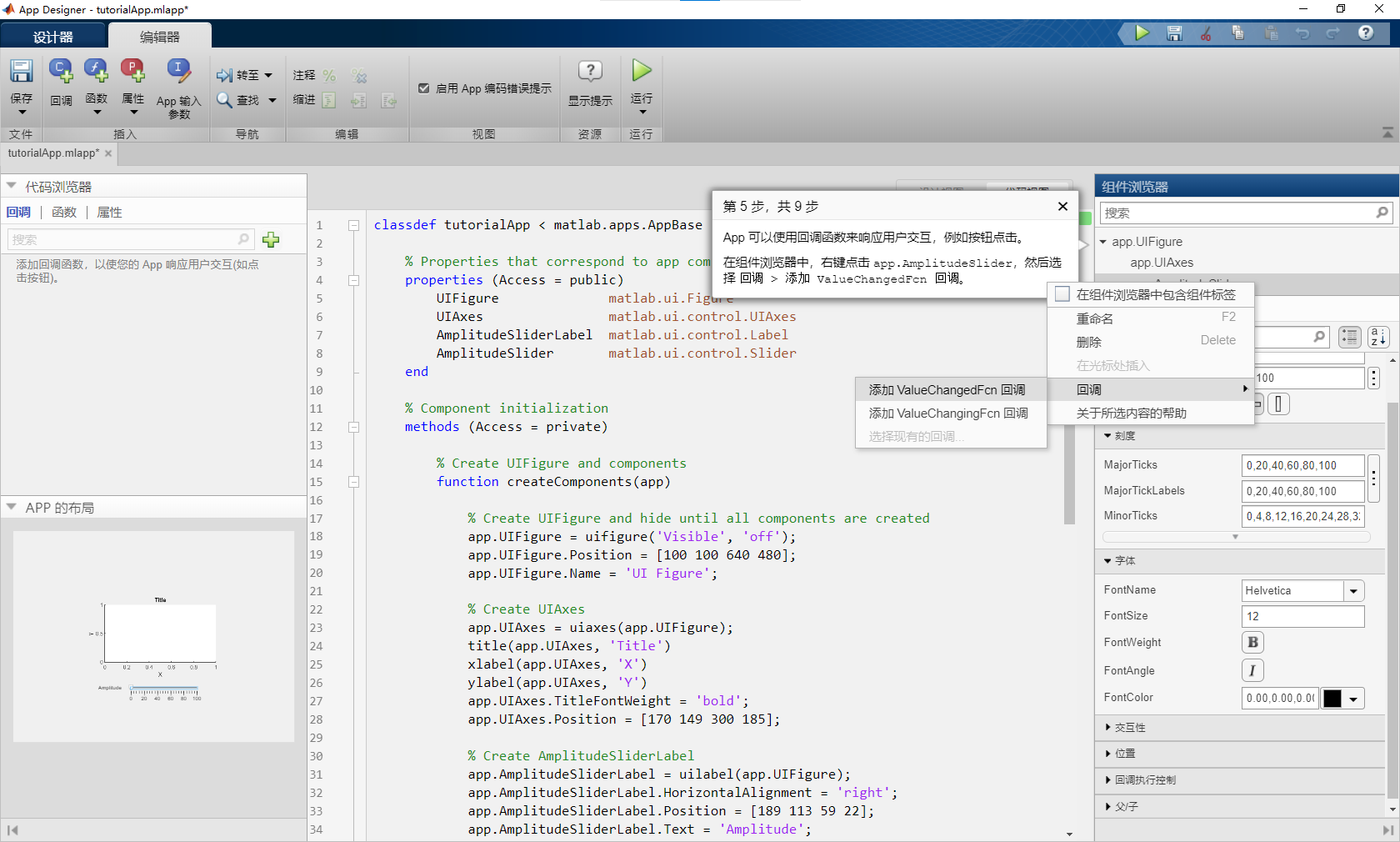


Fig. ‑ 教程第五步：创建回调函数

STEP6 根据教程指引在回调函数中输入绘图命令。

STEP7 调整绘图坐标区的属性。我们在设计视图中，可以对这些属性预先设计，不需要直接进行代码编写；但是在运行过程中需要改变的属性，我们无法这样调整，此时需要通过代码实现。

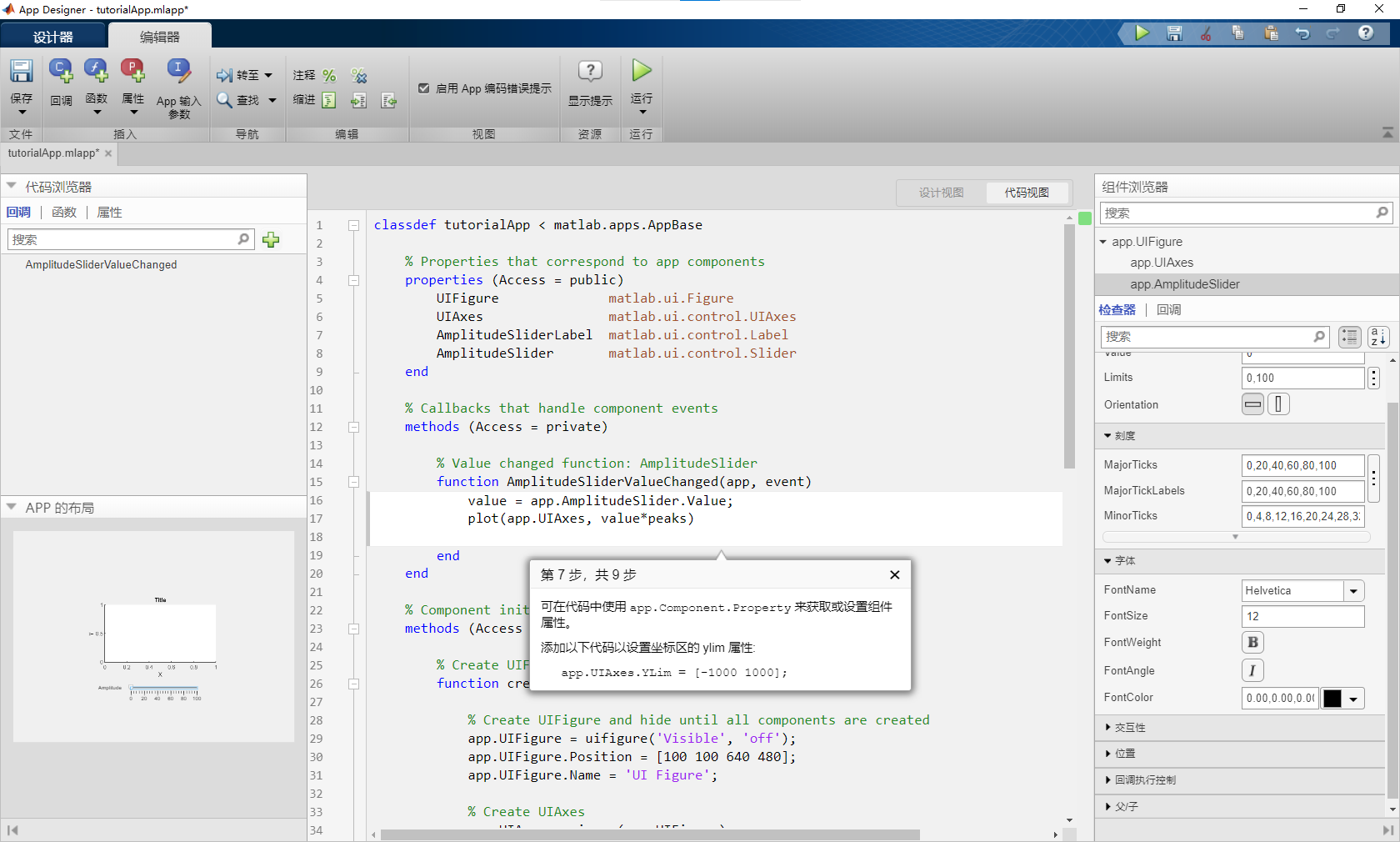


Fig. ‑ 教程第七步：代码编辑

STEP8 运行。如果事先没有保存，则会在运行前弹出保存对话框。实际开发时请适时保存，以免意外关闭工程。

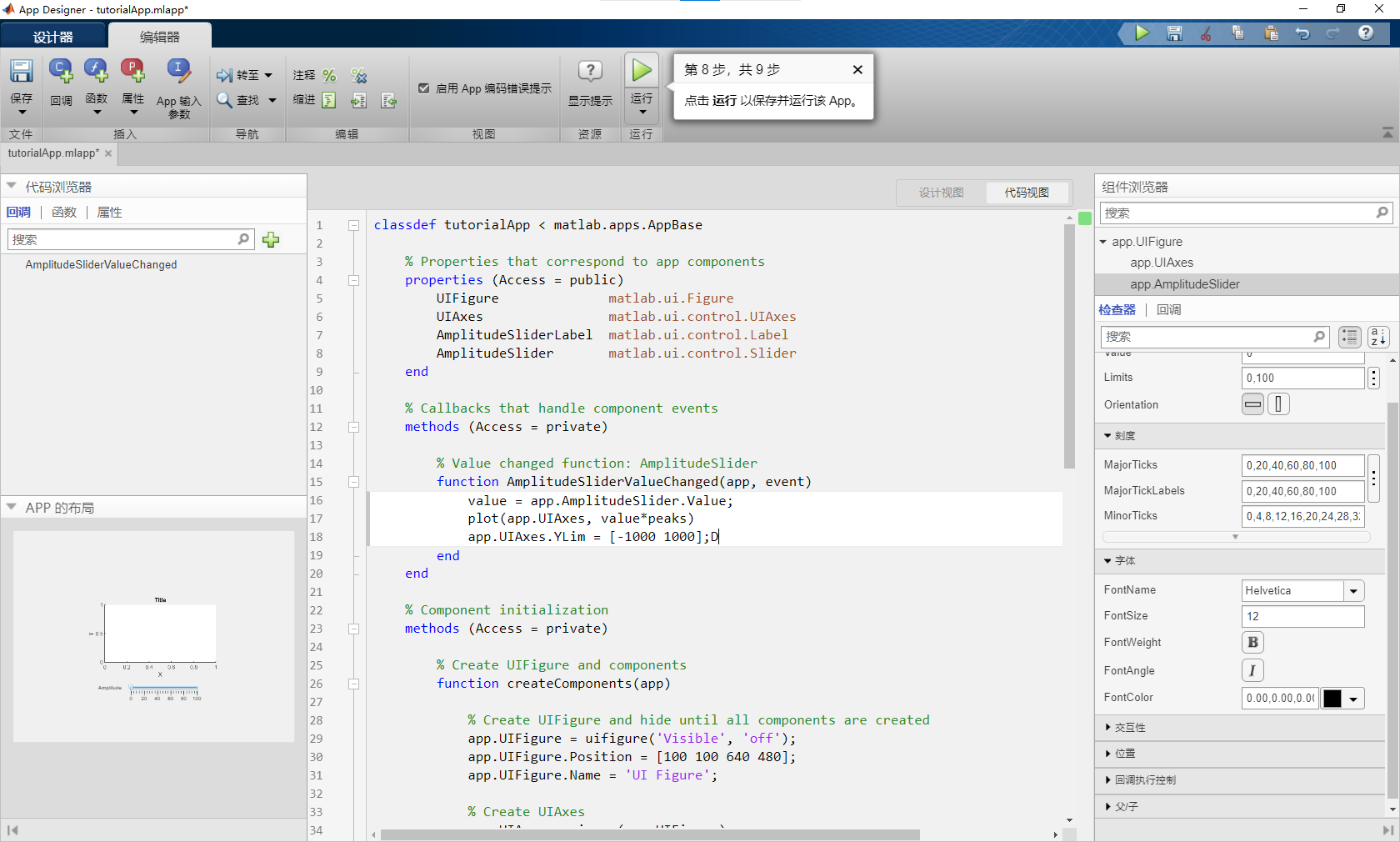


Fig. ‑ 教程第八步：运行

STEP9 测试App是否有问题。排除问题后重新测试，迭代更新直至满足设计要求。本例至此已完成了App的开发。App Designer对常用的几个模块做了进一步的说明，读者可以根据需要自行了解。App开发完成后的发布等工作将在下一章进行介绍。

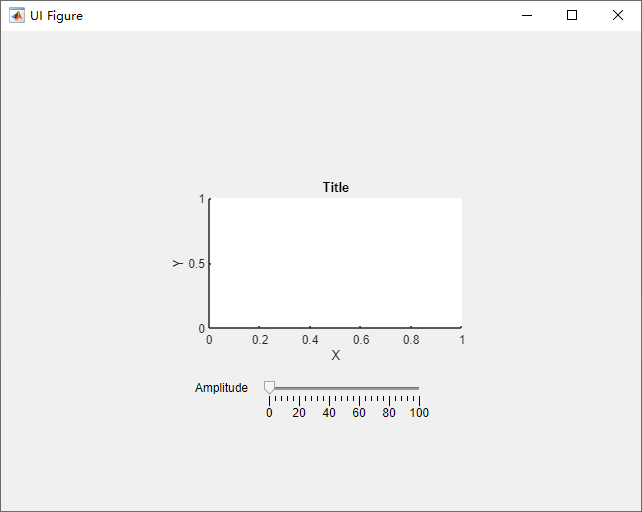


Fig. ‑ 教程App运行结果

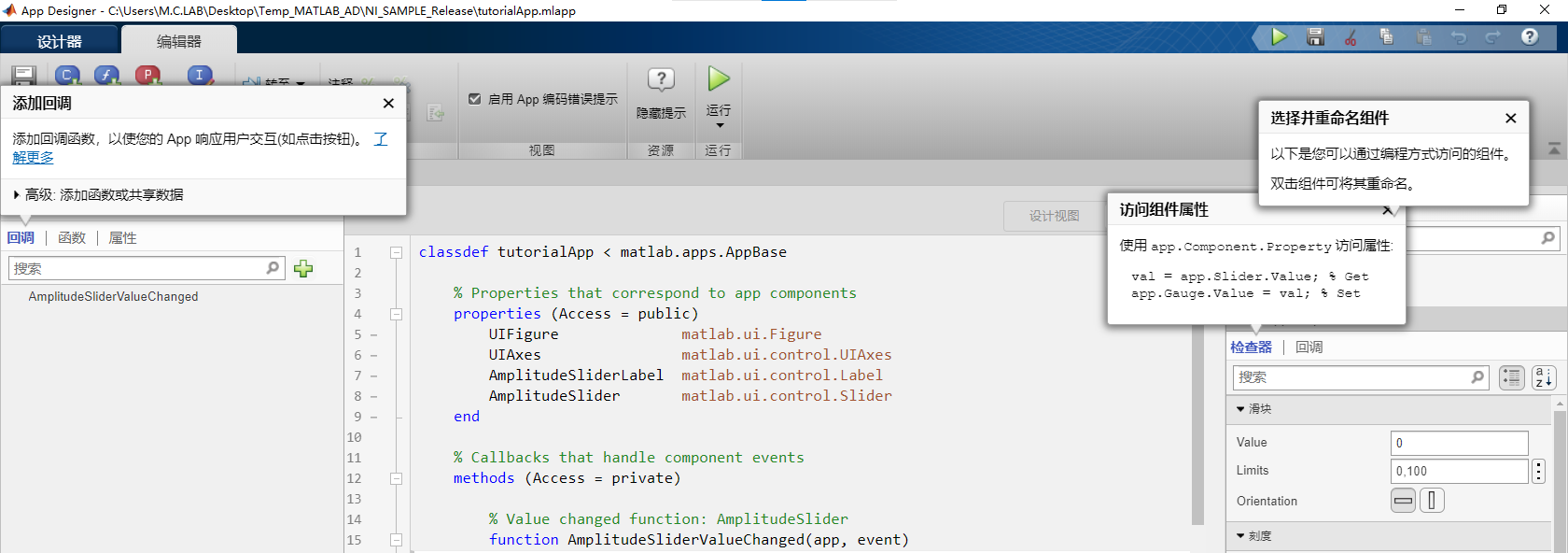


Fig. ‑ App Designer的一些重要提示信息

本章对App Designer中开发中涉及的一些基本概念和常用组件进行了介绍。读者在使用App Designer开发前，应当对这些基础性的内容有充分的了解。下一章将对App Designer的开发流程展开更加细致的介绍。

# App Designer使用入门

本章将在上一章的基础上，对App Designer的实际使用分步进行更加具体的介绍。每一节实际上对应了App开发中的一步工作，对开发中的问题，给出了一般性的解决思路。实际应用的开发和需求紧密相关，合适的方法才是最好的方法，因此本章的内容更多是一种参考，读者并不一定需要完全按照本章介绍的开发模式进行App的设计。

## 明确开发需求，并转化为具体对象

在正式开始编程之前，我们需要明确开发需求：用户希望App具备哪些功能？对每一项功能，需要提供什么形式的输入和输出接口？这些接口需要使用哪些组件，又如何对归类这些组件？我们可以先通过草图的形式，绘制App的功能框图，理清各个功能之间的依赖关系、输入和输出内容；然后根据每项功能所要求的输入、输出形式，在组件库中选择合适的组件；最后，对预选的组件进行布局，确定App支持的窗口大小和各组件的位置。这一过程可以通过【fig#】来表示。

Fig. ‑ 确定开发需求的流程

完成草图方案设计后，我们便可以直接在App Designer的设计视图中创建各组件并完成其布局。这样的设计流程可以减少因考虑不周导致组件缺失的情况的发生，节省了新加组件后调整布局、修改代码的工作量。

## 为App定义属性、实现方法

完成2.1节的工作后，App在外观层面已经形成了具体的框架。接下来，我们需要完善其内容，搭建功能层面的框架。

根据应用程序是如何运行节的介绍，App运行生成的应用程序是以AppBase作为基类实现的一个对象，为了实现App和用户的交互功能，我们需要为它定义具体的属性和方法，即成员变量和成员函数，来帮助它执行完整的运行流程。

首先我们来看方法部分。App各组件的回调函数是其主要的组成部分之一。根据我们事先绘制的草图中对各项功能的描述，我们对每一个组件，判断组件是否需要在交互过程中承担某项任务。对满足要求的组件，我们创建它的回调函数。App Designer为组件创建的回调函数将自动完成创建，回调函数的命名遵循固定的规则，和组件名称、响应事件有关，因此规范的命名将有助于开发者对App进行后续的扩展和维护。创建得到的回调函数会给出一段使用示例代码，通常会包含所需要的与响应事件有关的参数。建议读者在完成每个回调函数的创建后都为其添加说明性的注释，以明确准备通过代码实现的具体内容。

创建好的回调函数将会被记录在代码浏览器的回调栏下，方便开发者根据名称快速访问。另外，也可以通过在设计界面、APP的布局模块、组件浏览器中，右键点击对应的组件，跳转到代码视图中回调函数所在的位置。

需要注意，App Designer创建的回调函数都是App类的私有成员，无法直接在MATLAB中进行外部调用，函数内的局部变量也不会和工作区的变量共享。如果要获取函数内的局部变量，则需要将其作为函数输出参数进行传递，或者使用assignin等方法导出。

除此以外，用户也可以定义非回调函数的方法。当回调函数的实现比较复杂或是有多个回调函数包含重复的代码段时，将其代码封装成辅助函数，便可以通过调用这些子函数来完成回调函数需要执行的任务。辅助函数默认设置为App类的私有成员，供App的内部成员函数使用。在代码浏览器中可以直接创建辅助函数；也可以直接以代码形式添加，在methods内为其添加定义即可。在调用辅助函数时，函数的第一个输入参数总是app。

App的方法和App要实现的功能之间有着明确的对应关系，因此在初步设计阶段就可以很容易地将它们构建出来；相对地，App的属性则并不都在用户的交互过程中直接体现，需要开发者仔细地审视App中包括具体算法在内的整个运行流程，然后从中提取出必要的运行参数定义为App的属性。例如，某项课程要开发一个人脸变换演示App，用户输入人脸图像A和B，App将A中人脸的面容特征变换到B的脸部，并显示变换后的图像C。App的运行流程为：

1. 初始化：定义一个App类对象，生成主窗口和内部元素并显示，进入消息循环。

2. 用户输入人脸图像A和B，App接收两张图像数据并存储。

3. 用户点击按钮“人脸变换”，App调用按钮的回调函数，执行其中的人脸变换功能。人脸变换的是一个较为复杂的过程，在实现时又调用了分别用于提取人脸特征点和变换图像的两个辅助函数。具体地，人脸变换需要提取人脸特征点在图像中的坐标，分别用数组SetA和SetB记录；然后通过几何变换将A变换到C，使得变换后得到的C的特征点坐标与B的相同。

4. App得到了变换后的图像C，并将图像C显示在主窗口的坐标区中。

App的运行流程中，除了图像数据A, B, C外，还有图像的人脸特征点构成的数据SetA, SetB。提取人脸特征点的现成方法多种多样，开发时我们无法确定哪一种最合适，可能需要反复尝试，而每种方法函数的输入输出参数都可能有所不同。为了调试的便利，我们可以将一些共性的内容单独提取出来作为App属性。在本例中，就是图像A, B, C以及记录人脸特征点坐标的数组SetA, SetB。在这些App属性中，用户仅和图像数据进行直接交互，而不必关系人脸特征点究竟如何；但开发者会用到所有的App属性，因此必须充分考虑功能实现中每一步涉及的具体内容。

App的属性的定义和辅助函数的构建相似，但更加简单。在代码浏览器中可以为App添加私有属性或公有属性，默认添加的是仅限于App内部成员使用的私有属性。属性也可以在直接以代码的形式添加，在properties内为其添加定义即可。当需要调用属性时，使用前缀app.即可访问属性的值。

为App定义完善的属性，并完成组件的回调函数、辅助函数的编写后，理论上App已经可以运行并实现所需功能。但在实际开发过程中，错误和缺陷总是在所难免，通常我们还需要不断调试改进才能使App满足用户需求。这依赖于读者的反复实践和经验总结。

## 遇到不会实现的功能时，如何解决？

本节将重点讨论在开发过程中经常遇到的问题，即对某一项具体功能的实现感到无从下手时，如何一步步通过查阅文档和学习案例予以解决。对MATLAB中一般的命令操作，也可以采用类似的方法学习。

在2.1节中，我们已经对App的功能在草图上进行了描述，虽然我们可能尚不清楚该如何编写代码实现这一功能，但是功能的输入、输出，以及功能实现中涉及哪些关键变量，这些内容无关代码编辑，是可以完全确定的。读者在开发遇到此类困难时，应当先检查在草图规划阶段，是否已经将App的开发框架描绘得足够清楚。

在确认上述步骤已经没有任何问题后，我们的困难将具体定位到代码实现这一部分。在上一步我们已经确定了代码需要实现的功能的输入、输出和关键中间变量，并且在创建组件时已经为该功能指定了回调函数，那么此时我们可以根据需要合理地将它们定义为App的属性或者回调函数/辅助函数内的局部变量。完成变量定义后，剩下的工作还有方法的实现。

若读者对MATLAB不熟悉程序编写，可以先使用叙述性的语言和数学公式描述方法的实现步骤，再依次用代码实现各个步骤。细分的步骤通常能够在现有的示例或网络资料中找到解决方案，如数组的排序、数据的读取、文件的导入/导出、最小二乘法求解方程等。读者可以在MATLAB中寻找合适的示例，通过浏览示例的缩略图来确定其中是否含有类似功能的组件，对满足要求的示例进一步检视相关组件的代码；也可以借助网络搜索引擎，使用包含上述细分步骤的关键词和MATLAB词条进行检索，学习MATLAB中实现细分步骤的通用方案。

上述流程可以解决大部分在开发过程中因不熟悉MATLAB及App Designer带来的困难。如果困难仍然没有得到解决，通常意味着这将会是一个很有价值的问题，值得展开深入的研究探讨。当然，也有可能是设计的方案不合理导致后续实现的困难，此时应当回顾第一步的方案功能描述和草图规划阶段，检查是否有可以改进之处，并重新完善设计方案。

另一类困难通常出现在调试过程中，App的运行结果和预期不一致，即出现了bug。此时我们需要定位问题在代码中的具体位置，对症下药进行修复。断点调试和步进调试是常用的手段。其中，断点可以通过单击代码视图中的某行代码左侧“-”非常方便地设置或取消。此外，App Designer支持在调试过程中实时显示代码中变量的数值，只需要将鼠标停留在已赋值变量上，就可以快速查看其当前值。这一部分内容更加依赖于实践，需要读者在实际开发中积累经验。

## 测试与维护：使App尽善尽美

在App开发的中后期阶段，我们需要对App的整体开发进度进行评估，以更好地安排开发工作。具体而言，我们需要测试App的各项功能的正确性和完整性：前者指的是功能核心算法的代码是否编写正确，而后者指的是人机交互、参数传递、组件关联等“前端”代码是否完善。通常，我们要优先保证功能的正确性；在程序可以正确运行的前提下，再考虑各种可能的意外情况，尽量减少程序发生错误导致无法运行的可能。

假设我们已经完成了App的某项具体功能的代码编写工作，现需要对该段代码进行测试。在该段代码的入口和出口处设置断点后运行App，通过预设的操作，程序将停留在代码段的入口断点处。再次点击运行，若该段代码没有问题，程序将继续运行直至代码段的出口断点处。若程序意外中止而没能运行到第二个断点，或是程序运行到第二个断点处的输出和预期不一致，这意味着代码段的编写存在问题，其正确性未得到保证。此时我们应当仔细检查代码段，定位并改正其中的错误。

现假设我们通过调试已经保证了代码段的正确性。此时我们将关注点转至代码段的外围：该段代码接收的输入参数和产生的输出参数是什么？这些参数的类型和内容格式有什么要求？App是否能够保证提供给这段代码的输入参数满足这些要求？如果不满足，这段代码应当如何响应？进一步地，当我们检查App的多项功能时，这些功能运行时彼此是否会产生冲突？如果用户在代码段运行时突然关闭了App，代码段中调用的参数会如何变化？若用户重新打开App后再次运行这段代码段，是否会发生错误？……

诸如此类的问题，将App在运行中可能发生的各种情况纳入考虑，以检查App的功能的完整性。针对这些问题，我们需要依次给出解决方案，并完成代码实现。此处我们结合一个非常典型的例子加以说明：在回调函数或者辅助工具函数中需要实现某种运算，其输入参数包含用户输入的数值或文本内容（譬如计算用户输入的数的倒数）。若用户什么也没有输入，函数能否正常运行？若能运行，其输出结果是否会对后续其他功能带来问题？若用户输入了0或者其他不合法的内容又会如何？根据具体情况，我们可以通过设置组件的相关属性来限定输入参数的类型和范围，或者在功能实现的代码段前添加对输入参数的判断，对输入参数不满足要求的情况特殊处理，也可以专门定义一个辅助工具函数对输入参数做预处理，等等。对App完整性的构建工作具有很大的自由性，同一个问题存在多种可行的解决方案，读者可以从对其他功能的兼容性和维护的便利性出发，选择合适的解决方案。当然，对一项复杂的功能，其完整性牵涉的问题不计其数，对其全部做出正确的处理是不现实的。我们可以先考虑其中最常出现的问题，并针对这些问题完善App的代码；在第一版App开发完成后，通过调试、用户反馈等渠道，我们会收集到更多未曾考虑的问题，此时再对App做进一步的维护，使其尽善尽美。

建议：除了功能的正确性和完整性这两项最为重要的内容外，评估App开发是否完善的另一项因素是其交互反馈是否完善。软件开发者既需要实现用户的预期功能，又需要将各项功能有机结合在一起。这种功能的整合，依赖于良好的交互反馈：在用户执行一项操作后，App是否能够通过图形变化或文字输出等形式向用户传达操作已执行/正在进行/执行失败等等信息？对商用软件而言，优秀的交互反馈不仅仅是加分项，更是决定其用户市场的关键性因素之一。在App Designer中我们可以使用指示灯、标签、编辑字段等组件作为载体来实现简单的交互反馈。我们建议读者尽量为每项交互功能都设置合适的交互反馈，因为这能够降低软件的调试和使用难度，同时显著提升用户的使用体验。

## 编写使用说明，打包、发布

当我们终于如释重负地完成App的设计时，还不应忘记App是为目标用户设计的，因此App开发的最后一步还必须包括使用说明的编写和打包、发布工作。

在开始这项收尾工作前，请先确认是否已经完成了【工具栏】模块中App详细信息的填写。这些内容将作为发布的程序的基本信息显示给用户，包括版本号、作者、摘要说明等。

使用说明的编写可以有不同的风格，但是目的主要是让目标用户了解App的使用方法，并提供App的一些使用建议和常见问题解决方案。前者应当以简练的文字清晰表达，后者则应当给出详细完整的说明。这部分并不是本文的介绍重点，因此不再赘述。

App Designer提供了App的打包和发布功能，在【工具栏】模块可以找到执行该功能的【共享】按钮。App Designer为App的共享提供了三种方案：打包为MATLAB App、Web App和独立桌面App。

打包为MATLAB App适用于在同一版本或更新版本的MATLAB中运行，适用于小规模的、仅限于MATLAB平台用户的发布。安装包的生成和安装所耗费的时间最短，安装完成的App将显示在MATLAB的APP栏中。

Web App则是将App发布到MATLAB网络平台的预先操作。如果开发者希望将App发布到平台上供所有MATLAB用户使用，则可以使用这一功能。

独立桌面App将不需要在MATLAB中才能运行，而是作为一个独立的应用程序运行。但是，独立桌面App的运行依赖于MATLAB的运行环境，必须满足以下其一：安装了和开发所用的相同版本的MATLAB，或是安装了匹配版本的MATLAB Runtime。因此，打包文件必须包含运行环境，或是提供下载运行环境的方式。好在这项工作已由MATLAB代为实现，因此操作过程仍然非常简单。读者可以根据需要自行选择开发完成的App的打包发布方式。

# 开发项目：基于Hall传感器阵列和NI采集卡的位移测量App

本章将对一个更加具体的实际开发例子进行介绍。开发项目被应用于《测试与仪器》课程实验演示。该实验的硬件系统主体是一个直线位移测量平台，使用的测量设备为多个Hall传感器组成的阵列。通过NI采集卡，获取传感器的实时数据，并传输到计算机中。对数据的处理和位移的解算在MATLAB中完成。同时，作为测量解算结果的参考，位移测量平台上安装了磁栅尺。要求开发一款名为Linear Displacement Measurement Platform（LDMP）的App，具有如下功能：Hall传感器和磁栅尺读数头的数据的采集和实时显示；数据的导出；NI采集卡采样率的调整；绘制Hall传感器读数和磁栅尺读数（参考位移）之间的关系曲线；绘制上述关系曲线的频谱图。以下我们将从草图设计开始，逐步实现App的各项功能。

## 初步设计：模块划分和App创建

首先，根据App的基本功能需求，我们可以以草图的形式绘制App的基本框架，将App分为三个模块，如下图：

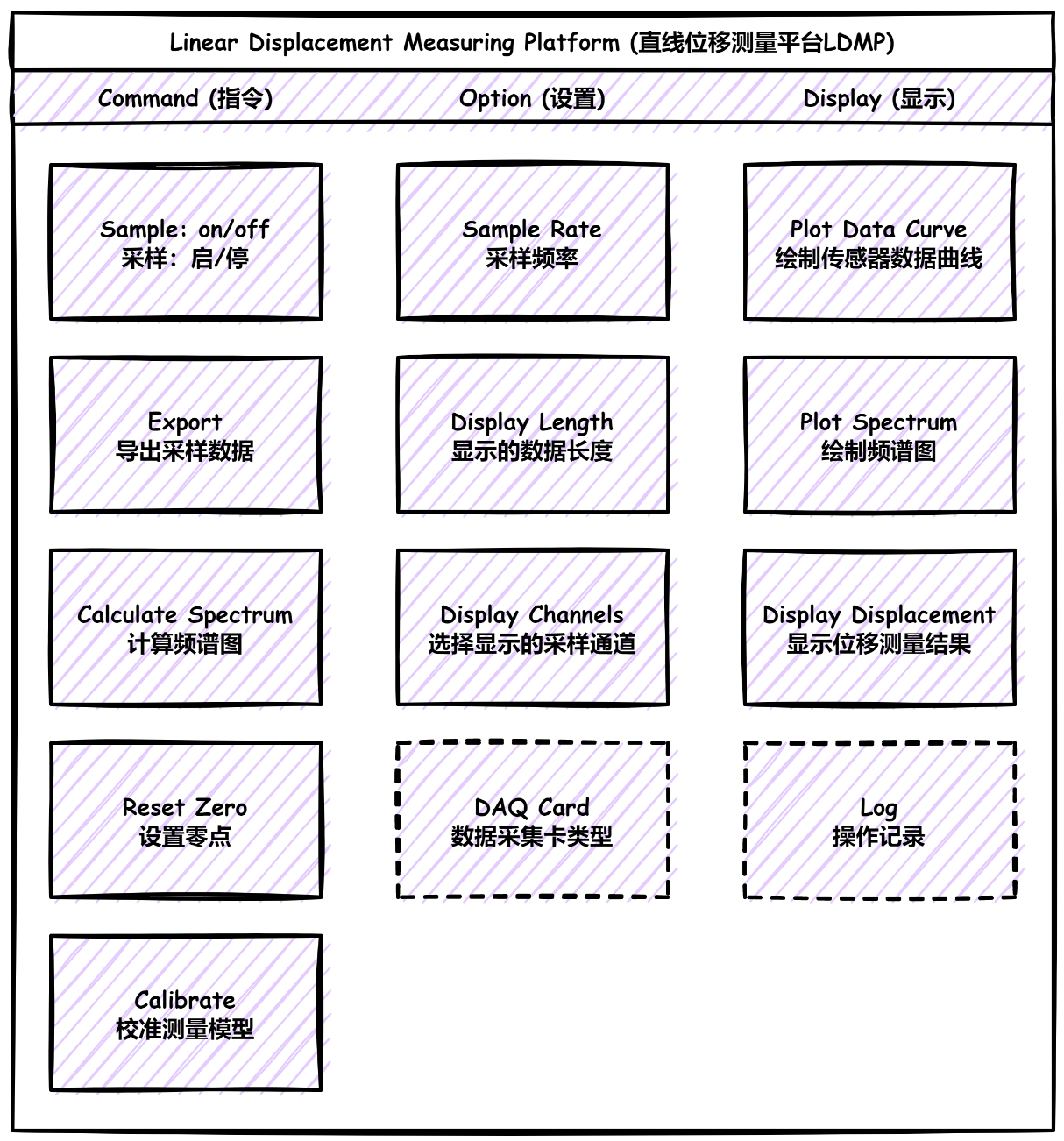


Fig. ‑ 设计草图：App的框架

三个模块之间的主要区别是和用户的交互方式。因为每个模块内的组件和用户的交互方式类似，因此可以采用类似的方法，并放在同一个面板中布局。具体地，每个模块的构建思路如下：

1. 指令：用户通过这个模块中的组件向App发出指令，App接收指令执行特定的任务。采样的开始/停止、数据导出、计算频谱等功能符合这类交互模式，可以通过按钮类组件实现。

2. 设置：用户通过这个模块向App传递参数，App接收参数后存储在内存中。例如，App在执行采样时，需要用户指定采样频率；App绘制Hall传感器阵列的信号时，用户可能并不需要查看所有传感器的数据，此时则需要用户指定想要查看的数据通道选择性绘制。这些交互过程中，用户向App发送的信息较为复杂，通常需要通过编辑字段类组件实现。

3. 显示：App需要向用户提供传感器数据和频谱分析等内容的可视化结果。用户通常并不直接向该模块发送指令，而是通过指令和设置模块简介改变显示模块的内容。坐标区可用来实现绘图功能；标签、编辑字段等组件可以用来显示App的数值计算结果，但是要注意这些组件的内容仅用于显示，不能被用户直接修改，即需要将组件的可编辑属性（Editable）设置为false。

注意：虚线框对应的模块子功能并不是必要的，但是添加这些功能将显著改善用户的使用体验。在初步设计阶段，读者可以先不考虑这些功能的实现，而是在后续的改进版本中完善。因此，在下文具体设计工作的介绍中，将不包括这两个部分。

至此，完成了App的整体框架的构建。接下来我们需要根据App Designer中提供的可用组件，进一步细化每个模块中的内容。下图是一种可能的细化方案（组件不包括用于显示静态信息的标签）：

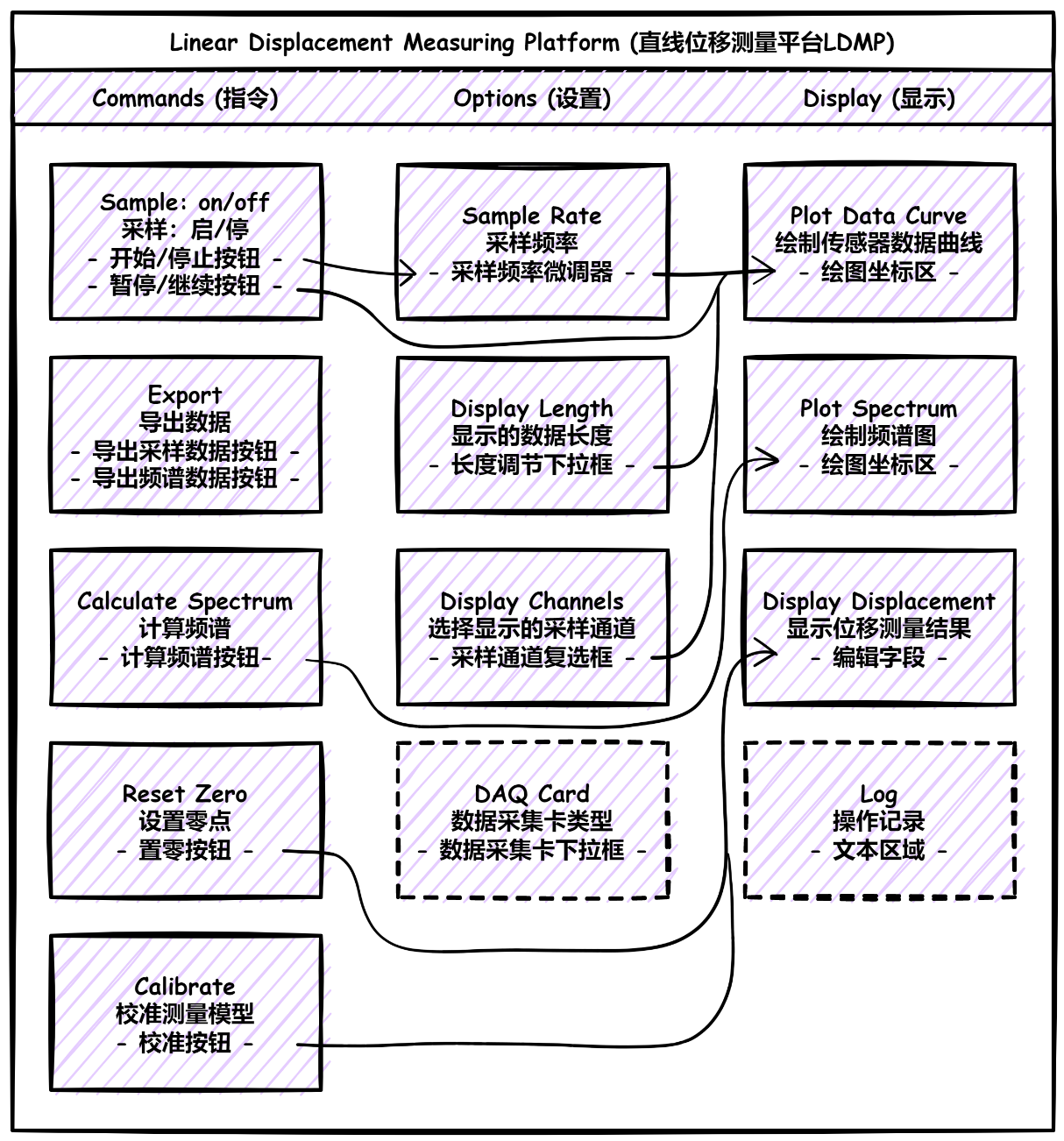


Fig. ‑ 设计草图：App的关键组件及组件间的依赖关系

在上图中，我们明确了每个模块会使用到的组件，并且将组件之间的依赖关系也用实线连接的方式表示了出来。此时，我们可以在草图层面对App的设计合理性进行检验：用户每一次交互所调用的回调函数，都属于某一个特定的组件，根据组件和构建的依赖关系，App是否能够接收来自用户的信息并正确执行相应的功能。例如在用户选择Hall传感器采样通道后，绘制传感器实时数据和Hall传感器相对参考位移的曲线应当和选择的通道对应，因此用于更新采样通道选择的回调函数中必须对绘图组件进行操作；又如App在完成传感器采样信号的频谱计算后，需要将计算结果绘制出来，因此在执行计算频谱的回调函数中需要对绘制频谱图的组件进行操作。

上述设计的目的在于尽可能减少App开发过程中因功能实现不完整造成反复修改甚至改换整体方案的情况的发生，从而提高开发的效率。在经过草图阶段的功能性和完整性检验后，我们可以正式开始在App Designer中进行App的构建。

在App Designer中，新建App，单击【工具栏】中的【App详细信息】，并在其中完善App的信息：

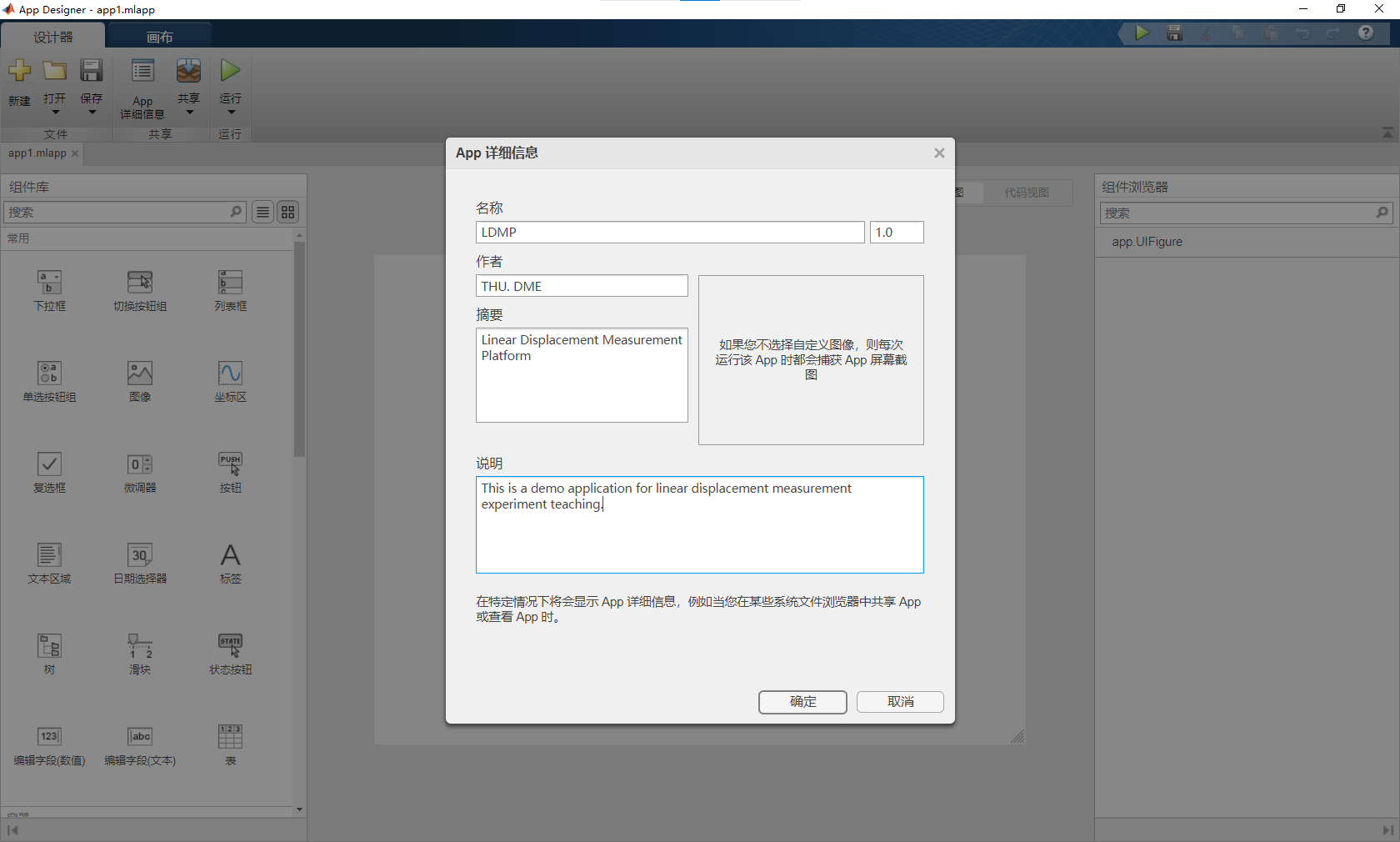


Fig. ‑ 完善App的信息可以帮助我们明确开发目标

完成App的创建后，单击【保存】，将项目文件保存到合适的位置。开发过程中也需要注意及时保存，若需要尝试并比较多种不同方案的优劣，则请采用不同的名称另存为项目文件进行备份。

根据本节给出的设计方案中的模块和组件配置，我们可以很快完成App中组件的创建，如下图：

【更新】

Fig. ‑ 完成组件创建的App界面

其中使用了选项卡组、网格布局和面板来进行外观设计。

接下来完成组件初始化属性的配置。调整外观相关属性以使App美观易用；与功能实现相关的属性则应根据需求仔细确定。在LDMP中，这些属性主要包括：

指令模块中，按钮类组件的启用状态（Enable），状态按钮还需要确定其初始值（Value）：例如，当开始/停止按钮为Off状态时，需要禁用暂停/继续按钮等组件；当开始/停止按钮转变为On状态时，则需要禁用采样频率等组件。

设置模块中，采样频率、显示长度和采样通道的初始值（Value）；特别地，采样频率组件的启用状态（Enable）和范围、数据格式等都需要仔细设定。

显示模块中，数据和文本显示组件的可编辑性（Editable）：位移测量结果、操作记录显示栏等组件只能用于显示信息，不能被用户修改。

显示模块中，坐标区绘图的标题、横轴和纵轴名称等外观属性。

## 为App定义方法和属性

完成组件的创建工作后，下一步是对功能的具体实现，即完成App的方法和属性的定义和实现。

在已经完成的草图设计中，可以直接提取出所需的方法，即回调函数。每一个指令模块和设置模块的组件，都对应一个或多个回调函数，用来实现用户交互的方法，回调函数涉及的关键组件则可以通过依赖关系确定。将所有回调函数进行整理，得到下表：

Tab. ‑ LDMP的回调函数一览

|  |  |  |
| --- | --- | --- |
| 回调函数 | 目标功能 | 涉及的关键组件 |
| startupFcn | 完成App的创建后执行一次，用于初始化属性等工作 | / |
| LDMPUIFigureClose Request | 关闭App时执行一次，用于关闭子进程等工作 | / |
| SwitchValueChanged | 单击后，创建/删除采样对象  创建采样对象后，App将以一个固定的频率更新显示内容 | 指令模块：采样启停  设置模块：采样频率、采样通道  显示模块：绘制数据、绘制曲线 |
| PauseButtonPushed | 单击后，暂停采样和显示内容更新，按钮文本从Pause变为Resume。再次单击后将继续采样并更新显示内容，按钮恢复原状 | 指令模块：采样启停  设置模块：采样频率、采样通道  显示模块：绘制数据、绘制曲线 |
| ExportButton\_ CommandsPushed | 单击后，导出采样数据 | 指令模块：导出数据 |
| DisplayLengthDrop DownValueChanged | 改变该下拉框的选项后，将改变绘图的数据长度 | 设置模块：显示调整  显示模块：绘制曲线 |
| HallSensor\_1 ValueChanged  ~  HallSensor\_8 ValueChanged | 勾选/取消后，将绘制/不绘制对应通道的Hall传感器信号 | 设置模块：显示调整  显示模块：绘制曲线 |
| ResetZeroButton Pushed | 单击后，将当前位置设置为位移零点 | 指令模块：位移置零 |
| CalibrateButton Pushed | 单击后，校准位移测量模型 | 指令模块：校准模型 |
| CalculateSpectrum ButtonPushed | 单击后，计算传感器数据的频谱并绘制结果 | 指令模块：计算频谱  显示模块：绘制频谱 |
| ExportButton\_ SpectrumPushed | 单击后，导出频谱计算结果 | 指令模块：导出分析结果 |

在App Designer中，按Tab. 3‑1依次创建各个组件的回调函数。在为每个回调函数添加代码实现之前，我们可以还需要完成App属性的定义。

注意：在2.2节中我们也介绍了辅助函数。辅助函数并不是必要的，但是能够在代码规模较大时将代码进一步根据功能细分，避免造成混乱；对一些需要重用的代码段，定义辅助函数则可以减少代码编写的工作量。在目前的设计阶段，我们还不清楚需要定义哪些辅助函数，因此这些内容可待对回调函数进行代码实现时再考虑。

重新审视草图设计方案和回调函数，对其中有必要存储的数据进行定义。经过整理同样得到如下包含所有关键属性的表格，在表中对属性的类型和定义该属性的原因也做了说明：

Tab. ‑ LDMP的属性一览

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 具体含义 | 类型 | 定义该属性的原因 |
| EncoderRange | 磁栅尺编码器读数的最大范围 | double类型的scalar | 直接得到的编码器的读数是符号整型变量，超过该最大值时会溢出成为负数导致结果跳变，此时需要对采样结果加/减该值将其调整到统一的范围内，因为是一个固定值，所以预先存储 |
| EncoderRatio | 磁栅尺位移和读数的比例关系 | double类型的scalar | 求解位移时需要使用，因为是一个系统固定参数，所以预先存储 |
| DisplayChannels | 用于显示的Hall传感器的采样通道 | bool类型的vector | 用来记录需要向用户显示哪些通道的数据。用户通过设置模块的组件交互改变该属性，并传递到用于显示数据的回调函数中 |
| PlotLegend | 绘制数据时显示的图例 | 由char array组成的cell array | 内容由DisplayChannels决定。将它作为一个独立的属性存储，避免每次绘图都重新从DisplayChannels中更新 |
| PlotLength | 用来限制绘图的点数的上限 | double类型的scalar | 若绘制的点数太多，App可能达不到预期的刷新频率，因此需要设置一个这样的属性限制用于绘图的数据量 |
| RecordData | 用于存储采集卡数据的结构体 | struct | 为了方便存储数据的扩展和统一管理，将这些数据存储在一个结构体中 |
| RecordData.*Time* | 采样数据对应的时间 | double类型的vector | 这些数据会被用于显示结果的计算和导出 |
| RecordData.*Hall* SensorData | Hall传感器的采样数据的 | double类型的matrix | 这些数据会被用于显示结果的计算和导出 |
| RecordData.*Encoder Data* | 磁栅尺编码器的采样数据 | double类型的vector | 这些数据会被用于显示结果的计算和导出 |
| RecordData.*Length* | 存储的采集卡数据的长度 | double类型的scalar | 用来实时记录数据的长度，为数据管理和代码编写是提供便利 |
| CalibrateData | 用于存储标定好的模型参数 | double类型的scalar；或者struct | 标定完成的模型参数可以作为App的属性存储起来，避免频繁的函数参数调用 |
| DataSpectrum | 用于存储信号频谱的相关数据 | double类型的scalar；或者struct | 信号频谱的数据可以作为App的属性存储起来，避免频繁的函数参数调用 |
| EncoderZeroValue | 磁栅尺编码器的零点 | double类型的scalar | 磁栅尺只能测得相对位移，因此需要设定一个零点作为基准并存储，用于参考位移的计算 |
| SampleSession | NI采集卡的数据采集对象 | Session object | 定义一个该对象后，才能使用对象中的方法获取采集卡的数据 |
| SessionListener | 用于接收NI采集卡数据 | Listener object | 作为SampleSession对象的一个方法，用来接收采集卡的数据 |
| VirtualSession | 未连接数据采集卡时，模拟数据采集对象创建的定时器 | Timer object | 与SampleSession类似，用来模拟传感器数据的采集过程。定时器的回调函数TimerFcn则实现了类似SessionListner的功能 |
| SessionTimer | 用于实时显示和计算的定时器 | Timer object | 在App运行状态下，需要以固定的频率实时刷新显示模块 |
| IsCreated\_SS  IsCreated\_VS  ISCreated\_ST | 用于记录数据采集对象或定时器对象是否已被创建 | bool | 这些对象在App启动后，只需要创建一次。为了避免重复创建，使用bool变量记录它们的创建信息 |
| PlotLine\_HS  PlotLine\_E  PlotLine\_HSE  PlotLine\_FFT | 用于记录每次绘制的线条的数据（分别对应：时间-Hall传感器数据、时间-磁栅尺编码器数据、两类传感器的数据关系、幅值谱） | Line object | SessionTimer每次调用回调函数更新绘图区时，从PlotLine中读取需要绘制的线条的数据 |

提示：并不一定所有预先定义的属性都会被用到，但是在代码编写前进行充分的考虑仍然非常重要，而且这不会带来坏处。事实上，在本例中，PlotLegend和PlotLength这两个属性在最后的设计方案中没有被使用，因此在程序代码中被注释。但是，如果要添加相关的拓展功能，只要取消注释，这些属性立刻就能够被应用到相应的程序中。

开发者需要尽早明确属性的类型。在后续的扩展开发中，需要增加新的属性或修改原来的属性、重新完善回调函数是非常普遍的情况，但若能够定义基本的方法和属性使其具备较好的可扩展性，则可以显著节省后续开发的时间和精力。在属性的定义中，建议读者仔细考虑每个属性应当定义的类型：如果属性是变量，那应当定义为单个标量，还是数组，还是矩阵，还是更一般的元胞数组？如果属性是对象，那应当为它定义哪些成员？如果在后续的开发中，属性类型需要扩展，则由于不同属性类型的计算方式不同，对使用这些属性做参数的方法实现可能需要进行较大的修改。

在MATLAB中，不需要我们显式地对每个属性的类型做出定义。对double类型变量的属性，在我们为其设置初值时，MATLAB会自动为其确定类型；对结构体变量的属性，我们可以使用struct命令定义其成员变量的名称和初值；对由MATLAB中的其他类创建的对象，我们根据帮助文档中类的使用说明为其设置初值。根据Tab. 3‑2，我们完成了App的属性的定义，如下：

【code】

部分属性如绘图线条PlotLine\_[TYPE]的初始化需要设置较多参数，我们可以将这部分内容放在App在运行开始时执行的回调函数startupFcn中执行：

【code】

至此，App的所有组件的框架都已经搭建完成。下一步的工作是的代码实现。这部分将涉及MATLAB的编程工作，但是每个函数的输入、输出和要实现的功能都已经实现在方案设计中给出，因此在可实施性已经得到了保证。本文将不再对这一部分做详尽的介绍，仅给出一些关键功能的实现作为参考。在代码实现过程中遇到的困难，可参照2.3节的内容解决。

## 关键功能实现：NI采集卡数据的获取和显示

NI采集卡数据的获取和显示实际上是LDMP中最为重要的功能。事实上，App在开启采样功能后，就在以固定的频率循环接收采集卡数据，因此绘图、解算位移的显示等功能都应当在这个循环中执行。本节我们将完成该项功能的实现。

由于本项目使用的是NI的数据采集卡，MATLAB中对NI数据采集卡数据的获取可以使用Data Acquisition Toolbox（以下简称DAT）来实现，从而节省了软硬件通信的开发工作。

在本开发项目中，我们从NI采集卡获取的数据包括Hall传感器的模拟量信号和磁栅尺编码器信号。根据DAT的说明文档，我们可以分别找到二者的使用例子。

对模拟量的输入输出，在DAT中，选择【Analog Input and Output】，了解其代码实现所需的命令。考虑到我们由实时绘制数据图的需求，【Acquire Data in the Background】是我们最需要了解的内容。在文档中，MATLAB的给出了实现这一功能的具体说明。

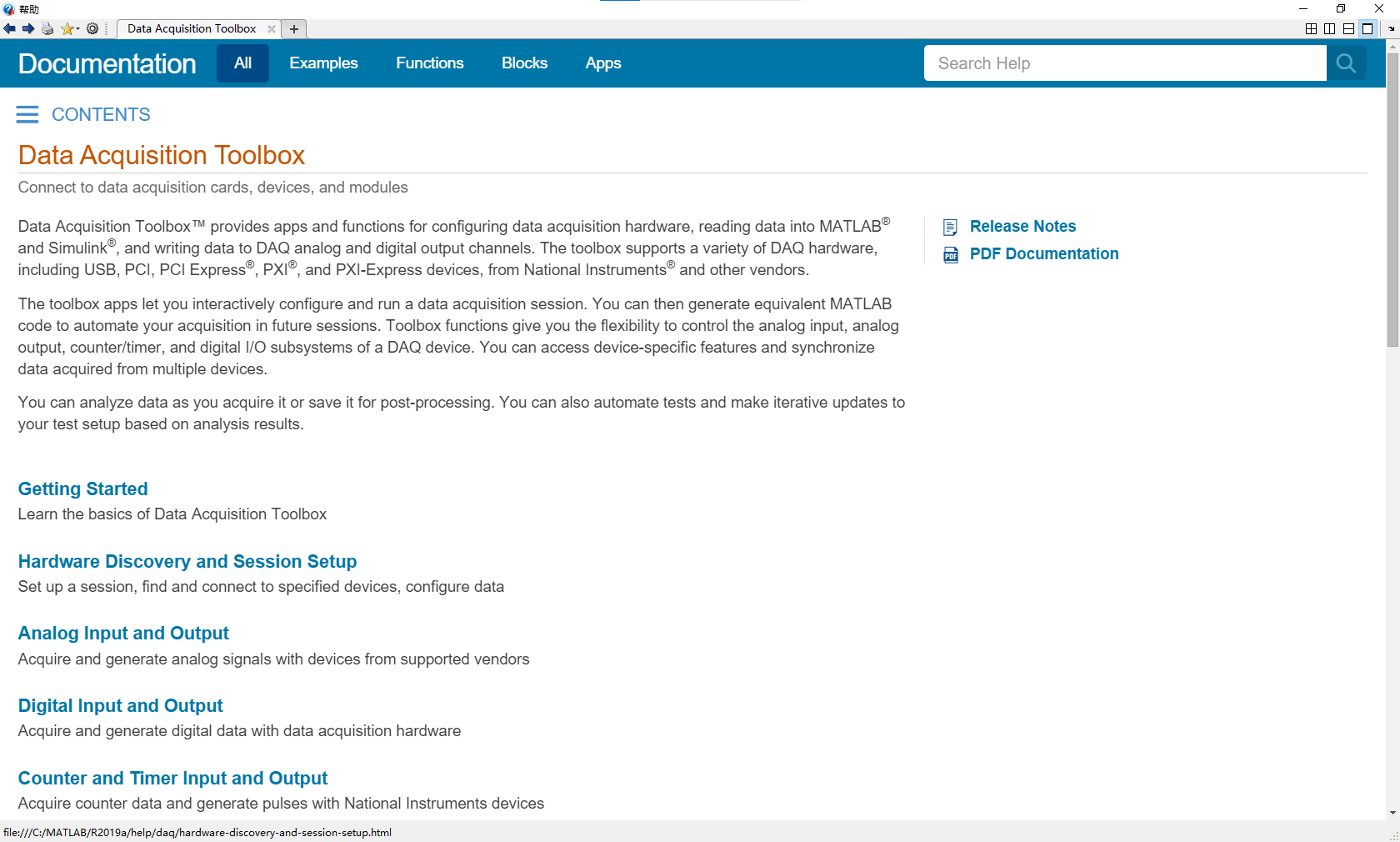


Fig. ‑ DAT的帮助文档主界面

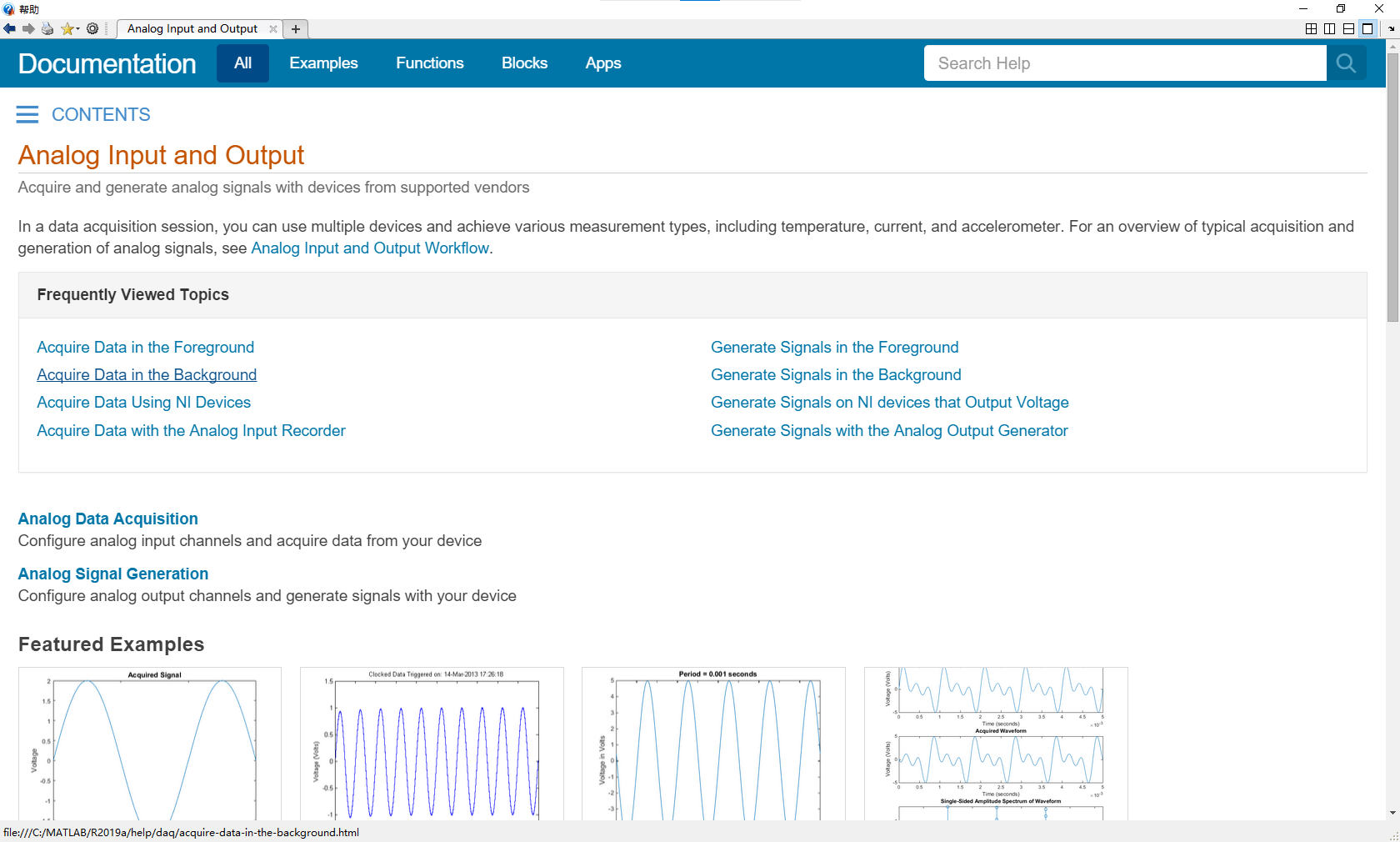


Fig. ‑ 与获取Hall传感器数据的解决方案相关的帮助页面

通过阅读文档，我们了解到daq.createSession和addlistener是用于实现采集卡模拟量信号获取的关键命令，前者用来创建获取模拟量的对象，后者用来创建侦听采样数据的对象。完成定义后，使用startBackground命令在后台不断获取数据。整个流程在【Analog Input and Output WorkFlow】中由【fig#】描述。但是，这些命令的具体使用方法和参数的配置还不明确，因此我们进一步寻找这两个命令的帮助文档和合适的例子来学习。

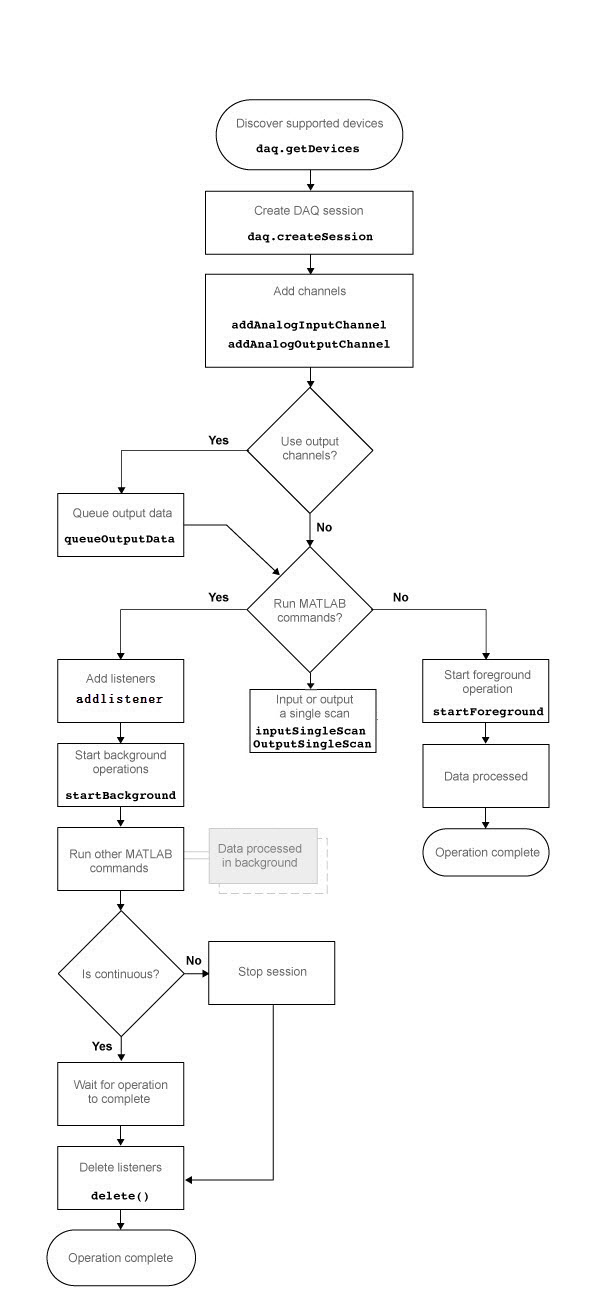


Fig. ‑ 使用DAT实现模拟量输入和输出的工作流程

在daq.createSession的帮助文档中，给出了该方法的使用格式：

session = daq.createSession(vendor);

其中，session就是我们需要创建的数据获取对象，而vendor则与我们所使用的采集设备有关。在本例中，我们使用的是NI的采集卡，对应在可选项中的'ni'。创建session后，我们还需要正确配置其参数。继续阅读帮助文档，在Output Arguments这一部分中，给出了session的所有属性。逐条查看这些属性的帮助文档，理解每个属性的具体含义，并结合我们的开发需求——实时获取并绘制采集卡数据，找出其中需要配置的属性：

Tab. ‑ 本例中data acquisition session需要配置的属性

|  |  |
| --- | --- |
| 属性 | 说明 |
| Channels | 数据输入或输出的通道。模拟输入通道的增加和移除需要通过addAnalogInputChannel和removeChannel来完成 |
| IsContinuous | 这是一个bool状态量，用来决定是否持续进行数据获取。在本例中，App持续获取采集卡数据，直到我们通过交互按钮暂停或停止。因此，我们需要将这个属性设置为true。 |
| NotifyWhenDataAvailableExceeds | 当获取的数据量超过这个属性定义的值时，将出发DataAvailable事件。通过调整这个属性，我们可以控制App接收多少数据量后批量处理一次数据 |
| Rate | 在本例中，就是数据采集卡的采样率。我们可以通过改变该属性改变采样率。需要注意采样率受实际硬件限制，有可能并不和我们通过程序设定的一致，特别是在采样率设定得很高或不是常规数值的情况下。 |

注意：有一些属性，如Tab. 3‑3中的Channels，无法直接进行赋值，而是在其他属性配置好后就已经确定，或是由系统硬件决定。这类属性是只读（read-only）的，在阅读帮助文档时请读者注意区分。

在上述属性中，我们注意到session的运行涉及一个特殊的事件：DataAvailable。我们进一步查看该词条的帮助文档，发现它将我们引导到了addlistener这一命令：

lh = addlistener(session, 'DataAvailable', callbackfct);

lh = addlistener(session, 'DataAvailable', @(src, event) expr);

事实上，addlistener创建了一个用于侦测特定事件的对象，在本例中这个事件就是DataAvailable。顾名思义，该事件触发并被侦测到后，由采集卡获取的一批新的数据将可供处理，具体的处理方式则通过回调函数callbackfct（简单的回调函数可以直接使用句柄@(src,event) expr）定义。而我们关心的数据则被存储在事件对象event中，包括Data, TimeStamps, TriggerTime。在帮助文档中可以找到它们的具体说明。

至此，我们已经掌握了session的配置方法。回到【Acquire Data in the Background】词条下，此时我们应当能够很好地理解示例代码了。一个更完整的例子在DataAvailable词条下可以找到，我们仔细查看这个例子。其中，对NI采集卡中的数据获取，该例子采用了以下代码实现：

s = daq.createSession('ni');

addAnalogInputChannel(s,'cDAQ1Mod1','ai0','Voltage');

s.DurationInSeconds = 5;

lh = addlistener(s,'DataAvailable',@plotData);

startBackground(s);

wait(s)

delete(lh)

回调函数plotData的实现如下：

function plotData(src, event)

plot(event.TimeStamps, event.Data)

end

具体地，这个示例创建了一个名为s的data acquisition session对象，采集设备为NI的cDAQ1Mod1，使用的通道为ai0，其物理单位用伏特表示。s在启动后将运行5秒，每获取一定的数据（s.NotifyWhenDataAvailableExceeds），将触发DataAvailable事件，调用plotData函数对新的数据进行一次绘制。

完成上述配置后，执行startBackground命令使s在后台运行，并等待直到其运行结束。然后删除侦听对象lh以释放其内存。

现在我们可以将我们需要实现的具体功能和该示例程序中的内容对应起来，结合上文中对data acquisition session属性的学习，修改相应的配置。这样就完成了Hall传感器数据接收的代码编写。我们先创建一个脚本文件（\*.m或者\*.mlx）对代码进行测试。在本例中，我们使用的是NI的数据采集卡，名称为Dev1：

% Create session and listener

SampleSession = daq.createSession('ni'); % create DAQ

addAnalogInputChannel(SampleSession, 'Dev1', 'ai0', 'Voltage');

SampleSession.Rate = 1e3; % Sample rate

SampleSession.IsContinuous = true; % Data will be sampled continuously

SampleSession.NotifyWhenDataAvailableExceeds = 1e3; % Define data length for triggering 'DataAvailable' event

SessionListener = addlistener(SampleSession, 'DataAvailable', @ProcessData);

提示：如果不清楚采集卡的名称，可以在设备管理器或者MATLAB的工具箱【】中查看。或者使用预设的名称尝试运行addAnalogInputChannel命令，若不存在该名称的设备，MATLAB将弹出错误，同时给出可用的设备名称。

回调函数ProcessData先仿照MATLAB中的例子给出定义，确认成功获取采集卡数据后再进一步修改：

function ProcessData(src, event)

plot(event.TimeStamps, event.Data);

end

运行脚本文件，在MATLAB的工作区生成了SampleSession和SesssionListener。我们可以在命令行输入对象名称确认其属性：

这就是我们需要的配置。如果NI采集卡已经和计算机正确连接，我们可以直接运行SampleSession查看结果：

startBackground(SampleSession); % Start DAQ session

MATLAB将以1 Hz的刷新频率显示Dev1的ai0通道采集的数据，每次显示的数据长度为1000，如【fig#】所示。

【fig#】

注意：确认测试结果没有问题后，切勿忘记手动停止SampleSession。如果不需要在脚本中进行其他测试，还需要进一步删除SampleSession和SesssionListener，否则当我们在App Designer中编写类似代码并运行时，由于之前的data acquisition session对象尚未被删除，硬件设备被占用，这将导致运行失败。

上述代码可以被很容易地改写到App Designer中。回顾2.2节的内容，在App Designer中代码在格式上的修改主要包括两处：对象SampleSession和SessionListener应当是归属于App的成员，访问它们需要加上前缀【app.】；用于处理事件DataAvailable的回调函数ProcessData本身并不是App的回调函数，可以将它作为一个App的成员，即辅助函数的形式实现，此时它的输入参数将会增加一个【app】。完成上述修改后的代码如下：

% Create session and listener

app.SampleSession = daq.createSession('ni'); % create DAQ

addAnalogInputChannel(app.SampleSession, 'Dev1', 'ai0', 'Voltage');

app.SampleSession.Rate = 1e3; % Sample rate

app.SampleSession.IsContinuous = true; % Data will be sampled continuously

app.SampleSession.NotifyWhenDataAvailableExceeds = 1e3; % Define data length for triggering 'DataAvailable' event

app.SessionListener = addlistener(app.SampleSession, 'DataAvailable', @app.ProcessData);

我们在辅助函数中完成数据的导入和绘制：

function ProcessData(app, src, event)

plot(event.TimeStamps, event.Data);

end

至此，我们已经解决了如何获取模拟量数据的问题，最后一步是将代码放置到App的代码视图中合适的位置。上述创建对象的工作，我们在运行App后，通常只会执行一次。我们将这段代码放置在【Switch】的回调函数【】中，并且用属性【】记录其创建信息。在关闭并重新打开【Switch】后，我们不需要再次创建SampleSession，只需要改变采样率等参数即可。同理，启动SampleSession的命令startBackground和停止命令stop，也同样在回调函数【Swtich】中实现。

此外，我们还设计了暂停/继续按钮，用户单击此按钮只是简单暂停了采样，并不会修改采样模式、采样率等参数。在该按钮的回调函数【】中我们也要为其添加startBackground命令和stop命令。

最后，考虑到用户在关闭App时，SampleSession需要被删除，以免下次启动App时该对象仍然存在并占用硬件设备，我们在回调函数中【】添加stop和delete命令，释放与SampleSession相关的内存。

完成上述工作后，我们实现了如下代码：

【code】

对磁栅尺编码器信号的获取，我们也遵循相同的步骤来完成代码的编写。在DAT中选择【Counter and Timer Input and Output】并查看。【fig#】展示了【Counter and Timer Input and Output Workflow】中我们所需要的工作流程。考虑到我们只需要获取编码器的输入数据，并不需要产生输出信号，我们查看【Acquire Counter Input Data】中的具体内容。

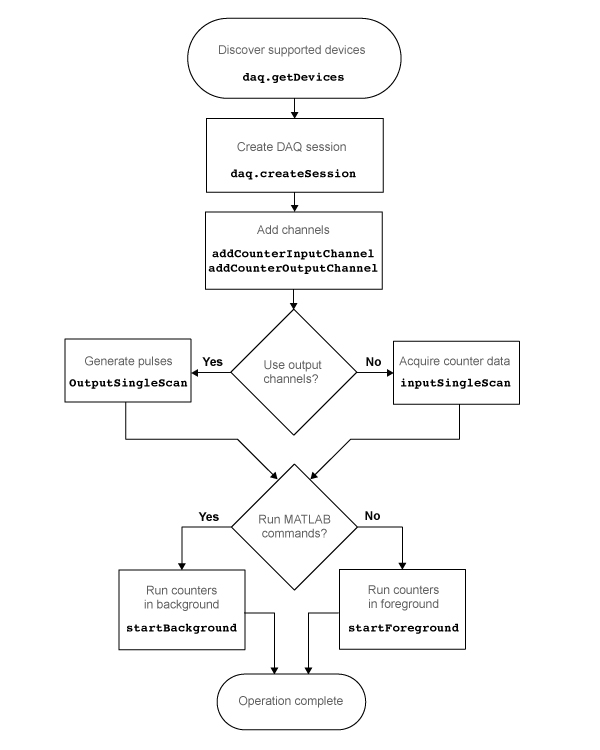


Fig. ‑ 使用DAT实现编码器数据输入和输出的工作流程

在该文档中，并没有介绍后台获取编码器数据的方法。但是通过上文对模拟量数据读取方法的学习，我们应当可以变通地克服这个困难。在文档中，【Acquire Counter Input Data in the Foreground】和我们的需求是最为接近的，其主要的区别在于该例在前台运行，而我们需要让程序在后台运行，所以我们只需要将startForeground命令改为startBackground即可。该例使用了addCounterInputChannel来获取编码器信号，同时也使用addAnalogInputChannel命令添加了一个模拟通道。这是由于NI的编码器需要一个时钟作为基准，而添加模拟通道可以和编码器共享内部时钟。对我们的开发案例，因为需要获取Hall传感器的模拟信号和磁栅尺编码器信号，所以这并不会成为问题。

【Acquire Counter Input Data】中仅仅提供了一个非常简单的例子，至于编码器通道参数的配置仍不明确。【Counter and Timer Input and Output】下有一些更加具体的例子，如【Measure Frequency Using NI Devices】和【Measure Pulse Width Using NI Devices】，读者可以进一步参考。这些例子和我们在实现模拟量读取时查看的例子非常相似，此处不再赘述。

此处我们还需要注意的是编码器数据相较于模拟量数据的不同之处：本开发例中使用的编码器测量的是磁栅尺读数头的位移，但是由编码器直接得到的数据并不是位移，而是和位移相关的脉冲计数值。我们使用的编码器是一种增量型编码器，与位移检测相关的信号一共有A, B, Z三路。【fig#】展示了增量式编码器的原理。不难看出，在忽略编码器的“阶梯状”量化误差的前提下，计数脉冲数和相对位移之间是成正比的。但是究其细节，采用哪一种脉冲计数方法、零位信号脉冲应当如何使用，这些问题都需要我们根据实际情况确定。

【fig#】

在【Counter and Timer Input and Output】下列出了addCounterInputChannel所添加的编码器通道的所有属性，其中就包含了与上述编码器配置相关的属性。读者可以按照上文对模拟量输入部分data acquisition session的属性配置类似的流程完成该项工作。配置好编码器的属性后，我们在一个脚本文件中代码进行测试：

【code】

观察绘图结果，计算编码器读数和实际位移之间的比例系数，并完成编码器数据到位移的转换：

【code】

在脚本文件中测试无误后，我们将这部分代码也改写到App Designer中。这部分内容和模拟量数据获取的处理方法类似，请读者参阅2.2节完成。最后，我们确定编码器代码的放置位置。注意到我们已经在获取模拟量时创建了data acquisition session，为了避免资源的浪费，我们可以将编码器通道的创建代码与模拟量通道的创建代码放在一起执行。当我们使用startBackground命令启动数据获取对象SampleSession时，将同时开始模拟量和编码器数据的采集；停止运行和删除对象的操作也已经完成，我们不需要再重复编写这些内容。至此，我们在App Designer中编写得到的代码如下：

【code】

此时，我们的App已经初具雏形，能够实现信号数据的实时获取和显示了：

【fig#】

注意：MATLAB中的DAT模块也在随版本不断更新。在MATLAB R2020a后采用【】实现上述相同的功能。二者的实现过程是类似的，但代码编写有所不同。读者若使用的是这之后的版本，请参照文档学习使用方法。

利用DAT，我们已经完全实现了NI采集卡数据的获取和显示的功能，通过使用事件和回调函数在后台运行程序代码的思路应当给读者留下了一定的印象。然而，我们目前实现的上述代码仍然存在一个明显的不足：事件回调函数ProcessData似乎承担了太多的工作——它本身应当仅用于处理传感器的数据并将数据记录到App的RecordData属性中。在App的后续开发中，我们还需要根据Hall传感器实时计算位移或展开其他数据分析工作，难道这些内容连同图像绘制都需要在ProcessData中实现吗？这种做法虽然并未违反我们在App草图设计阶段所作的功能性规划，但是会导致ProcessData显得十分臃肿，并使得显示模块的刷新频率和数据采集频率一致而无法单独调整。

但是，若我们不将这些方法在ProcessData中实现，又应当将它们放置在App的什么位置呢？

设置一个简单的循环是否可行？

事实上，若我们在启动SampleSession后进入一个while循环，以设定频率完成显示模块更新等其他内容，将遭遇一个新的问题：退出while循环的条件应当如何设置？请注意，因为MATLAB顺序执行代码的特性，在while循环的过程中，App并不会响应按钮等组件的交互，更不会触发事件执行回调函数。因此，直到while循环主动退出，用户才可能通过单击按钮等方式与App进行交互，否则App将陷入无法操作的“死锁”状态。显然，仅依靠简单的循环实现类似DAT后台运行的功能存在困难：这些代码并不是在后台执行的，无法触发类似单片机等微处理器中的“中断”，毕竟我们根本就没有为程序定义“中断”模式和处理方法。

一个更好的方案是使用一个类似data acquisition session的、封装好的“模块”来解决，即由MATLAB来代为解决后台运行的问题。这启发我们寻找MATLAB中是否已有这样的类：它具备完善的方法，能够在运行时触发中断事件——例如每隔一段固定的时间周期，执行一次中断事件回调函数。如果存在这样的类，那么我们完全可以借助这个类独立实现绘图、位移解算等工作。

MATLAB的定时器类【timer】正好符合我们的要求。接下来我们使用timer来更好地解决传感器数据的实时绘制问题。打开timer类的帮助文档：

【edit】

完成本节的工作后，我们在App Designer中编写的代码如下：

【code】

读者可以本节内容为参考，进一步考虑如下问题：在开发软件时，并不一定总是具备完善的硬件条件支持，例如由于课程资源限制，NI采集卡数量有限而暂时无法使用。在未连接NI采集卡的条件下，本节编写的代码将无法正常运行。为了测试其他模块代码的正确性，有必要另外设计一个模拟采样条件，此时的传感器实时数据由计算机模拟生成的数据代替。对于这一功能需求，应当如何实现？

根据本节介绍的内容，我们可以提出如下解决方案：定义一个timer类对象VirtualSession，以代替SampleSession实现模拟采样的功能。这是对本节介绍内容的一个很好的练习，读者可自行尝试实现。

## 关键功能实现：位移测量结果的解算和显示

本节将对另外一个重要功能进行实现——直线位移平台位移测量结果的解算和显示。位移的解算涉及到物理模型的求解和算法设计，这部分实际上是课程的主要教学内容之一，因此在此处仅作必要的介绍，不深入进行解释。

位移的解算依赖于预先建立的模型。读者可简单认为模型在工程实现上由两部分组成：模型结构和模型参数。前者由物理规律唯一确定，后者则反映了实际对象的具体信息，如机械尺寸、质量、电流等，测量对象发生变化时，模型参数也随之改变。在本项目的App运行的初始阶段，并未掌握所有模型参数，或是模型参数不够准确，因此需要进行标定，即修正模型参数使模型更加符合物理实际。模型参数标定完成后，对数据采集卡的数据输入，App便能够根据已知模型对位移进行求解。

综上所述，位移的解算需要分为以下几个步骤实现：数据采集（预先完成）🡪模型参数标定🡪数据采集（循环）🡪位移解算（循环）。数据采集的工作在3.3节中已经完成，我们只需要再完成参数标定和位移解算两部分的代码实现。模型参数的具体设定取决于我们使用何种模型。例如，若我们采用基于数据的模型（插值法、神经网络等）来计算位移，则存储模型参数的App属性CalibrateData可以定义如下：

【code】

对本开发例采用的测量模型，模型参数的求解本质上是一个非线性方程求解问题。首先，明确要实现的代码功能的输入和输出参数。对位移的求解，输入输出分别为：

输入参数：某个时刻连续采样得到的Hall传感器的模拟量信号（向量），位移零点（标量）。

输出参数：对应时刻直线位移平台动子相对零点的位移（标量）。

我们将这段代码封装为一个辅助工具函数，如下：

function Disp = CalculateDisp(app, Y\_measure)

% Calculate Disp from Y\_measure

end

注意：读者也可以将输入参数设置为一段时间内的采样信号，以增加信息量提高求解精度，或者应用滤波使动态显示的结果更加平滑。此时需要对输出参数的位移对应的时刻做一个明确的定义，例如采集到的数据中第一个点对应的时刻。

对模型参数的标定，我们使用编码器数据作为位移参考，同时可以采样较长时间，以提高参数求解精度。此时的输入输出分别为：

输入参数：一段时间内采样得到的Hall传感器的模拟量信号（矩阵）和磁栅尺的编码器信号（向量），位移零点（标量）。

输出参数：模型参数（多个标量，或者用一个向量包括所有模型参数作为输出）。

明确了输入输出后，我们同样将这段代码封装为一个辅助工具函数，如下：

function Stat = CalibrateModelPara(app, CalibrateData)

% Calibrate model parameters

end

函数内部是具体的代码实现。如果开发者已经有了现成的可调用算法，可以在内部直接调用完成算法的实现。完成这段代码的编写后，在功能层面便可以将辅助函数看作一个黑箱，我们只需关心它的输入输出和完成了什么功能，而不必关系它如何实现这些功能。

本文将不再对位移测量模型和解算算法做更具体的介绍。接下来我们将关注在模型参数标定过程中，读者可能遇到的一些普遍问题。通过解读对这些问题的解决方案的寻求和优化，将有助于读者理解如何将算法从思路转变为代码。

对本文开发例中所要建立的位移测量模型，待测位移是通过Hall传感器测量的磁场强度求得的。假设位移平台仅在一个磁场位移周期内移动，且忽略量化误差和环境扰动因素，则Hall传感器的数值所表征的相对磁场强度和待测位移是一一对应的。但是，我们在标定模型时，即使传感器的位置保持不变，Hall传感器的模拟量信号幅值仍会发生抖动，从而导致我们得到的测量数据中，在同一个磁栅尺参考位置对应的同一个Hall传感器的模拟量信号是一个变化范围，而不是单一的值。这实际上是一种测量不确定度的直观表征。显然，我们无法直接使用测量数据求解模型参数，而是必须通过数据预处理。假设在每个磁栅尺参考位移处得到的单个Hall传感器的数据满足正态分布，那么我们可以通过极大似然估计来给出该处相对磁场强度的最佳估计值。当然，也可以使用较为简单的处理方法，如对该处所有的值求平均作为相对磁场强度的估计结果。

在MATLAB中，我们应当如何实现上述数据的预处理过程？这项工作的核心是去重。一个容易想到的方案是遍历数据，从而为具有不同的磁栅尺参考位移的数据分组。通过编写一个循环，对每个数据点，判断其磁栅尺参考位移是否和之前的数据相同；若存在相同的数据，将新的数据点归到该组中；否则新建一个数据组。完成数据分组后，我们根据每个数据组中的Hall传感器数据估计相对磁场强度，例如取其平均值。这一方案的代码实现不涉及太多的MATLAB方法，编写实现难度较小。

现在我们借助MATLAB中的现有方法实现数据的预处理。通过“去重”这一关键词在MATLAB中检索并进一步由其例子延伸，我们发现【unique】命令和【accumarray】可以很好地解决这一问题。其中，unique的作用是去重，而accumarray的作用是对具有相同下标的数据做批量处理，二者刚好满足我们的上述需求。

【edit】

数据预处理常用的另外一步是插值。在数据去重后，我们得到的数据的磁栅尺参考位移可能并不是在均匀分布的。由于位移平台运动速度太快，传感器可能在部分位置没有进行采样，导致我们最后得到的相对磁场强度-位移信号的位移不是等间隔的。通常，我们更习惯于处理等间隔采样数据，FFT等分析方法也默认数据是等间隔采样得到的。为了数据处理和分析的遍历，我们可以使用插值来将数据“补”成均匀的，同时尽可能保证数据能够反映真实的物理模型。插值的实现方案在任何一门数据处理课程中都会进行详细的介绍。本文将采用MATLAB中常用的一维插值命令【interp1】来完成这项工作。

【edit】

以上两个例子是本节开发工作中几乎必定会遇到的问题。对读者在本节开发内容的代码编写中可能遇到的其他问题，都可以采用类似的从思路到代码的模式来实现和完善。

最后我们还需要将函数的输出提供给显示模块的回调函数，用于相关组件的解算位移的实时显示。这一部分实现过程相对简单，我们只需要在每次完成位移的解算后，将结果赋值给编辑字段的值即可：

【code】

至此，我们完成了位移测量结果的解算和显示部分功能的代码实现。

频谱计算和显示和本节所作的工作类似，主要涉及的内容是使用FFT计算Hall传感器信号的幅值谱。我们可以为存储频谱数据的结构体DataSpectrum做出类似于CalibrateData的定义，用于存储计算好的频谱数据。读者可在理解掌握本节内容的基础上自行完成这部分功能的实现。

## 编写使用说明并发布

需要注意，使用说明的编写应当从用户的角度出发，因此不一定遵照设计思路来阐述App的功能。本开发项目设计App的使用说明以如下顺序编排内容：

【】

由于该App面向本科生课程的实验教学，可以不需要脱离MATLAB平台运行。考虑到调试和更新的便利性，可以使用MATLAB App来发布。

单击【工具栏】的【共享】，选择【MATLAB App】。App Designer将弹出“打包为App”窗口，如【fig#】。

【fig#】

打包分为三个步骤：

第一步，确认需要打包的文件是否完整。特别需要注意App中是否使用了共享的资源和帮助文件，例如一些外部调用的函数脚本文件等。

第二步，完善App的信息。【fig#】中的一部分信息在创建App时已经进行了填写，此处可以进行确认和修改。除此以外，我们还可以补充开发者联系方式和代码依赖产品等信息。

第三步，打包为安装文件。设置安装文件的输出位置后，单击【打包】，等待App Designer完成打包工作。

在文件资源管理器中切换到打包文件的输出路径下，找到安装文件。双击安装文件，MATLAB将弹出确认安装窗口，如【fig#】。单击【？】，MATLAB将自动完成App的安装。在MATLAB的【APP】栏下找到LDMP，单击运行即可开始使用App。

# 附录

## A MATLAB帮助文件

此处提供了与本文相关的MATLAB帮助文件的索引。读者可以通过点击链接查看。此处未提供的帮助文本请参阅MATLAB的帮助文档，或在Mathworks官方网站搜索最新文档。

App Designer……