

使用MATLAB App Designer开发项目

**——简单教程和示例**



2021.11

# 目录

[目录 1](#_Toc86593080)

[前言 2](#_Toc86593081)

[1 MATLAB App Designer简介 3](#_Toc86593082)

[1.1 一些必须熟悉的名词 3](#_Toc86593083)

[1.2 应用运行流程 4](#_Toc86593084)

[1.3 常用组件 5](#_Toc86593085)

[1.4 一个简单的例子：3分钟学习教程 7](#_Toc86593086)

[2 App Designer使用入门 8](#_Toc86593087)

[2.1 明确开发需求，并转化为具体对象 8](#_Toc86593088)

[2.2 为App定义属性、实现方法 8](#_Toc86593089)

[2.3 遇到不会实现的功能时，如何解决 9](#_Toc86593090)

[2.4 编写使用说明，打包、发布 10](#_Toc86593091)

[3 开发项目：基于Hall传感器阵列和NI采集卡的位移测量App 11](#_Toc86593092)

[3.1 初步设计：模块划分和App创建 11](#_Toc86593093)

[3.2 为App定义方法和属性 12](#_Toc86593094)

[3.3 关键功能实现：NI采集卡数据的获取和显示 14](#_Toc86593095)

[3.4 关键功能实现：位移测量结果的解算和显示 14](#_Toc86593096)

[3.5 编写使用说明并发布 15](#_Toc86593097)

[附录 16](#_Toc86593098)

[A1 MATLAB帮助文件 16](#_Toc86593099)

# 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的学习成本很低，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。希望这篇文档能够在这一点上帮到各位读者！

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:

# 1 MATLAB App Designer简介

## 1.1 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。介绍的内容将以MATLAB R2019b为参考，但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。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】。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。

**【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。

**【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*。

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。

## 1.2 应用运行流程

——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。

## 1.3 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#】

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】**

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】

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【sec#】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】

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】

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】

网格布局是MATLAB【ver#】中新增的组件，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。另外，需要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；低版本的MATLAB将无法识别网格布局，导致App无法正确运行。

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。

## 1.4 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。

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。

STEP2 完成了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。

STEP4 现在开始实现交互本身。

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。

STEP6 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。

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。

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。

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。

# 2 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。

## 2.1 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包含哪些功能？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，初步确定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#】

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。

## 2.2 为App定义属性、实现方法

完成【sec#】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【sec#】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。

## 2.3 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【sec#】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，并对症下药对bug进行修复。断点调试和步进调试是常用的手段，App Designer在调试过程中支持实时显示代码中变量的数值，通过和理论预期进行比对，逐步排查问题所在。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。

## 2.4 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设计工具栏中的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。

# 3 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FFT变换。以下我们将从草图设计开始，逐步实现App的各项功能。

## 3.1 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#】

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、采样数据导出、计算FFT等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。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FFT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。

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#】

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两个绘图组件进行操作；又如App在完成传感器才采样信号的FFT计算后，需要将计算结果绘制出来，因此在执行计算FFT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并完善App的信息：

【fig#】

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：

【fig#】

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，状态按钮类组件的初始状态（Value）：

设置模块中，采样频率的值（Value）的默认值、范围、显示格式：

设置模块中，采样通道的默认选择：

显示模块中，数据显示组件的可编辑性（Editable）：

显示模块中，坐标区绘图的标题、横轴和纵轴表示的物理量和单位：

## 3.2 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：

表3-1 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 | / |
| LDMPUIFigureCloseRequest | 关闭App时执行一次，用于关闭子进程等工作 | / |
| SampleSwitchValueChanged | 单击后，启动/停止采样  采样启动时，App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 |
| ExportButtonPushed | 单击后，导出采样数据 | 指令模块：导出数据 |
| FFTButtonPushed | 单击后，计算传感器数据的FFT并绘制结果 | 指令模块：计算FFT  显示模块：绘制频谱 |
| CalibrateButtonPushed | 单击后，校准位移测量模型 | 指令模块：校准模型 |
| SampleChannelListBoxValueChanged | 选择需要显示的采样通道后，在显示模块中变更为相应的显示内容 | 设置模块：采样通道  显示模块：绘制数据、绘制曲线 |

在App Designer中，按表3-1依次创建各个组件的回调函数。接下来还不明确的是这些回调函数的参数。在确定每个回调函数的参数前，我们可以先完成App属性的定义。

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：

表3-2 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值 | 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 |
| RecordTime | 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 |
| RecordHallSensorData | 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 |
| RecordEncoderData | 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 |
| 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 | DataAcquisition 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 |
| TStart | 接收一次采集卡数据工作的开始时刻 | double类型的scalar | 和TStop一同用于记录App接收采集卡数据并实时处理耗费的时间，用来确定何时更新显示模块各个组件的内容 |
| TStop | 接收一次采集卡数据工作的结束时刻 | double类型的scalar | 同TStart |

需要注意的一点是，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：是单个标量，还是数组，还是矩阵，还是更一般的元胞数组？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。

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，因此在可实施性上不具有困难。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，可参照本文【sec#】的内容解决。

## 3.3 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DAQ）来实现，从而节省了很多软硬件通信上的工作。根据DAQ的说明文档， NI采集卡中的数据获取通常采用以下代码结构实现：

【code】

考虑到我们由实时绘制数据图的需求，【】是合适的选择。模拟量的采集通道数目根据Hall传感器的数量确定，数字量的采集通道仅用于磁栅尺编码器，数量为1。采样频率也在此处设定，其数值从设置模块中用户指定的采样频率中获取。由此，我们完成了采集对象的定义。

App运行时，在用户单击启动采样按钮后，即App即开始接收采集卡的数据。因此这段代码应当被放置在【】中。接收到采集卡数据后，App还需要做一些计算和显示工作，这些内容的代码实现应当放在后面，并且在每次数据接收完毕后都执行一次。

【edit】

建议：在对组件进行细节设计时，从用户的角度出发，考虑如何实现更容易让用户理解其使用方法，以及如何在运行过程中提示用户任务的完成进度。改变按钮上的显示信息、添加表示动态运行的图标、创建进度条等都是经常被采用的方式。

## 3.4 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。由于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教学内容，和本文主题的相关性较低，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🡪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【sec#】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。

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：

【edit】

FFT计算和显示和本节所作的工作类似，读者可以在理解掌握的基础上自行尝试实现。

## 3.5 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。

# 附录

## A1 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。