Microsoft

PowerApps 画布应用编码规范和指南

白皮书

概述： 这份技术白皮书的目标读者是 Microsoft PowerApps 企业应用制作者。它包含了对象、集合和变量的命名规范，以及如何开发一致的、高性能的和易于维护的应用的指南。

作者： Todd Baginski, Pat Dunn

技术贡献者： Mehdi Slaoui Andaloussi, Alex Belikov, Casey Burke, Ian Davis, Brian Dang, Rémi Delarboulas, Aniket J. Gaud, Nick Gill, Audrie Gordon, Erik Orum Hansen, Eric Mckinney, Santhosh Sudhakaran, Hubert Sui, Vanessa Welgemoed, Keith Whatling

中文译者： Changwei Zhang（微软产品技术专家）

内容

介绍

Microsoft PowerApps 来自微软的高生产力应用开发平台。 Microsoft 使用它在 Microsoft Dynamics 365 for Sales， Microsoft Dynamics 365 for Service， Microsoft Dynamics 365 for Field Service， Microsoft Dynamics 365 for Marketing， 和 Microsoft Dynamics 365 for Talent 中构建第一方应用。 企业客户也可使用同样的平台来构建他们自己的业务线应用。 您组织内的个人或团队用户也可以使用它来构建个人或团队生产力应用而不必写很多代码（甚至不用写代码）。

白皮书的目的

这份白皮书的目标读者是负责为企业或者政府设计、构建、测试、部署和维护的企业应用制作者。它由微软 PowerApps 团队， Microsoft IT， 和行业专家共同编写。 当然，企业开发者可以自由的编写自己的规范和实践。 尽管， 我们认为遵守这些准则将在如下这些领域帮助开发人员：

- 简洁性
- 可读性
- 可支持性
- 易于部署和管理
- 性能
- 辅助功能

白皮书涵盖的范围

本书涵盖了截止到 December 2018 的所有特性。 如下主题不包含在本书中：

- PowerApps 应用构建基础。 这份白皮书假设你已经掌握如何构建 PowerApps 应用，同时并非专家级。 对于博客，教程，培训资源和社区支持， 请访问 https：//docs。microsoft。com/en-us/powerapps/index。
- Microsoft Power BI 和其他 Microsoft Power platform 的组件。
- PowerApps 以外的其他代码， 例如 Microsoft Azure App Service 和 函数 App。
- 通用治理和应用程序生命周期管理(ALM)。
- 环境管理。要了解此主题，建议您阅读 Administering a PowerApps enterprise deployment 。

译者注：Common Data Service 已于 2020 年 11 月改名为 Dataverse，部分相关术语名称发生变化，请参考官方文档了解详情。

这是一份持续更新的文档

当 Microsoft Power platform 的功能和行业标准变更时， 白皮书也会跟着更新。

Microsoft 倾听开发者的反馈，并持续革新 Power platform 来帮助用户更好的构建应用。 因此， 当新功能发生改变今天的最佳实践可能会过时。因此要定期的检查最新版本的最佳实践和指南。

感谢所有给这份白皮书贡献出集体经验和指引的人。现在，开始查看这份白皮书吧。

通用命名规范

本章将介绍驼峰命名法和帕斯卡命名法，如果你对他们很熟，可以跳过本段。

驼峰命名法

你应该用驼峰命名法命名控件和变量。驼峰命名法采用首个单词小写，其余单词首字母大写的方式来进行命名。 例如， 一个文本输入框可以被命名为 `txtUserEmailAddress`。

帕斯卡命名法

你应该使用帕斯卡命名法命名数据源。 帕斯卡命名法有时被称为大驼峰命名法。 然和不同于驼峰命名法，帕斯卡命名法所有单词手写字母均大写。例如，一个常见的数据源 Microsoft Office 365 Users 连接器， 在代码中被命名为 `Office365Users`。

对象命名规范

当你在 PowerApps 应用中创建对象时， 使用一致的命名规范命名屏幕， 控件， 和 数据源是很重要的。 这个方法会让你的应用更容易维护， 能够提升可用性， 和 是你的代码更可读，当它们被人引用时。

注意： 当我们在准备这份白皮书的时候，发现很多组织使用不同的命名规范。 一个有经验的开发者会重命名他的控件，只要他们会被脚本引用。 不同的开发者可能会倾向使用不同的前缀来命名控件。

这样做没问题! 这份白皮书中的命名规范只是指导， 和 不同的组织可以制订自己的开发规范。 核心是要保持一致性。

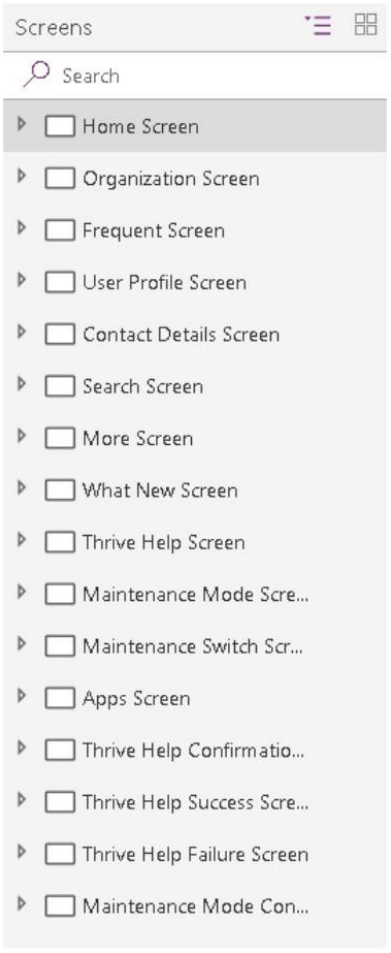
屏幕名称

屏幕应当反应屏幕的用途，以使得复杂应用在 PowerApps Studio 中可以容易的跳转屏幕。

比较少被注意到的一点是屏幕名称会被屏幕阅读器读出来，以使得视觉障碍者可以使用。 **因此，使用直白的语言命名屏幕，并可以包含空格以及不应有缩写是很重要的**。 同时我们推荐使用“Screen”这个词作为名称结尾， 使得当它被读出来来时是易于理解的。

以下是一些好的例子：

- `Home Screen`
- `Thrive Help Screen`



以下是较差的例子：

- Home
- Loader Screen
- EmpProfDetails
- Thrive Help

控件名字

所有画布应用中的控件名字 应该都使用驼峰命名法。 它们应该以三字符的类型开头， 紧接着是控件作用。 这个方法帮助识别控件类型，使得更容易在脚本中使用和 搜索。

一个好的例子： lblUserName

以下是一些通用控件的缩写。

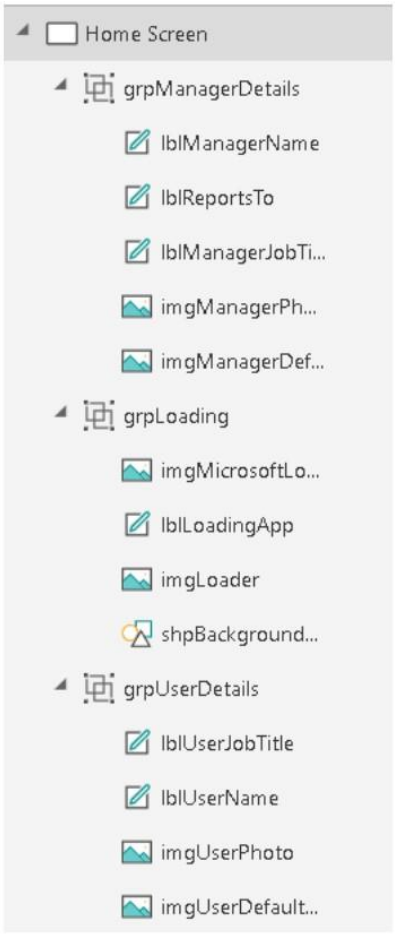
控件名称	缩写
button	btn
camera 控件	cam
canvas	can
card	crd
集合	col
combo box	cmb
dates	dte
drop down	drp
form	frm
gallery	gal
group	grp
header page shape	hdr
html text	htm
icon	ico
image	img
label	lbl
page section shape	sec
shapes (rectangle， circle 等)	shp
table data	tbl
text input	txt
timer	tim

控件名必须是整个应用中独一无二的。 如果控件被多个屏幕重用，控件名必须以屏幕名称缩写作为后缀– 例如， galBottomNavMenuHS， “HS” 代表着“Home Screen”。 这个办法可以让我们更容易的跨屏幕在脚本中引用控件控件。

下面是一些不好的例子：

- zipcode
- Next

如下图所示，如果你保持对控件命名一致， 你的控件在导航视图中看起来会非常清晰，同时你的代码也会很清楚。



数据源名字

当你添加了一个数据源到你的应用，不能在 PowerApps 应用中修改它的名字。 这个名字是从来源连接器或数据实体中继承的。

以下是一些例子：

- **从连接器中继承的名字：** Office 365 Users 连接器的数据源对象会被命名为 `Office365Users`。
- **从连接中推断数据实体的名称：**当你有一个 sharepoint 列表 **Employees**，此时被连接到 PowerApps 中时， 对应的数据源 名称会是 `Employees`。

关于连接器和连接的更多信息， 请查看 Overview of canvas-app connectors for PowerApps 。

标准行为连接器

在那些会暴露出方法的标准行为连接器中，例如 LinkedIn， 你会看到他们产生的数据源名字和暴露出的方法名采用帕斯卡命名法。 例如， LinkedIn 数据源名字是 `LinkedIn` ，以及暴露的方法名为 `ListCompanies`。

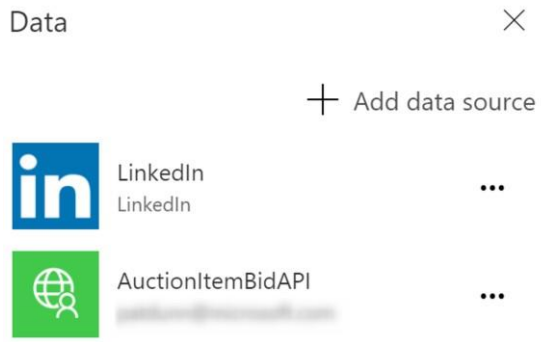
```
ClearCollect(
    colCompanies,
    LinkedIn.ListCompanies()
)
```

自定义连接器

你的环境中任何人都可以创建自定义连接器。 他们用于连接自定义接口(APIs) 例如你们的 IT 开发的第三方 LOB API。 帕斯卡命名法被推荐用于数据源名字 和它们的方法。 只是要注意自定义连接器的名字和 和他们在 PowerApps 中出现的名字可以不同。

例如， 这里有个连接器名字是 **MS Auction Item Bid API**。

但是当你用它创建一个连接时，将他添加到 PowerApps 作为一个数据源，它的名字会变成 **AuctionItemBidAPI**。



为了了解原因，你可以查看 OpenAPI 文件。 在这里你会看到一个 `title` 属性包含了 `Auction Item Bid API` 这个文本。

```
"info": {
  "version": "v1",
  "title": "Auction Item Bid API"
},
```

PowerApps 移除了这个属性的所有空格并将其作为数据源名称。 我们推荐使用帕斯卡命名法修改这个属性的值，例如 `AuctionItemBidAPI` 和 使用它作为自定义连接器的名字。 这种情况下，将不存在名称冲突。 在导入 OpenAPI 文件创建自定义连接器前要修改下它的名字。

注意： 如果你没有选择从现有 OpenAPI 文件创建连接器，而是从空白开始创建，PowerApps 会弹窗要求你输入自定义连接器名称。 这个名字被用做连接器名称和 OpenAPI 定义文件中 `title` 属性的值。再次建议，连接器名字采用帕斯卡命名法，例如 `AuctionItemBidAPI`。

Excel 数据表（DataTables）

PowerApps 使用 Microsoft Excel 中的数据表来从 Excel worksheet 读取数据。 当你使用 Excel 文档作为数据源时，要记住如下要点：

- 为 DataTables 提供描述性的命名。 这个名字会在 PowerApps app 中作为数据源名称来读取数据。
- 每个 Sheet 里使用一个 DataTable。
- DataTable 和 worksheet 要使用一样的名称。
- DataTables 中每一栏标题都要使用描述性的名字。
- 使用帕斯卡命名法。 DataTable 的命名要遵循每个单词首字母大写的命名方式 (例如，`EmployeeLeaveRequests`)。

代码命名规范

在将代码添加到 PowerApps 应用程序时，对变量和集合使用一致的命名规范变得越来越重要。如果正确命名了变量，则应该能够快速分辨出每个变量的类型，目的和作用范围。

我们发现不同组织会使用很多不同的代码命名和对象命名规范。例如，一个团队为其变量使用数据类型前缀（例如 strUserName 表示字符串），而另一个团队为其所有变量添加下划线前缀（_）将它们分组。不同的团队表示全局变量和上下文变量的方式也有所不同。

同样的指导原则也适用于此：为您的团队制定规范，并保持一致的用法。

变量名字

- 变量名称应当能描述其功能。 考虑变量的功能和用途， 然后进行适当的命名。
- 全局变量和上下文变量的前缀应当不同。

放聪明点！ PowerApps 虽然允许上下文变量和全局变量使用相同的名称。但这可能会引起混淆，因为默认情况下，除非使用消歧运算符，否则您的公式将使用上下文变量。请遵循以下约定避免这种情况：

- 上下文变量的前缀使用 `loc`。
- 全局变量的前缀使用 `gbl`。
- 前缀后的名字应当表现变量的功能/用途。 可以使用多个单词命名，同时不必使用任何分隔符号 (例如，空格或下划线)。
- 使用驼峰命名法。变量名的前缀以小写单词开始， 之后的每个单词首字母都应大写 (`lowerUpppperUpper`)。

以下是一些范例：

- **全局变量：** `gblFocusedBorderColor`
- **上下文变量：** `locSuccessMessage`

以下是一些不好的用例：

- `dSub`
- `rstFlds`
- `hideNxtBtn`
- `ttlOppCt`
- `cFV`

- `cQId`

避免使用过短和难以理解的变量名字， 例如 `EID`。应该使用 `EmployeeId` 来替换。

注意： 当你的应用有很多变量时，你只需要在脚本栏输入前缀即可看到变量列表。 如果你遵循了我们的命名指南， 您可以在开发应用程序时轻松地在公式栏上找到它们。最终，这种方法可以加快应用程序的开发速度。

集合名字

- 能够集合的内容。 考虑集合的内容和用途， 再适当地命名。
- 集合名称的前缀应当使用 `col`。
- 前缀后的名称内容应当表明集合的内容/用途。 可以使用多个单词命名，同时不必使用任何分隔符号 (例如，空格或下划线)。
- 使用驼峰命名法。变量名的前缀以小写单词开始， 之后的每个单词首字母都应大写 (例如，`colUpperUpper`)。

以下是一些范例：

- `colMenuItems`
- `colThriveApps`

以下是一些不好的用例：

- `orderscoll`
- `tempCollection`

注意： 当你的应用有很多集合时，你只需要在脚本栏输入前缀即可看到集合列表。 如果你遵循了我们的命名指南， 您可以在开发应用程序时轻松地在公式栏上找到它们。最终，这种方法可以加快应用程序的开发速度。

组织你的代码和对象

使用分组来组织

一个屏幕上所有的控件应当分组来组织，以致于你可以容易的识别它们的用途，也可以跨屏幕移动它们，或者折叠它们来让编辑器画面更简洁。 Gallery， Form， 和 Canvas 控件 虽然已经内部分组了， 但是它们也可以被放进其他的组里以便更好的组织屏幕内容。

你可以使用[增强的 Group 控件](#)，它允许分组嵌套。

文本格式化功能

随着公式复杂度的增加，可读性和可维护性受到影响。很难读取包含多行函数的大型代码块。格式化文本功能添加换行符和缩进以使您的公式更易于阅读。 至于代码注释，多余的空格将应用程序包中删除。因此，发布应用程序之前无需“移除”格式。

保持控件数量的最小化

为了最大程度地减少复杂性，请尝试限制应用程序中的控件数量。例如，不要使用四个图像控件相互重叠并且具有不同的可见属性设置来切换展示的图片，而是使用一个图片控件采用逻辑切换来显示不同的图片。

在最佳的位置放置代码

随着 PowerApps 应用复杂性的增加，在调试应用时查找代码会变得更加困难。一致的模式使这一挑战变得更加容易克服。尽管本节并不详尽，但它提供了一些有关在最佳位置放置代码的指导原则。

作为一般性指导，请尝试将代码尽可能地移至“最高级别”，以便将来查找。一些开发者喜欢将代码放在 OnStart 属性中。只要您了解 OnStart 属性的局限性，以及对应用性能的潜在影响，这样放置是个不错的选择。其他开发者更喜欢将代码放入 OnVisible 属性中，因为它很容易找到，并且只要屏幕可见，代码就可以可靠地运行。

代码封装

任何时刻，都不要试图在一个屏幕中放置多个屏幕的代码逻辑， 只会导致所有的代码依赖一个屏幕。 例如， 一个开发者构建了一份人员查看应用，能够在一个 Gallery 里显示人员的组织架构。当用户点击一个员工的名字时，该应用将会跳转到一个新 屏幕并显示员工的资料信息。 在这个案例中， 该开发者没有将载入员工个人资料的逻辑代码放到 gallery 的 OnSelect 属性。 替代的方式是，该应用仅仅传递了下一个屏幕所需的任意变量作为 Navigate 函数的上下文变量。 **User Profile** 屏幕 自己完成载入用户个人资料的所有动作。

以下是该示例中组织架构 gallery 的 OnSelect 属性中 Navigate 函数的代码。

```
Navigate(
    'User Profile Screen',
    Cover,
    {
        locSelectedEmployeeID: ThisItem.id,
        locSelectedEmployeeName: ThisItem.displayName,
        locSelectedEmployeeJobTitle: ThisItem.jobTitle,
        locProfileFetchComplete: false,
        locDirectReportsFetchComplete: false,
        locMgrHierarchyFetchComplete: false,
        locPeersFetchComplete:false,
        locSelectedTab:"Profile"
    }
)
```

然后，在 **User Profile** 屏幕的 OnVisible 属性中，使用上一个屏幕传入的用户 ID 调用 Office365Users. UserProfileV2 函数以获取用户资料。 后续的代码会使用其他传入的上下文变量。

```
ClearCollect(colUserProfile, Office365Users.UserProfileV2(locSelectedEmployeeID))
```

注意： 以上示例使用 ThisItem 的属性值作为上下文变量而不是让下一个屏幕 直接引用前一个屏幕上 gallery 的 Selected 属性的值。 此处特意使用了该方法， 因为该应用有多种方式从其他屏幕跳转到 **User Profile** 屏幕。 通过传参屏幕封装了该屏幕，使得它可以在此应用和 其它应用中被重用。

OnStart 属性

一般来讲，应该尝试限制 OnStart 属性中的代码量， 因为这里很难调试。要在那里调试代码，必须先保存，关闭应用，然后在 PowerApps Studio 中重新打开 PowerApps 应用程序以使代码再次运行。你无法在该属性中创建上下文变量。 考虑到它是 Application. OnStart，此处脚本仅能在任何屏幕显示前运行一次。

以下是推荐在 OnStart 属性中的行为：

- **屏幕路由：** 不像 OnVisible 属性， 你可以在 OnStart 属性中使用 Navigate 函数。 因此， 它可以被用作切换路由的地方。 例如， 你可以传入一个名叫 mode 的参数来决定显示哪个屏幕。

```
Navigate(
    Switch(
        Param("mode"),
        "new",
        'New Order Screen',
        "edit",
        'Edit Order Screen',
        "history",
        'Order History Screen',
        'Dashboard Screen'
    ),
    ScreenTransition.None
)
```

- **模拟或调试权限：** 你可以在 OnStart 属性创建代码来检查当前用户是否在一个邮箱列表中， 如果用户在列表中， 开启调试模式以显示隐藏的屏幕和控件。

```
Set(
    gblAllowDebug,
    If(
        User().Email in [
            "bob@contoso.com",
            "susan@contoso.com",
            "rajesh@contoso.com"
        ],
        true,
        false
    )
)
```

注意： 你还可以检查 Azure Active Directory（AAD）组成员身份，以将安全设置应用于您的应用程序。。

- **静态全局变量：** 在 OnStart 属性中创建错误消息的集合或设置全局样式变量， 例如控件颜色和边框宽度。 其他实现方式，请看白皮书后续的创建隐藏的配置屏幕部分。
- **“仅一次运行” 的代码：** 根据定义，OnStart 属性中的代码仅在应用启动中并在第一个屏幕显示前运行一次。此外， OnVisible 属性中的代码在屏幕每次显示时都会运行一次。因此， 如果你需要代码仅运行一次， 考虑把它放到 OnStart 属性。
- **快速执行的代码：** 关于 OnStart 属性的特定内容，请查看后续的性能优化部分。

关于 OnStart 和 OnVisible 属性的更多信息， 请看 Todd Baginski 的 PowerApps OnStart 和 OnVisible 开发技巧 视频。

OnVisible 属性

OnVisible 属性 用于放置每次用户跳到屏幕时都会执行的代码。 该属性放置代码要小心。 如果可以， 避免在 PowerApps 应用的第一个屏幕的 OnVisible 属性中放置逻辑代码。 取而代之的是， 尝试在控件属性中放置内联代码。

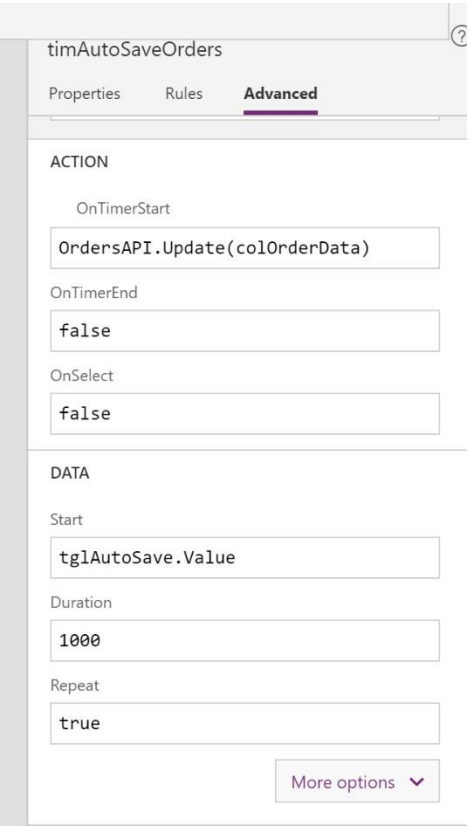
`OnVisible` 属性是适合设置全局或上下文变量的地方。 然而，对于此处调用的代码要小心。 快速执行的调用， 例如调用 `Office365Users.Profile` 函数或设置控件会重用的颜色变量， 是可接受的。 然而， 避免放置复杂逻辑和长时间运行的代码。

关于和 `OnVisible` 属性有关的性能问题的更多信息， 请看高代价的调用部分。

OnTimerStart 属性

定时器展示了执行基于事件的代码的有趣可能。 典型的来讲， 开发者会隐藏 timer 控件，同时绑定定时器的 `Start` 属性到变量或者控件状态。

例如， 如果你有一个表单允许用户切换自动保存功能的开启和关闭，你可以创建一个名为 `tglAutoSave` 开关切换控件。 屏幕上有一个定时器，它的 `Start` 属性设置为 `tglAutoSave.Value`， 同时 `OnTimerStart` 属性中的代码可以保存数据。



`OnTimerStart` 属性中， 你也可以使用 `ClearCollect` 函数定时刷新数据。

`OnTimerStart` 属性也支持 `Navigate` 函数。 你可以使用它来跳转到特定的屏幕，当满足特定条件时。 例如，一个加载屏幕设置了一个上下文变量和定时器，当所有的数据载入完成时 ，会跳转到显示数据的屏幕。此外， 你也可以使用该属性跳转到一个“会话超时”屏幕，当经过一段时间还无法载入完成数据时。

这种模式带有两个注意事项：

- 当你在 PowerApps Studio 中编辑应用时，定时器不会触发。 即使 `AutoStart` 设为 `true`，或者 `Start` 属性属性的值为 `true`， `OnTimerStart` 属性的代码也不会触发。 然而， 当你切换到预览模式(F5)时，定时器将会触发。

- 在 Navigate 函数触发前， 可能会有足够的延迟时间来运行屏幕上的其他代码。

  例如， 你在一个加载屏幕上有一个定时器控件。 你设置了控件的 Start 属性为一个 bool 类型的上下文变量一个 bool 类型的上下文变量 locRedirect， 同时你在 OnTimerStart 属性中放置了如下代码。

```
If(
    locIsError,
    Navigate(
        'Error Screen',
        None,
        {locStatusMessage: locStatusMessage}
    ),
    Navigate(
        'Confirmation Screen',
        None,
        {locStatusMessage: locStatusMessage}
    )
)
```

  加载屏幕的 OnVisible 属性接受一个员工 ID 的参数，如果这个 ID 不是数字的话（因为员工 ID 参数错误的条件是非数字），会设置 locRedirect 的值为 false。

```
//Get the employee ID from PeopleData
UpdateContext({locUserID: Text(PeopleData.PersonnelNumber)});
If(
    !IsNumeric(PeopleData.PersonnelNumber),
    //Couldn't get personnel number from basic profile call. error out
    UpdateContext(
        {
            locIsError: true,
            locStatusMessage: "Unable to retrieve user information",
            locRedirect: true //this will cause our redirect timer to fire
        }
    )
);
```

  当 locRedirect 值为 true 时， 定时器控件的 OnStart 属性的代码会被执行， 但此时存在微小的延时，在此期间 OnVisible 属性的代码会继续执行。 因此， 在后续的代码中要加一个额外的错误检查。

```
//there is a small delay when a timer fires. Change the status message only if there is no error
If(
    !locIsError,
    UpdateContext({locStatusMessage: "Authorizing User..."})
);
//Get token
UpdateContext(
    {
        locSuccessFactorToken: LDService.GetSfToken(
            {
                scope: {
```

OnSelect 属性

控件的 OnSelect 属性每当被选中时都会被调用。 控件可以通过用户交互被选中， 例如按钮的点击或文本输入控件的选中。 此处执行的代码可以验证表单数据以及显示验证消息或者提示文本，或者它可以读取表单数据并写入到你的数据源。

注意：避免在 OnSelect 属性中放置长时间运行的代码， 因为长时间运行的代码可能让人以为应用已停止响应。 更多信息， 请看性能优化 和 高代价的调用部分。 还可以考虑使用加载指示器或状态消息以帮助防止用户感觉到应用缓慢。

当使用 Select 函数选中控件时，控件 OnSelect 属性里面放置的代码会被执行。 一个有用的模式是使用一个加载屏幕作为应用的初始屏幕。 label 控件可以用来显示消息，例如“加载应用数据中。” 这个 label 控件的 OnSelect 属性 可以调用你的数据源和初始化变量， 然后跳转到应用的主屏幕。 通过在应用的 OnStart 属性里使用 Select 函数来选中 label 控件触发它。

尽管初始化代码可以放置在 OnStart 或 OnVisible 属性中， 但采用这个模式有以下的好处：

- OnVisible 中代码不允许用于跳转屏幕。 因此， 你必须添加跳转代码到一个控件， 例如定时器。
- OnStart 中代码会增加应用加载的时间，或者如果启用非阻塞式 **OnStart** 规则（**Use non-blocking OnStart rule**），可能会导致以外的结果。例如一些必须在屏幕加载前要初始化完成的变量或者同步操作不适用非阻塞式加载。
- 当 label 控件通过代码调用方式被选中， 视力障碍的用户将在加载数据时听到屏幕阅读器向他们阅读的标签控件的文本内容。 这个模式提供了一个很好的体验，当用户使用屏幕阅读器时，用户就可以听到“Loading Screen，” “加载数据中，” 或 “Home Screen。” 等语音提示。

注意：如果你在一个屏幕 OnVisible 属性里使用 Select 函数选中一个控件， 同时该控件使用 Navigate 函数跳转到其他屏幕， 那么你可能没办法编辑该屏幕。 为了避免这种情形，可以在隐藏的设置屏幕上使用一个 toggle 开关控件来控制该屏幕是否立即跳转的行为。 可以通过在该屏幕上的控件 OnSelect 属性代码里进行跳转前检查开关的值来判断是否立即跳转。

其他代码组织的建议

- 不要在 If 语句初始条件后，通过显式调用 If 函数嵌套多层逻辑。

```
If(
    One = 1,
        UpdateContext({Nothing: false}),
    If(One = 2,
        UpdateContext({Nothing: true}),
    If(One = 3,
        UpdateContext({All: true})
)))
```

- 为了编写多级逻辑判断，直接在 If 函数后面指定逻辑代码即可， 不需要显式调用 If 函数。

```
If(One = 1,
    UpdateContext({Nothing: false}),
  One = 2,
    UpdateContext({Nothing: true}),
  One = 3,
    UpdateContext({All: true})
)
```

- 尽可能避免使用冗长的表达式。
- 要手动设置代码格式，请遵循以下准则：
  - 每个分号都应当换行。

```
ClearCollect(AwesomeCollection, {AwesomeStuff: "Awesome"});
UpdateContext({TemplatesGood: true});
Set(GlobalVariable, "On")
```

  - 对于较长的单行公式，请尝试在合理的位置换行：可以在括号，逗号，和冒号之前和之后换行。

一般编码准则

点击目标

如果当一组控件被点击时，应当执行指定的动作，那么你可以使用三种方法来实现：

- 最简单的方法是把控件加入组合，将点击事件关联到该分组的 `OnSelect` 属性。
- 将逻辑代码放入到其中一个控件上（最重要的控件），然后在该组其余控件的 `OnSelect` 属性中放入代码选中有逻辑代码的控件-`Select(`*有逻辑代码的控件*`)` 。 和第一个方法一样， 没有要求额外的控件，在编辑器选中该控件也容易。
- 在这组控件的顶部放置一个透明的矩形控件，然后使用这个矩形控件的 `OnSelect` 属性放置逻辑代码。

我们推荐第三种方式，因为采取该方法，如果这组控件内部有控件发生变化也不会对逻辑代码造成影响。 这种方法还使开发者在可点击区域的形状方面更具灵活性。 尽管很难直接在屏幕上选择矩形内的控件，但是您可以在编辑器左侧的屏幕窗格中单独选择它们。

有关此方法的更多信息，请参见文章 HOW TO： Use Transparent Rectangles Effectively In a PowerApp 作者为 Todd Baginski。

变量和集合

上下文变量

要限制上下文变量的使用。 仅在绝对必要时尝试使用它们。

要知道合适应该使用上下文变量 vs 全局变量。当需要在所有屏幕上都可用时，请使用全局变量。如果要将变量的作用范围限制为单个屏幕，请使用上下文变量。

当全局变量更合适（并且更容易调试）时，避免在屏幕之间传递上下文变量。

尽量在一个 UpdateContext 函数调用中更新所有必要的上下文变量。这样，您可以使代码更高效并且更易于阅读。。

例如， 使用如下代码一次更新多个上下文变量。

```
UpdateContext({All: true, Nothing: false, One: 1, Two: "two"})
```

不要分开更新上下文变量。

```
UpdateContext({All: true});
UpdateContext({Nothing: false});
UpdateContext({One: 1});
UpdateContext({Two: "two"})
```

全局变量

当可以只使用一个变量时，请勿使用多个变量。这是多个变量的示例。

```
Set(SelectedMeetingId,First(Filter(AllFutureMeetings,isCurrent = true)).Id);
Set(SelectedMeetingName,First(Filter(AllFutureMeetings,isCurrent = true)).Subject);
Set(SelectedMeetingStartTime,First(Filter(AllFutureMeetings,isCurrent = true)).Start);
Set(SelectedMeetingEndTime,First(Filter(AllFutureMeetings,isCurrent = true)).End);
Set(SelectedMeetingHours,DateDiff(SelectedMeetingStartTime,SelectedMeetingEndTime,Hours));
```

相反，您可以只使用一个变量，如下所示。

```
Set(SelectedMeeting, ThisItem);

SelectedMeeting.Id;
SelectedMeeting.Subject;
SelectedMeeting.Start;
SelectedMeeting.End;
SelectedMeeting.Start;
DateDiff(SelectedMeeting.Start, SelectedMeeting.End, Hours)
```

集合

要限制集合的使用。 仅在绝对必要时尝试使用它们。

使用 `ClearCollect` 函数替代 `Clear;Collect` 两次函数调用。

```
//Use this pattern
ClearCollect( colErrors, { Text: gblErrorText, Code: gblErrorCode } );

//Not this pattern
Clear(colErrors);
Collect( colErrors, { Text: gblErrorText, Code: gblErrorCode } )
```

为了计算一个本地集合的记录条数， 使用 `CountIf` 替代 `Count(Filter())`。

嵌套

避免使用不必要的 DataCard 和画布，尤其是当它们有嵌套的 Gallery 时。

也要尽量避免其他的嵌套操作， 例如 `ForAll` 函数。

```
ClearCollect(FollowUpMeetingAttendees,ForAll(ForAll(Distinct(AttendeesList,EmailAddress.Address),LookUp(Attende
```

注意： 该文档的先前版本指出，嵌套的 Gallery 会被弃用。这是错误的，我们对此错误表示歉意。

性能优化

OnStart 代码

OnStart 属性对于那些只需要调用一次进行应用初始化的代码是至关重要的。同时将数据初始化调用也放入此属性的想法可能很诱人。 然而， 当 OnStart 代码正在运行时，用户会一直看到应用加载画面，和“Getting your data” 的消息提示，因此用户感知到的应用加载时间会更长。

为了获得更好的用户体验，我们建议您在主屏幕上为数据内容显示占位符来等待数据加载完成，例如双破折号（--）。 当数据加载完成时，就填充为正确的数据内容。这样，用户可以开始在主屏幕上阅读内容或与不依赖数据的控件进行交互。例如，用户可以打开“关于屏幕”。

Concurrent 函数

PowerApps 在模块中从上到下顺序运行数据源调用。如果您有多个调用，则此线性执行会对应用程序性能产生负面影响。此问题的一种解决方法是使用计时器控件来同时触发数据调用。但是，这种方法 难以维护和调试，尤其是当某些计时器依赖于其他计时器时。

Concurrent 函数 消除了使用计时器控件来同时执行多个数据调用的需要。下面的代码片段替换了以前驻留在计时器控件的 OnTimerStart 属性中的几个 API 调用。此方法易于维护。

```
    )
);
Navigate('Home Screen',Fade);

Concurrent(
    //Stores Manager details in CurrentManagerHierarchy collection
    ClearCollect(
        colCurrentManagerHierarchy,
        PeopleApi.PeopleGetMyManagerHierarchy({includePhoto: false})
    );
    //Stores peers of the Manager in CurrentPeers collection
    ClearCollect(
        colCurrentPeers,
        PeopleApi.PeopleGetUserDirectReports(
            Last(colCurrentManagerHierarchy).UserPrincipalName,
            {includePhoto: false}
        )
    );
    //Removes the manager from the CurrentPeers collection
    Remove(
        colCurrentPeers,
        Filter(
            colCurrentPeers,
            UserPrincipalName = locCurrentUser
        )
    ),
    //Collects direct reports in CurrentDirectReports collection
```

Format text Remove formatting

为了触发这些调用，您可以将其置于屏幕的 OnVisible 属性中。 或者，如果该方法变得太混乱，则可以将这些调用放在计时器控件中，并在屏幕的 OnVisible 属性或隐藏控件的 OnSelect 属性中设置计时器的 Start 属性引用的变量来启动定时器开始执行。 您还可以将计时器与其他控件结合使用，以在运行 OnVisible 属性中的代码时显示加载消息。 这是一种让用户知道应用正在执行操作的好方法。有关更多信息，请参见 在最佳的位置放置代码 部分。

注意： 为了使代码更易于维护，您可能更喜欢使用 OnVisible 属性。但是，如果使用 OnVisible，则将无法直接使用 Navigate 函数。

要查看并发函数如何使用，请参见 Todd Baginski 的视频， HOW TO： Use the Concurrent function To Make Your PowerApps Perform Better。

委托调用 vs 非委托调用

当您调用数据源时，请注意有些函数是可委托的，而其他函数则不是。可委托的函数可以在服务器上进行执行查询，并且性能更好。不可委托的函数要求将数据下载到客户端本地执行查询。此过程比可委托的调用更慢，而且数据压力更大（可能会导致将整表数据加载到本地的极端情况）。

有关更多信息，请参见 Understand Delegation in a Canvas App 。

使用本地集合

对于较小的数据集，尤其是在需要频繁访问的情况下，请考虑首先将数据集加载到本地集合中。然后可以对这些集合执行函数操作或将控件绑定到这些集合。 如果您频繁进行非委托调用操作，此方法将特别有用。请注意，这将对性能产生初步影响，因为数据必须先被获取并且返回的记录数量存在限制。 有关更多信息，请参阅 Mehdi Slaoui Andaloussi 的精彩博客文章， Performance considerations with PowerApps。

SQL 优化

您的组织可能正在将 Azure SQL 数据库用做您的后端数据源，以利用其丰富的管理功能和互操作性。但是，如果应用架构设计不当将无法应对并发场景，导致您可能不得不增加数据库事务单元（DTU） 大小和成本。例如， Microsoft IT 为内部会议（拥有 1,700 名与会者）构建了 Thrive Conference 应用。后端是一个 100-DTU SQL 数据库实例。在性能测试期间，Microsoft 要求其运营中心的 120 名员工同时打开该应用程序。此时该应用停止了响应。网络跟踪显示 PowerApps 连接对象抛出了 HTTP 500 错误。SQL 日志表明服务器已被占满，并且很多调用正在超时。

由于在会议之前没有时间重写应用，因此 Microsoft IT 将 SQL 数据库规模扩大到了 4000 DTU 满足了并发要求。因此，成本明显高于最初预算的 100 DTU 的服务器。此后，Microsoft IT 使用此处介绍的方法对设计进行了优化。 现在，一台 100 DTU 的服务器足以应付负载，并且 SQL 调用速度大大提高。

针对 SQL 的委托函数

在阅读了上一节中有关委托的概述之后，请参见数据源和支持的委托列表，以了解支持哪些顶级函数以及 Filter 和 Lookup 函数的哪些断言。 这对 PowerApps 应用的性能带来很大的变化，尤其是在移动设备上，因为不必在客户端上下载和查询整个数据集。

使用视图替代表

不必在表之间遍历以获取数据，而是通过暴露视图的方式。只要您已正确索引了表，此方法查询非常快，并且如果您通过使用在服务器上运行的委托函数限制得到结果的数量，它将更快。

通过流使用存储过程来提高性能

对于使用 Microsoft SQL Server 的 PowerApps 应用，最大的性能提升方式就是可以通过 Power Automate（流）调用 SQL 存储过程。 这种方法的另一个好处是可以将数据库设计与 PowerApps 应用分离。因此，您可以在不影响应用的情况下更改基础表结构。正如我们将在后面解释的那样，这种方法也更安全。

要在已经使用 SQL Server 连接器的 PowerApps 应用中使用此方法，必须首先从该应用中完全删除 SQL Server 连接器。然后创建一个新的 SQL Server 连接器，该连接器使用仅限于对数据库存储过程有 EXECUTE 权限进行访问的 SQL 登录账户登录。最后，在 Power Automate 流中，调用存储过程，并从 PowerApps 应用传入参数。

有关如何创建流并将其结果返回给 PowerApps 的详细说明，请参见 Brian Dang 的文章， [Return an array from a SQL Stored Procedure to PowerApps (Split Method)](#)。

这种方法具有如下性能优势：

- 存储过程在服务端会做查询优化。因此， 数据返回更快。
- 委托调用不再成为问题，因为你的存储过程会被优化以仅读取或写入相关数据。
- 你优化的流现在是可重用的组件。因此，对于常见的读写场景，环境中的其他开发者也可以重用它们。

高代价的调用

您的某些数据或 API 调用操作可能会代价很高或很耗时。执行时间长会影响用户感知到的性能。以下是一些实用技巧：

- 不要在跳转页面时进行高代价的调用。应当视图使下一页的加载瞬间完成。然后，在下一页的 OnVisible 属性中在后台执行调用。
- 使用加载消息提示框或进度展示控件（进度条等）让用户知道应用后台执行的动作有进度更新。
- `Concurrent` 函数是使您的调用并行运行的好方法。但是，使用这种方法时，长时间运行的调用会阻止后续代码运行。

这是在 OnSelect 属性中转到下一页的的错误示例。

```
Set(ShowExportConfirmDialog,false);
If(CheckPlanner,
      ForAll(Tasks,
            Planner.CreateTask(SelectedPlanId,Name,{bucketId:SelectedBucketId,dueDateTime:DueTime,assignments:AssignToId})
          )
 );

ClearCollect(Indexes,{Index:-1});

Navigate(ExportConfirm,None);
```

这是一个更好的示例。首先，这是 OnSelect 属性中的代码。

```
Navigate(ExportConfirm,None)
```

同时这是下一页的 OnVisible 属性 中的代码。

```
If(ExportConfirmed,
    Set(Loading, true);

        If(CheckPlanner.Value,
            ForAll(
                Tasks,Planner.CreateTask(
                SelectedPlan.id,Name,
                {
                    bucketId:SelectedBucket.id,
                    dueDateTime:AssnTaskDueDate,
                   assignments:AssignToUser.Id
                }
               )
            )
        );

    Set(Loading, false)
)
```

限制应用包的大小

尽管 PowerApps 在优化应用加载方面做了大量工作，但是您可以采取措施减少应用的占用空间。减小占用空间对于使用较旧设备或带宽较低网络延迟较大区域的用户尤其重要。

评估应用程序中嵌入的媒体资源。如果不使用某些内容，请将其删除。

- 嵌入式图像可能太大。为了代替 PNG 文件，请查看是否可以使用 SVG 图像。但是，请谨慎使用 SVG 图像中的文本，因为所使用的字体必须在客户端上已安装。 您需要显示的文字是在图像上叠加文字标签。 需要显示文本时，一个不错的解决方法是在图像上叠加文本标签控件。
- 评估分辨率是否适合设备屏幕的尺寸。移动应用的分辨率不必与台式机应用的分辨率一样高。通过进行实验，以获取适当的图像质量和大小的平衡点。
- 如果您有未使用的屏幕，请将其删除。请注意不要删除任何仅由应用开发者或管理员使用的隐藏屏幕。
- 评估是否要在一个应用中放入太多工作流。例如，您在同一应用中是否同时拥有管理员屏幕和终端用户屏幕？ 如果是这样，请考虑将它们分解为单独的应用。这种方法还将使多个人可以更轻松地同时处理这些应用，并且当需要更改应用并要求测试通过时，它将限制“爆炸半径”（测试量）。

定期重新发布您的应用

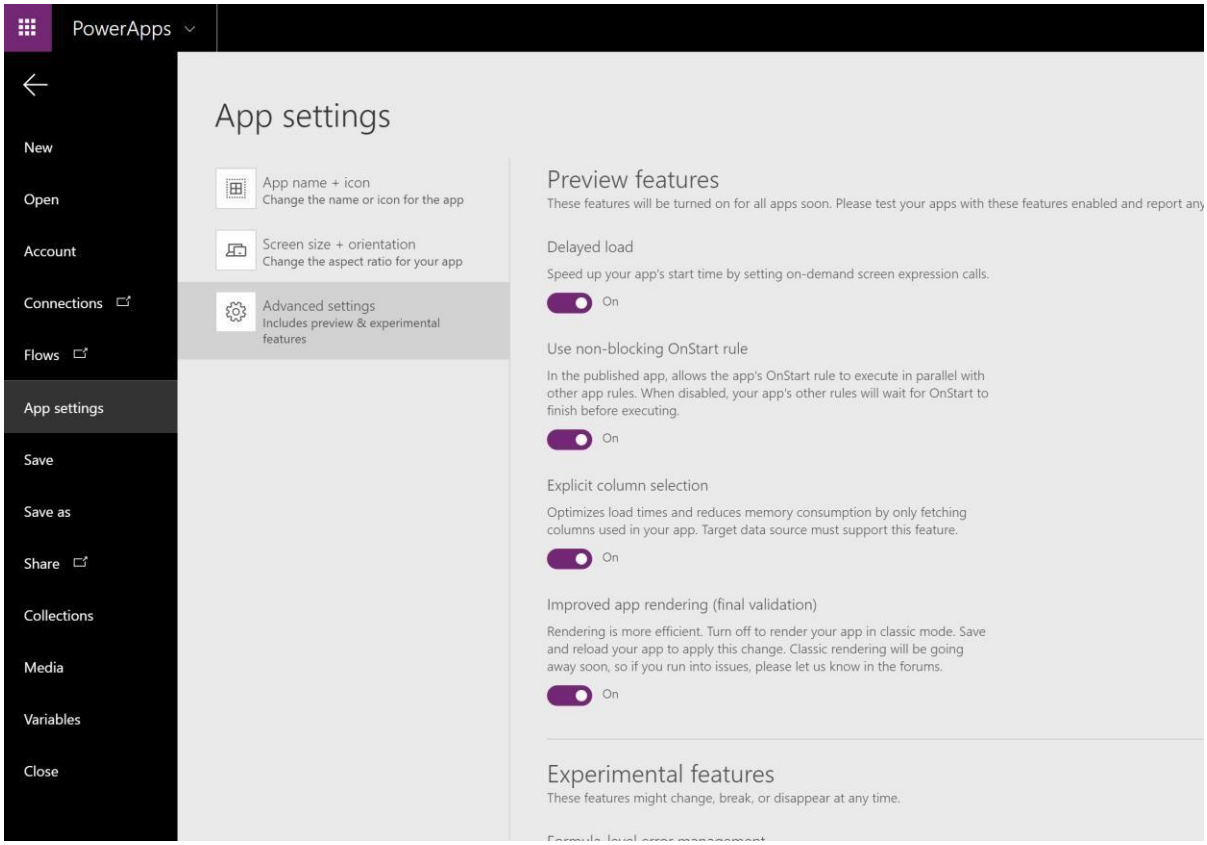
PowerApps 产品团队正在不断优化 Power 平台。有时，为了向后兼容，这些优化将仅适用于使用特定版本或更高版本发布的应用程序。因此，我们建议您定期重新发布应用以利用这些优势优化您的应用。

高级设置

PowerApps 产品组提供了一些预览功能，开发可以选择为其应用开启这些特性。其中一些功能可以大大提高您的应用的性能。例如，**延迟加载功能**可以为您的应用启用延迟加载。在应用初始加载期间，运行时将仅加载显示第一个屏幕所需的屏幕和代码。

译者注：很多预览功能随着时间推移已经成为了正式功能，一切以当前版本的应用设置为准。

使用这些功能需要您自担风险，并且在尝试使用这些功能时，请务必彻底测试您的应用。

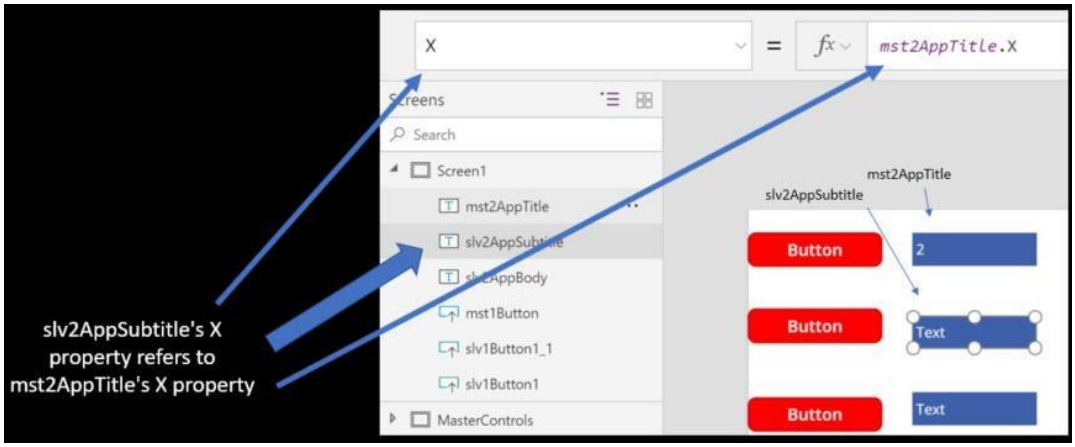


应用设计

使用父子关系设置相对样式

我们推荐您使用一个控件的样式作为其他控件基础样式。We recommend that you use one 控件的 style as the basis for styling other 控件。通常，对控件的颜色，填充，x，y，宽度和高度属性可以

使用相对样式。

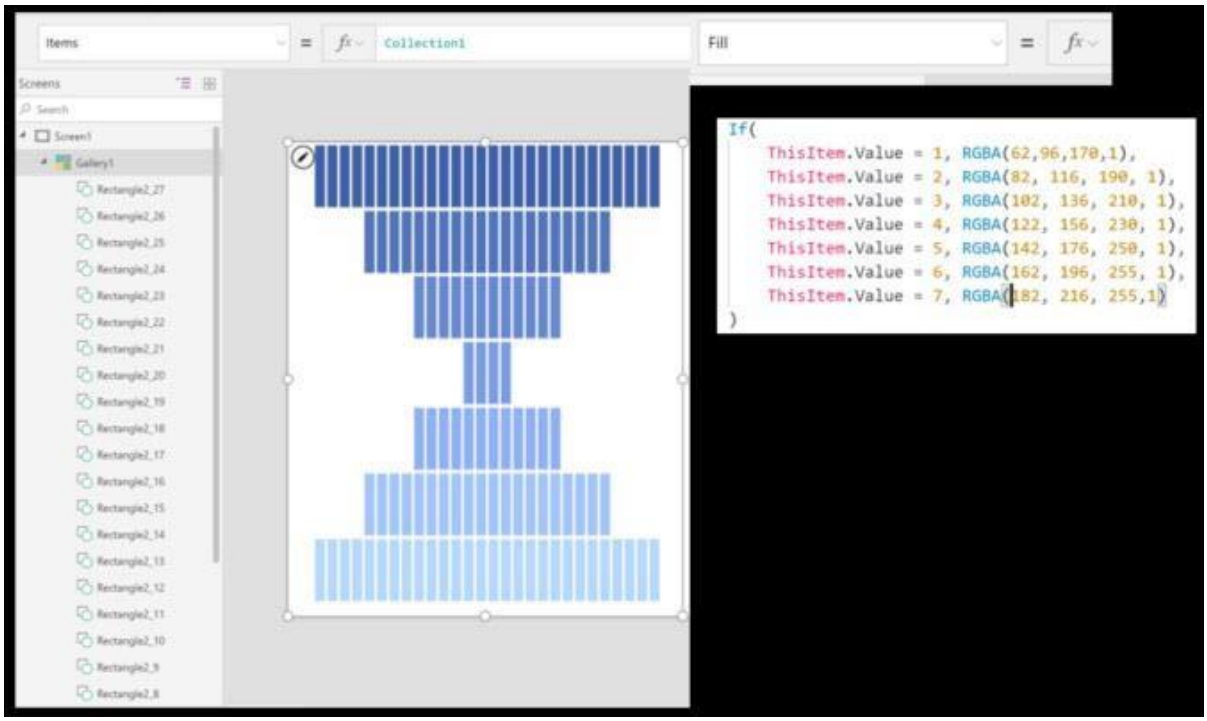


Gallery

对于几乎所有重复和线性的事物，都应使用 Gallery 来展示。

“体力活”（手动放置多个控件）方法最初可能会更快，但是稍后修改将非常耗时。

如果您必须呈现一系列看起来很重复的信息或控件，请始终考虑是否可以通过创建一个内部集合来使用 Gallery 展示内容。



将 Gallery 控件用作视图表单而不是使用 Display Form 控件也很有用。

例如， 你有一个从数据创建的三个屏幕的应用。 有一个“**Users**”数据源，其中包含用户的名字，职务，电话号码。

第一个屏幕– **User List Screen**，上有一个控件名叫 `galUsers`。这个控件会列出所有用户。

第二个屏幕– **User Details Screen**， 上有一个 gallery 控件名叫 `galUserDetails`。该控件的 `Items` 属性被设置为如下内容。

```
Table(
    {Title: "User Name", Value: galUsers.Selected.DisplayName},
    {Title: "Job Title", Value: galUsers.Selected.JobTitle},
    {Title: "Phone Number", Value: galUsers.Selected.PhoneNumber}
)
```

这种方法比尝试以 Display Form 控件修改三个单独的数据卡要快得多。

Forms

将表单用于重复的数据输入字段。

表单还使您可以快速对多个字段进行分组，而不必使用多个文本框。

与单个文本框相比，使用表单要容易得多，因为表单使您可以利用父子关系来实现相对样式。

Full Name
Barbara Sankovic
Approving Manager
Shreya Smith
Status
Pending
Start Date
12/19/2017 4:00 PM
End Date
1/3/2018 4:00 PM
Submit Date
8/23/2017 5:00 PM
Justification
PTO

Common Data Service for Apps

我们建议您使用单个屏幕来处理编辑/插入操作。

如果可能，请使用 CardGallery 控件来处理数据更新，而不是在 Patch 函数中引用单个控件。

当您命名上下文变量时，应指出与它们关联的记录。

多种设备尺寸支持

当您针对手机和平板电脑的布局制作相同的 PowerApps 应用时，首先创建该应用的一个版本，通过测试运行它，并完成它。然后在修改布局和屏幕之前将其转换为另一个版本。 这种方法有助于确保您的表达式，控件，变量，数据源等等都具有相同的名称。 因此这会使得我们开发和维护应

用更简单。 要了解如何将一种尺寸转换为另一种尺寸，请参阅文章 How to convert a PowerApp from one layout to another 。

应用配置数据

您可能会使用 SaveData 和 LoadData 在移动应用程序中存储用户定义的设置。它们提供了方便的数据缓存方式。

注意： SaveData 和 LoadData 仅在 PowerApps 播放器客户端应用内部工作。在设计应用时，请牢记此限制，因为在 Web 浏览器中加载 PowerApps 应用时，这些函数将不起作用。

您的应用可能需要在一处轻松更改应用设置，例如配色方案，其他应用的 URL 或定义是否调试控件在屏幕上可见的设置。因此，部署您的应用程序的人可以快速设置这些值，并且部署期间搞乱代码的风险也较小。可以将这些设置视同为 ASP.NET web.config 文件。 。

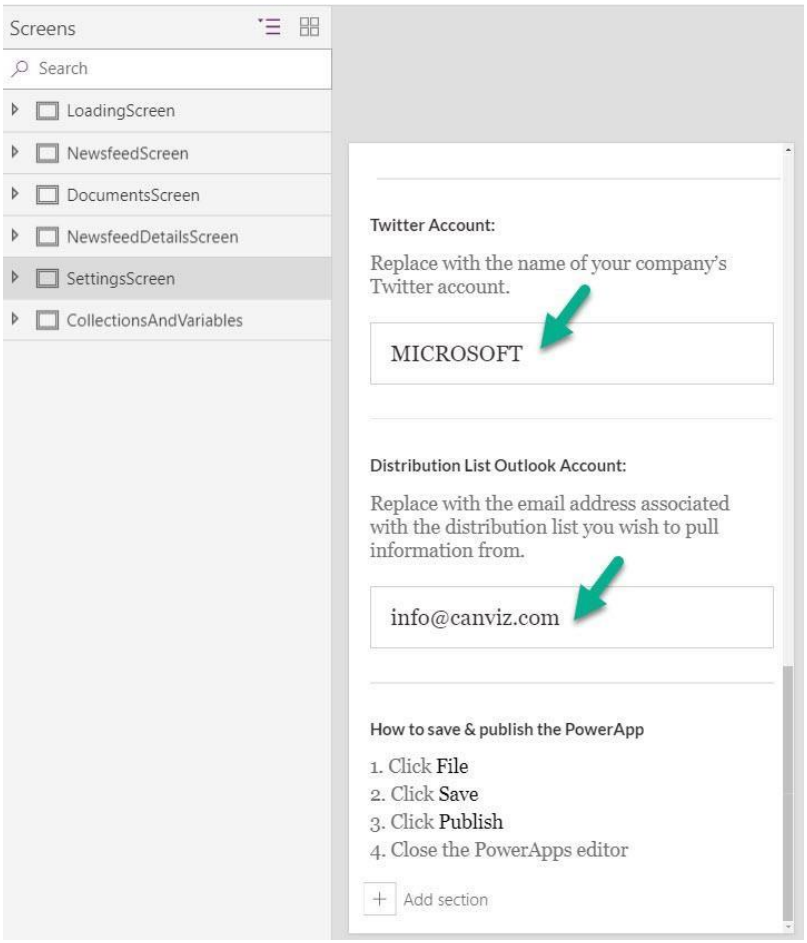
以下是存储应用配置值的几种方法。难度按顺序增加。

创建隐藏的配置屏幕

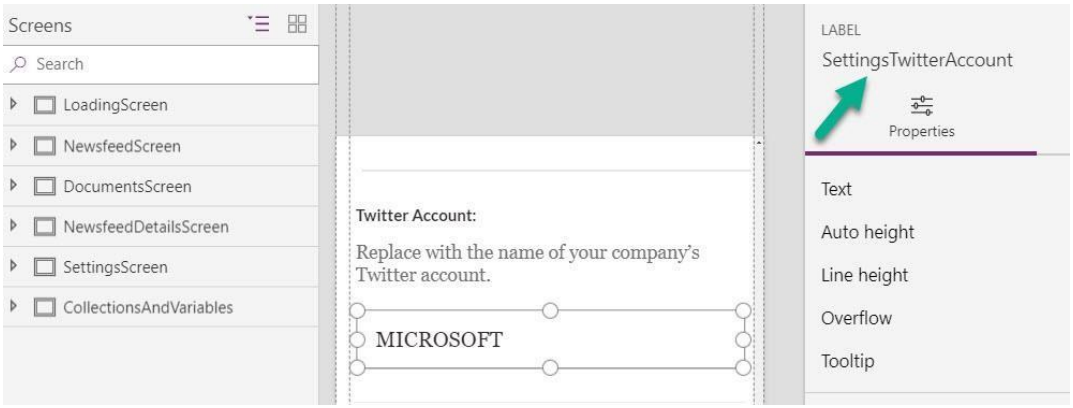
设置应用配置值的一种非常简单的方法就是创建一个隐藏的屏幕，然后将配置值存放到文本输入控件中。这样，您可以在不编辑代码的情况下更改应用设置。若要使用此方法，请按照下列步骤：

1. 确保应用配置屏幕不是应用中的第一个屏幕。按您的屏幕顺序将其放置在其他任何地方。我们建议您将它配置为最后一个屏幕，以便于查找。
2. 确保您的用户不能进入该屏幕。
3. 给自己一个进入屏幕的方法。最简单的方法是仅在编辑应用后才能访问设置屏幕，然后手动进入屏幕。在应用的主屏幕上，您甚至可能放置隐藏按钮，并且仅对应用开发者和管理员可见，并且使他们能够跳转到配置屏幕。您可以通过检查用户的电子邮件地址(例如，检查 User().Email)或他们的 AAD 组成员身份，或使用 PowerApps for Makers 连接器来验证用户是应用开发者或管理员。

以下示例来自 Microsoft PowerApps Company Pulse 示例模板。在这里，您可以看到文本输入控件，该文本输入控件使 PowerApps 管理员可以配置应用程序设置值。



在这里，您可以看到存储 Twitter 帐户设置值的控件的名称。



在这里，您可以看到该值用于从 Twitter 连接器返回 Twitter 帐户的推文。

尽管此方法是更改值的最简单方法，但确实存在一些缺点：

- 您必须重新发布应用以更改值并使它们持久化。
- 由于值始终存在于应用中，因此在导出应用程序之前，您必须创建一个更新这些值的过程，以准备迁移到另一个环境。

使用 Common Data Service 存储配置值

您可以创建一个新的 Common Data Service 表，并在其中存储配置值。由于这些值持久存在于应用外部，因此可以在不重新部署应用的情况下随时对其进行修改。 例如，URL 在预生产和生产环境中可以不同。

尽管此方法是维护配置值的好方法，但也有缺点：

- 和使用文本控件存储配置值的方法不同，这个方法要求调用 CDS 获取配置数据。因此，对性能会有轻微影响，并且如果 CDS 服务不可用（例如，如果用户使用的是移动设备同时失去了网络连接），该应用可能无法正确显示。
- 由于没有缓存，因此每次打开应用时都会产生新的 CDS 调用。
- 无法监控 CDS 调用是否失败。您必须依靠用户来通知应用获取配置值失败。

使用自定义 API

尽管该方法是最困难的，但是 Microsoft IT 已成功使用 Azure 应用服务，将配置值作为 Name/Value 键值对存储在 Azure 表存储中。

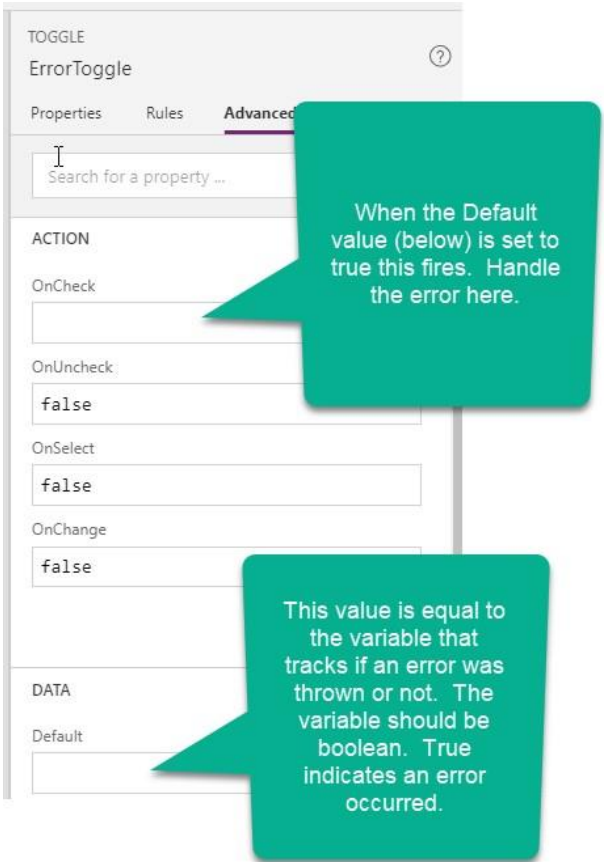
基于 OAuth 保证安全的自定义连接器可获取应用配置值，并且输出缓存（包括值更改时的缓存失效机制）可提高性能。带有警报的 Azure Application Insights 在出现问题时发送通知，并且使得对用户会话进行故障排除更加容易。

错误处理/调试

用 Toggle 控件进行错误处理

在 OnTimerStart 属性 部分， 我们演示了一种使用 Timer 控件处理错误的模式。另一种可用于处理错误的模式涉及 Toggle 控件。

下图显示了这种方法。



通过这种方法，可以将用于验证或错误处理的逻辑封装在单个控件中。Toggle 控件可以推断复杂的条件并得出真值或假值。其他控件可以引用该值以显示/隐藏错误消息，并执行更改字体或边框颜色、使按钮不可用、记录日志到 Application Insights 等动作。如果您使该控件可见并可编辑，则应用开发者可以将错误条件打开和关闭以观察用户界面（UI）的反应。 在开发或调试应用时可以节省时间和精力。

使用画布控件作为调试面板

在开发应用并对其进行测试时，可以使用画布控件创建一个半透明的调试面板，该面板显示在屏幕顶部。然后，该面板可以根据需要具有可编辑的字段、切换开关或其他控件。 这样您就可以在应用处于播放模式时更改变量。

有关分步教学视频，请参阅 Brian Dang 的教程 PowerApps - Best Practices： Debug Panel 。

向应用开发者显示调试控件

您不想向所有用户显示调试控件。因此，您必须手动切换调试控件的可见属性（在 PowerApps Studio 中），或自动切换某些用户的控件可见性。

一种优雅的方法是添加 PowerApps for Makers 连接器。此连接器可以被非开发者的用户将其用于只读调用。 然后调用 GetAppRoleAssignments 函数以确定已登录的用户是否是应用的开发者。

```
Set(gloCurrentUserEmail,User().Email);

ClearCollect(Makers,
    ForAll(PowerAppsforAppMakers.GetAppRoleAssignment("Your App ID Goes Here").value,
    {Email:properties.principal.email,Role:properties.roleName})
);

Set(gloIsMaker,
    And(gloCurrentUserEmail exactin Makers.Email,
    Not(LookUp(Makers,Email=gloCurrentUserEmail).Role="CanView"))
);
```

在此示例中， 你会将调试控件的 `Visible` 属性 绑定到 `gloIsMaker` 变量， 以使得那些控件仅对具有开发者权限的应用用户显示。

这种方法的好处是您无需使用配置表来指定特殊的调试权限。

您还可以通过检查用户的电子邮件地址(例如，检查 User().Email)或他们的 AAD 组成员身份来确保只有应用开发者或管理员才可以看到调试控件。

文档

代码注释

从 2018 年 6 月起， 你可以对代码注释。 当你开发应用时，肯定会频繁编写注释。注释会帮助你日后重新查看时理解代码， 同时下一个接手的开发者也会感谢你。

有两种注释：

- **行注释：** 使用双斜杠(//) 可以添加注释，也可以用于临时注释掉代码。被注释的行将不会被执行
- **块注释：** 任何在/* 和 */ 之间的内容都会被当成注释。它可以注释掉多行，也可以用于临时注释代码。

我们建议应当在格式化文本后再添加注释，**尤其当你的代码时代码块时**。 格式化文本功能会使用如下逻辑处理注释：

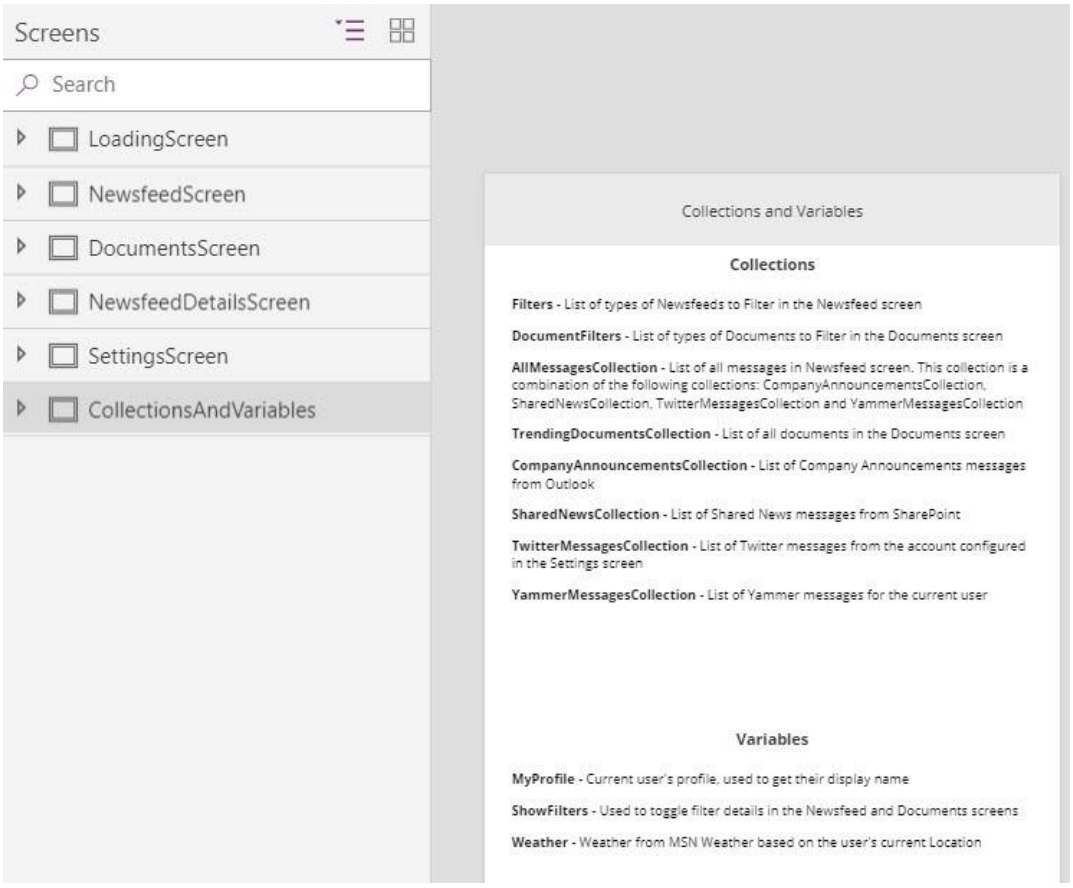
1. 如果一个属性的脚本以块注释符开始， 下一行代码也会被注释。
2. 如果一个属性的脚本以行注释符开始， 下一行代码则不会被注释。 反之， 注释行代码将会被注释掉。
3. 属性的脚本里任意行或块注释内容都会被附加到上一行代码内容之后。

不必担心会添加太多注释或注释太长。当 PowerApps 创建客户端应用包时，所有注释都将被删除。因此，它们不会影响包的大小或减慢应用的下载或加载时间。

文档屏幕

我们推荐创建一些屏幕用于查看 PowerApps 应用中使用的所有的集合和变量。 不要从其他屏幕导航到该屏幕。 这些屏幕仅仅用于编辑模式查看。

此处是 Microsoft PowerApps Company Pulse 模板的例子。


Screens
Search
LoadingScreen
NewsfeedScreen
DocumentsScreen
NewsfeedDetailsScreen
SettingsScreen
CollectionsAndVariables
Collections and Variables
Collections
Filters - List of types of Newsfeeds to Filter in the Newsfeed screen
DocumentFilters - List of types of Documents to Filter in the Documents screen
AllMessagesCollection - List of all messages in Newsfeed screen. This collection is a combination of the following collections: CompanyAnnouncementsCollection, SharedNewsCollection, TwitterMessagesCollection and YammerMessagesCollection
TrendingDocumentsCollection - List of all documents in the Documents screen
CompanyAnnouncementsCollection - List of Company Announcements messages from Outlook
SharedNewsCollection - List of Shared News messages from SharePoint
TwitterMessagesCollection - List of Twitter messages from the account configured in the Settings screen
YammerMessagesCollection - List of Yammer messages for the current user
Variables
MyProfile - Current user's profile, used to get their display name
ShowFilters - Used to toggle filter details in the Newsfeed and Documents screens
Weather - Weather from MSN Weather based on the user's current Location