FileMaker 同步指南

打造专属系统最佳实践

目录

同步简介

借助 FileMaker Go for iPad 或 FileMaker Go for iPhone，您可以随时随地发送 FileMaker 系统中的数据，即便脱离了 FileMaker Pro 上的托管文件，也能进行操作。您还可将字段中收集的数据添加到托管系统中。如果远程用户拥有全天候网络连接和足够的带宽，就可以直接对托管系统进行操作而无需同步。但是，FileMaker Go 远程用户要求不依赖网络连接即可随时随地操作，因此，专为他们设计的解决方案必须支持部署系统到托管系统的同步。

在两个系统之间同步数据可能与将记录从一个文件导入另一个文件一样简单，也可能与按照业务规则逐条更新记录一样复杂。本文讨论了同步过程中既相互独立又相互关联的两个部分 – 记录交换与记录处理。本文重点讲解 FileMaker Go 文件和 FileMaker Pro 托管文件之间的同步，这些概念也适用于 FileMaker Pro 用户所使用的文件之间的同步（比如，同步便携式电脑上部署的 FileMaker 系统与 FileMaker 托管系统）。

本文从探讨同步策略入手：从哪里开始同步？同步过程如何确认托管文件是否可用？同步过程中断时会发生什么状况？

介绍这些基本概念之后，本文将讨论如下问题：网络连接和网络中断、网络中断时如何恢复脚本以及 iOS 设备上的托管文件需要被关闭的时间、方式和原因。接下来介绍记录处理，包括识别待同步记录以及处理记录的时间和地点。此外，本文还涉及了您应当考虑的一些问题，包括确认托管文件和 FileMaker Go 上的部署文件中哪些记录需要被更新，以及以何种方式进行更新最为恰当等等。

下节内容介绍了同步过程的主要步骤。

本文最后一节将详细介绍同步过程。本节介绍了 FileMaker 原生功能的使用，而整个章节的内容基于这样一个原则：尽可能缩短同步过程所需要的网络连接时间。本节还介绍了示例脚本以及对构建表、关系和脚本提出的建议。

本文对几个高级别概念进行讲解，同时为有兴趣构建专属 FileMaker 系统的用户提供详细的技术指导，并为勇于尝试新事物的初学者介绍 FileMaker 的功能和潜力。此外，本文还涉及商务同步解决方案的若干概念。本文假定读者对 FileMaker Pro 的诸多功能已经比较熟悉，并且非常了解关于脚本的知识。

同步概念

同步（动词）：意为协同与合并。

同步有多种含义，适用于不同方案的不同进展阶段。对于那些在一天的开始收集到一组工作指令或任务的用户来说，同步的意思仅仅是在一天的开始将托管文件上分配给他们的记录下载至各自的 FileMaker Go 自定义商务解决方案，并在一天结束时将 FileMaker Go 上对工作指令做出的备注上传至托管文件。假设系统设计合理，这些上传的备注将与托管系统中每个用户的工作指令自动关联。下图描述了这种类型的同步。

图 1：简单类型的同步

如果用户对时效性有更高要求，并且需要对托管解决方案和 FileMaker Go 解决方案同时进行更改时，同步则意味着协同记录数据以确保托管文件和使用 FileMaker Go 部署的文件能够提供相同的信息。下图非常简单，第二步要求同步过程对记录进行逐条确认，并将选定的更新结果应用在其中一个或两个系统中：

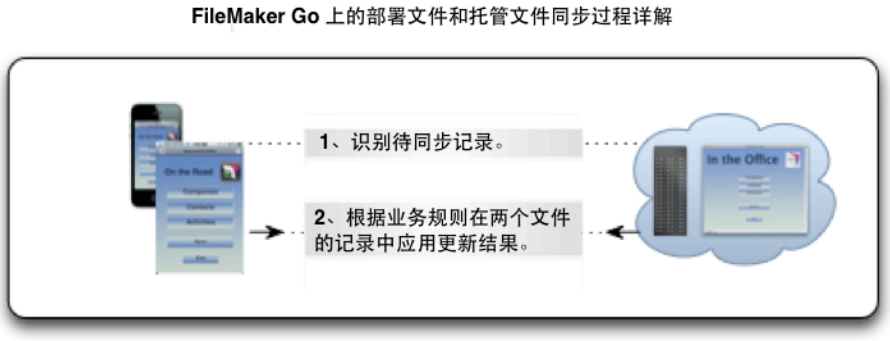

图 2：较复杂类型的同步满足您的业务规则

这些同步方案中包含两个文件：

- **部署文件**：部署在 iPad、iPhone 或 iPod Touch 上可借助 FileMaker Go 运行的 FileMaker 文件。
- **托管文件**：托管在 FileMaker Server 或 FileMaker Pro 上的 FileMaker 文件，可借助 FileMaker Pro 在台式电脑或便携式电脑上运行，或者借助 FileMaker Go 在 iOS 设备上运行。

此外，此处探讨的同步方案将同步过程分成两个部分：交换数据和处理更改。业务规则将处理更改作为数据交换的一部分。通过单独执行这两个功能，您可以在“脱机”状态下完成部分同步过程，从而最大限度地缩短 FileMaker Go 所需的网络连接时间。然而，如果高效设计的同步脚本运行速度足够快且性能足够可靠，那么需要对数据交换和处理这两个过程进行合并或是拆分将由其他因素决定。一般来讲，需要在移动设备上进行的操作越少越好，这样可以最大限度地节省需要在 iOS 设备上耗费的资源和时间。统观同步过程，这种方法可以为远程用户缩短等待同步完成的时间。

交换数据

同步过程的一个关键部分是在文件之间传输待同步的数据，比如，FileMaker Go 上的部署文件和 FileMaker Server 上的托管文件之间进行数据传输。同步过程可能包含以下几个环节：导入记录、使用同步脚本管理的关系或者调用托管文件中的脚本来创建、修改或删除记录。同步过程可将新记录移至各自对应的表中，并对现有记录进行编辑，或者将新记录和已编辑的记录移至中间表。在这些记录被分别处理之前，中间表将这些记录与“活动”表分隔。

然而，数据交换并非适用于所有数据（比如，最后一次同步后未被修改过的数据无需进行数据交换）。为了识别需要包含的记录，您的同步过程首先需要查找：

- 最后一次同步后任意一个文件中被删除的记录
- 最后一次同步后被修改的记录
- 新记录（最后一次同步后在任意一个文件中创建的记录）

处理记录

无论活动记录在数据交换过程中被更新，还是在单独过程中被更新，都需要遵循同一个的观念：创建日期早于最后同步时间且两个文件中都不包含的记录应当被删除；两个文件中都包含且最后一次同步后被更改过的记录应当被修改；任意一个文件中创建日期迟于最后一次同步时间的记录应当同时在另一个文件中被创建。

同步过程中可能具有挑战性的一个问题是修改的优先权。如果托管文件中的记录在早晨被修改，部署文件中同样的记录在下午被修改，那么哪次修改占有优先权？对这个问题以及其他类似问题（比如是否允许用户删除解决方案中的记录）的回答要以系统的业务规则为准。关于这些问题的进一步讨论，请参阅下文“开发您的同步策略”中“业务规则”章节。

FileMaker 脚本

有时候，用户可能需要手动将记录导入 FileMaker Go 和 FileMaker Pro 文件，这样做比较复杂，而 FileMaker 脚本可以很好地处理这一过程。该脚本会依附于一个按钮，远程用户可以点按这个按钮开始进行同步。如果需要在数据交换后处理托管文件中的记录，那么部署文件可能在 FileMaker Server 上同步脚本中的适当位置设置或启动另一个脚本。或者，设计和部署 FileMaker 系统以运行托管文件上的脚本，从而执行该任务。

使用 FileMaker Pro 或 FileMaker Pro Advanced 在 FileMaker 数据库文件中创建 FileMaker 脚本，然后，在以下三种文件中运行脚本：使用 FileMaker Go 部署的文件、在 FileMaker Pro 中使用的文件以及 FileMaker Server 上的托管文件。如需进一步了解如何使用 FileMaker Pro 编写脚本，请参阅 FileMaker 技术网站（FileMaker TechNet）所提供的资源，还可参阅第 6 模块 FileMaker 培训教程（两个链接均在“资源”章节中提供）。

FileMaker 组件的功能和作用

如上文所述，本文假定您将同步 FileMaker Go 上运行的部署文件和 FileMaker Server 上运行的托管文件（如同步简介中提到的，这些同步概念还适用于两个 FileMaker 文件之间的同步）。

FileMaker Go 解决方案和托管解决方案之间的同步包括三个组成部分：FileMaker Pro、FileMaker Server 和 FileMaker Go。下图显示了这三个部分如何协作：

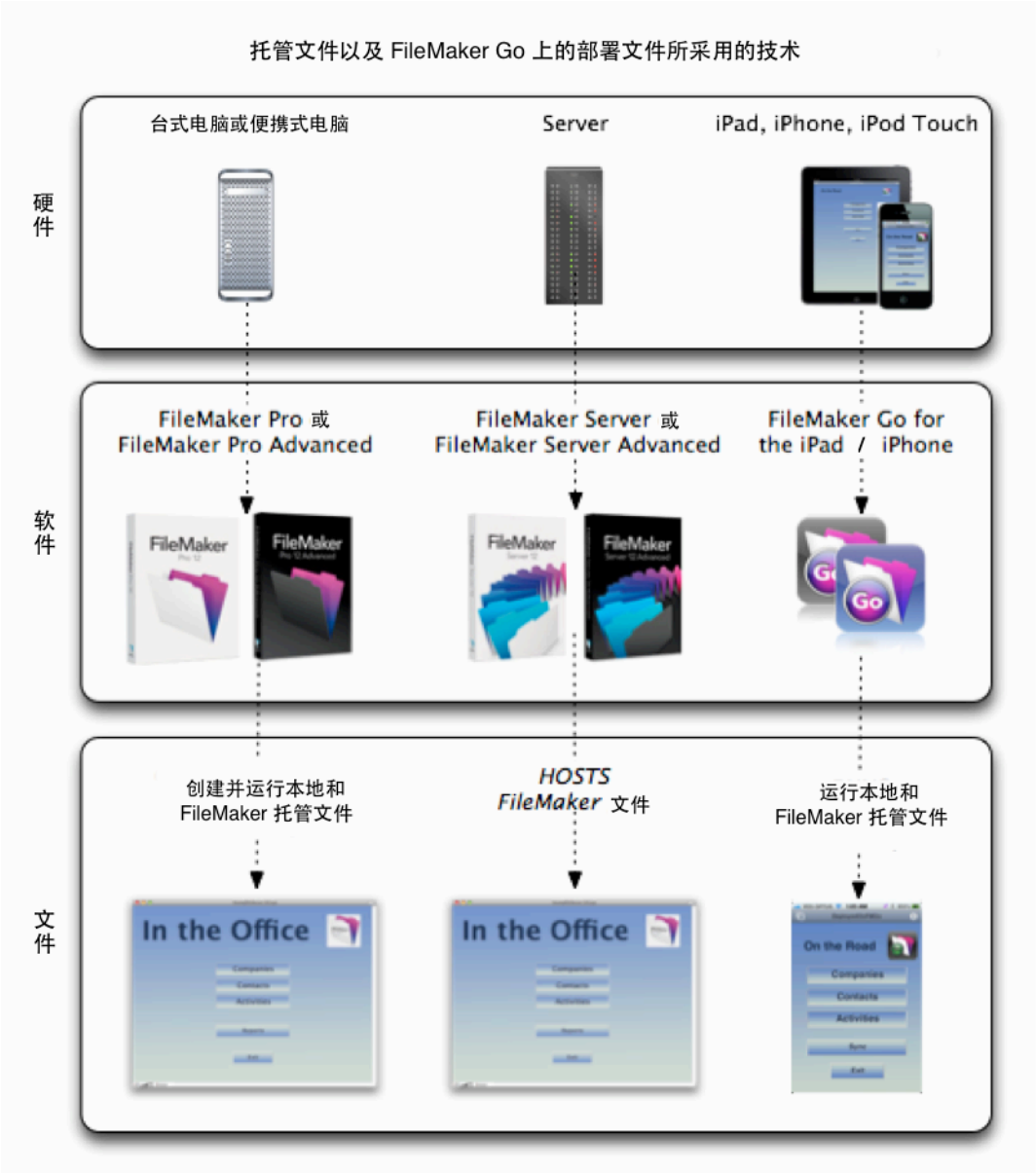

图 3：创建、托管和运行 FileMaker 文件所需的组成部分。

开发您的同步策略

开发同步策略时，您需要考虑下列问题：

- 如上所述，使用 FileMaker Go 打开的本地文件中可以包含托管文件的文件参考，而托管文件中则不包含使用 FileMaker Go 打开的本地文件的文件参考。因此，可以建立 FileMaker Go 上的部署文件到托管文件或到同一部设备上的文件之间的关系，同时调用其他文件中的脚本。但是，不可以与 iOS 设备之外的文件建立关系，也不可以从 iOS 设备之外的文件调用 FileMaker Go 文件中的脚本。然而，在 iOS 设备上打开的托管文件可以使用 fmp URL 协议打开文件并执行 iOS 设备文件上的脚本。（有关 fmp URL 协议的文档可参阅本文最后“资源”章节提供的 FileMaker TechNet 网站。）
- 如果 iOS 设备上打开的文件在任何其他在设备上打开的文件中找不到对应的表摹本，那么文件可能被关闭。也就是说，如果 iOS 设备上打开的文件包含依赖于设备上打开的其他文件（包括托管文件）的表摹本，即便用户手动关闭这些其他文件的窗口，文件也不会被关闭。
- 如果网络连接丢失，访问托管文件的 FileMaker Go 中运行的脚本将会中断。同样，如果 iOS 设备的焦点发生改变（比如，接到来电或用户点按主屏幕键），FileMaker Go 脚本也将被中断。
- FileMaker Go 将尝试重新连接托管文件，并在恢复网络连接时恢复被中断的脚本。
- 追踪同步状态（也就是最后一次成功同步的时间戳）可有效地进行接下来的同步过程（仅处理上一次同步后添加或更改的记录）。
- 每个用户（或设备）应当生成新记录独有的主键，以确保在不同设备上创建的记录拥有不同的主键。
- 记录特定的创建时间戳、修改时间戳和同步时间戳将有助于识别有待同步的记录。

从 iOS 设备角度进行计划

由于托管文件不能引用 iOS 设备上的部署文件，部署文件必须启用同步过程。因而，通过关系或导入过程与托管系统传输的任何数据必须由部署文件或 iOS 设备上的另一个文件进行管理。或者，在脚本参数和/或脚本结果中将数据作为限定的队列在文件之间传输。本文对于这种数据传输方法不做详细介绍。

iOS 设备上的文件可能拥有托管文件的文件参考。因此，“连接器”文件可以被部署在设备上，并用于在托管文件和部署文件之间传输数据，在这种情况下，连接器文件将拥有指向托管文件和部署文件的表摹本，而部署文件将不包含任何指向托管文件中的表的表摹本。以这种方式使用连接器文件的益处主要在于当同步过程结束关闭连接器文件时，部署文件也可能会关闭托管文件（然而如果部署文件包含与托管文件相关的表摹本，它将不会在没有关闭部署文件的情况下关闭托管文件）。下文“逐步介绍同步进程”章节将详细讨论连接器文件。

了解了部署文件打开和关闭托管文件与连接器文件的概念之后，您可以参考下列建议脚本化程序采用的主要步骤：

1. 部署文件运行脚本，以确认托管文件是否可用来同步。
2. 如果可以，部署文件打开连接器文件。
3. 连接器文件识别部署文件和托管文件中需要同步的记录。
4. 如果识别出任何需要同步的记录，连接器文件将根据业务规则指导完成添加新记录、编辑现有记录和删除记录的过程。具体方法如下：
    - 添加/编辑/删除部署文件和托管文件中的记录。
    - 将记录添加至部署文件和托管文件的中间表，然后启用两个文件中的后处理脚本序列。
5. 部署文件关闭托管文件。

在 FileMaker Go 中运行脚本的其他注意事项：

当 FileMaker Go 上的文件中正在运行允许用户异常中断的脚本时，如果用户点击屏幕，FileMaker Go 将询问用户是否继续运行脚本。脚本中的用户异常中断为默认设置，除非“允许用户异常中断”被设定为 [关闭]。虽然“允许用户异常中断 [关闭]”不适用于 iOS 设备上运行的全部脚本，该选项至少适用于同步脚本中修改记录数据的部分，这样可以保持事务的完整性。如果您想要允许用户在其他位置中止同步过程，那么需要在修改记录步骤之前使用“允许用户异常中断 [关闭]”，并在修改记录步骤之后使用“允许用户异常中断 [开启]”。或者，为了防止用户随时中止同步过程，可在此处讨论的同步进程中的脚本顶部添加“允许用户异常中断 [关闭]”。“允许用户异常中断”状态将传递给子脚本，因此，如果一个子脚本将被适当设置“允许用户异常中断”的另一个脚本调用，您可以选择不在该子脚本中包含此步骤。

安全性

同步过程要求能够创建、编辑和（根据业务规则）删除托管文件和部署文件中的记录和数据。如果任意一个文件中的用户账户不具备这些权限，那么可调用运行完整访问权限的目标文件中的脚本以执行相关操作。比如，设置为删除托管文件记录的脚本可被设定为运行完整的访问权限，并可删除通过脚本参数传递其主键的记录。

附加的安全选项用于设置 FileMaker Server，从而为数据库服务器和 FileMaker 客户端之间的数据传输加密。如果需要加密，您的解决方案可以使用 Get(ConnectionState) 函数确认与主机的通信使用安全套接字层（SSL）。

同样，FileMaker 的文件访问功能允许您批准指定外部文件与托管文件协作。这一约束条件可让您对可以访问托管文件的文件进行控制，从而为同步过程提供多一层的安全保护。

最后，如果托管文件位于安全网络之内，则需考虑在安全网络之外设定一个 FileMaker Server 实例，充当 iOS 设备和主要主机之间的媒介。本方案中，系统安全性部分允许与安全网络之外的部分进行联络，并进行记录交换以通过部署文件实施同步过程。请查阅 FileMaker 在线研讨会“FileMaker Go 商务应用：奥斯汀会展中心”了解本方案的应用实例。

网络连接

网络连接是同步的关键 – 没有网络连接，同步就不能进行。由于无法保证 iOS 设备拥有不间断的网络连接，同步过程很容易被中断。远程用户将通过无线网络（来自主机数据库服务器的相同或不同子网）或可用的蜂窝技术（比如 3G 或 4G）进行连接。这些方法都不确保足够的带宽或不间断的连接。

设置服务器可用性

为了应对连接丢失的可能性，您可以在脚本中添加经常性测试以确认托管文件的可用性。请参阅“逐步介绍同步进程”中的“中断后恢复同步”章节，了解构建脚本的相关信息。

第一个测试在同步进程的开始进行，查看托管文件的可用性。将“错误捕获”设为“打开”，并使用“打开文件 []”脚本步骤调用托管文件。Get(LastError) 的结果将指示文件是否可用。

如果托管文件不可用

托管文件可能因为没有打开文件而不可用。这种情况下，FileMaker Go 将获得来自 FileMaker Server 关于文件不可用的快速回复，且 Get(LastError) 结果为 100。

托管文件也可能因为缺少到 FileMaker Server 的网络连接而不可用。这可能是由下列因素引起的：

- iOS 设备没有网络连接
- iOS 设备用于通信的网络没有连接至 FileMaker Server 网络
- FileMaker Server 网络不可用
- FileMaker Server 没有运行

如果缺少到 FileMaker Server 的网络连接，FileMaker Go 将尝试连接大约一分钟，然后返回 Get(LastError) 结果为 802。

如果网络连接丢失

如果打开 iOS 设备上的文件时到托管文件的网络连接丢失（或者 FileMaker Server 上的文件被关闭时），FileMaker Go 将尝试重新连接数秒，在此期间远程用户将无法对任意 FileMaker Go 文件进行操作。如果在此期间连接恢复，FileMaker Go 将重新尝试与网络连接中断之前打开的托管文件进行连接。如果托管文件在网络连接恢复之后可以使用，那么第一个问题是 FileMaker Go 是否将自动进行重新验证（输入之前为文件输入的用户名和密码）。但是首先...

如果 FileMaker Go 休眠

其他导致托管文件连接中断的途径包括：

- 用户点按 iOS 设备上的主屏幕键
- 用户使用四指轻扫手势切换至 iOS 设备上的另一个应用
- iPhone 接到来电或者 iOS 设备接到 FaceTime 呼叫
- 用户使设备休眠
- 设备自动进入休眠状态

在这些情况下，FileMaker Go 在后台运行，并进入休眠或悬停状态。FileMaker Go 重新回到前台时尝试重新连接托管文件。如果已超越为用户权限专门设定的重新验证阶段（如果有），FileMaker Go 将与托管文件进行重新验证。更多详情请查看 FileMaker Pro“帮助”中的“fmreauthenticate”。

连接中断后的记录修改

FileMaker Server 根据不同情况以不同方式处理记录修改。一条记录被用户或脚本编辑时：

如果 iOS 设备...	托管记录将...
失去网络连接	不被修改，被锁定*
关闭文件或窗口（用户操作）	被修改，不被锁定
使 FileMaker Go 进入休眠状态	不被修改，被锁定*

* FileMaker Server 确认用户不再是托管文件的客户时释放被锁定的记录。

追踪同步状态

为了诊断同步进程中的潜在问题，使用同步日志追踪已完成同步的部分非常有帮助。比如，下列日志显示了同步过程的步骤列表：

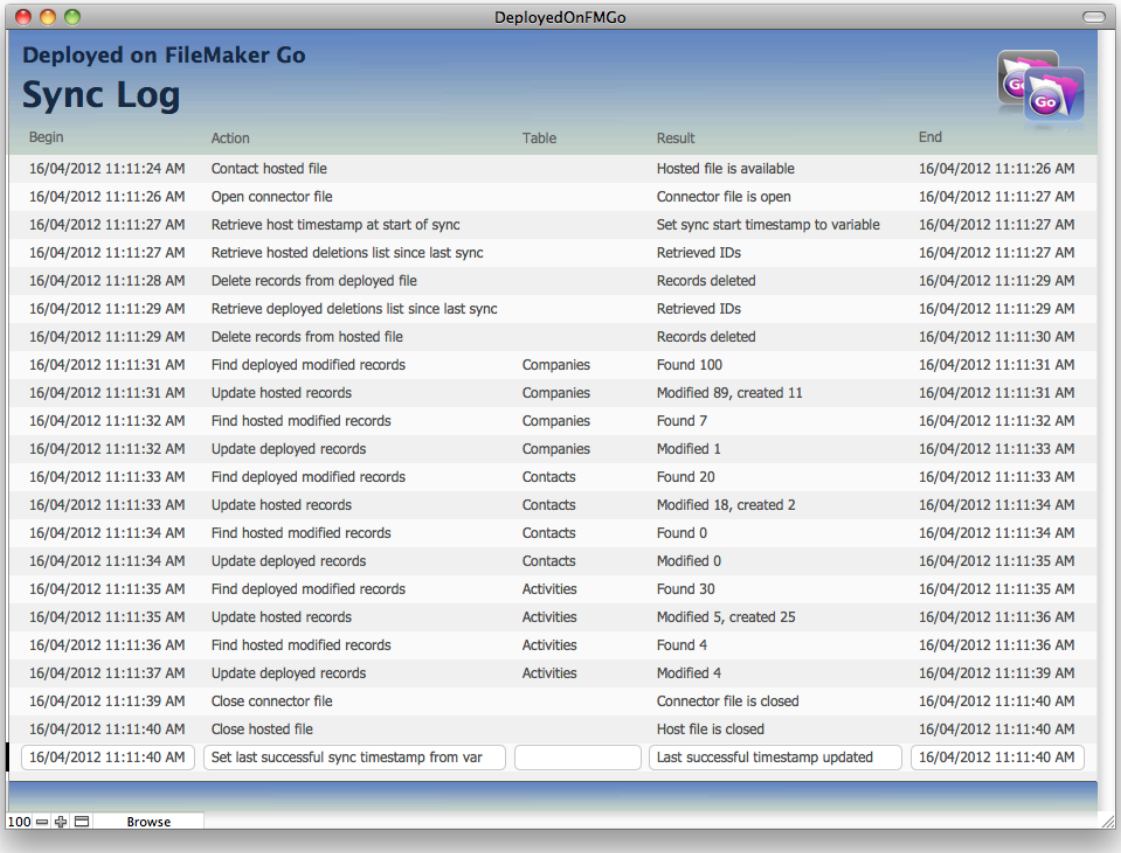

图 4：同步日志包含每个步骤的记录

识别待同步记录

为了让同步脚本识别需要包含的记录，并指明哪些记录在特定时间已被同步，每条记录都需要一个追踪修改时间的字段。本文中这个字段称为 zModStamp，并在进行所有修改时（同步期间进行的修改除外）自动更新为当前时间戳。同步期间，该字段应当被更新，以反映拷贝更改的记录修改时间戳值。

管理这项操作可使用依赖两个其他字段的自动输入计算（替换现有值）：称为 gModOverride 的全局存储时间戳字段（位于您解决方案的“系统”表中）以及称为 ModTrigger 的自动输入修改时间戳。zModStamp 中的自动输入计算可使用下列公式：

```
Let(x = System::gModOverride; If(ModTrigger and x; x; ModTrigger)
```

同步脚本能够通过将 zModStamp 值和最后成功同步的时间戳进行对比来识别待同步记录。

注释：上述字段采用简单的命名方式命名，可以被称为（不单是）“匈牙利标志法、骆驼拼写法、没有空格”。匈牙利标志法可以通过名称指明对象的功能。此处应用的命名方法为某些字段名称加上前缀，比如，为计算字段加上“c”；为诸如时间戳的内务字段加上“z”；为全局字段加上“g”等等。诸如文本和数字的数据字段通常不包含前缀。骆驼拼写法为字段名中单词的第一个字母使用大写格式，这样可以让名称更容易辨认，无需空格。

识别待同步记录的另一种方法是使用修改和删除日志。同步脚本可以使用该日志，通过对比日志中的记录 ID 和时间戳与待处理表中的记录 ID 和时间戳识别有待修改或删除的记录。

可以通过对比创建时间戳与最后同步时间戳，在与处理修改相同的脚本中处理删除。如果创建时间戳先于最后同步时间戳，且记录仅存在于部署文件或托管文件中，当该时间戳在一个系统中被删除时，在另一个系统中也应当被删除。如果创建时间戳迟于最后同步时间戳，且记录仅存在于一个系统中，那么该时间戳为新增时间戳，应当被添加在另一个系统中。

业务规则

正如上述“处理记录”中提到的，业务规则将确认哪些修改（当相同记录被至少一个用户修改时）的优先权高于其他修改。设定业务规则时，一开始即可询问“当发生冲突时，谁将胜出”？回答可以是直接的，比如“远程用户胜出”或“托管文件胜出”。回答也可以更为详细，比如“拥有最近修改时间戳的记录胜出”或“拥有最近时间戳的记录胜出，但不适用于管理员在更早的时间做出修改的情况”。

下表列出了几个需要考虑的业务规则，以及相应的记录处理操作：

如果远程用户…	他们的同步进程将…
仅需要来自主机的记录	下载记录
创建新记录	上传新记录
修改记录并使其优先权高于在主机上做出的修改	上传修改的记录，改写托管记录
修改记录并使其优先权不高于在主机上做出的修改	只要主机上的记录没有被修改，即可在同步过程中修改托管记录，并根据托管记录中的更改修改部署记录
删除记录	删除创建时间戳先于最后同步时间戳的记录，这种情况下记录存在于一个系统，而不存在于另一个系统中

您可能需要考虑一些其他因素来确定对同步脚本做出哪些更改，比如哪些用户在众多远程用户之中处于优先地位、更改发生在一天中的什么时间或一周中的哪一天、在哪部设备上进行了更改（服务器还是 iOS 设备）、用户所属的组别（管理者还是管理员）、远程用户的位置以及远程用户的距离等。该列表无法包含所有因素，重要的是理解适当的业务规则并将其逻辑应用于修改记录的脚本中。

某些情况下可能需要用户审核并手动调整，这同样也能反应业务规则。由于同步过程可对每条记录进行处理，脚本无法解决的任何冲突都被列为异常情况。该列表应包含唯一的记录 ID 以及帮助审核

人员确定需要接受的更改的字段值。本文对审核机制不做介绍，但如果您创建了审核流程，在流程中包含入口、导航和/或依附于按钮的脚本有助于审核人员有效地进行更改。

记录创建/修改指令

业务规则的另一方面与记录创建指令息息相关。例如，如果您的解决方案为公司、发票、项目和付款创建了表，您将拥有一个使用应付款项报表的用户。如果用户在同步过程中提取报表，而付款表中记录的更新先于项目表或发票表，那么应付款项报表可能不准确。如果客户、订单和项目中的记录在付款之前被修改，那么也可能会出现同样的问题。

为了最大限度地降低这个问题造成的影响，您可以在向“父”记录提交更改之前通过入口或关系修改相关记录（如果有）。这样可确保一组完整的相关数据作为一个事务同时提交，而不是同时修改一个表中的所有记录，并在逐条修改之后提交记录。如果提交之前修改了一个上下文中的相关记录，系统可在提交之前将更改保存在缓存中。这种方法的优势在于如果脚本在提交之前被中断，系统不会修改父记录或任何相关记录。

还要考虑允许运行同步脚本的时间，比如报表将被提取的时间段之外，以及订单表数据将被更新的时间段。

交换数据

如果您的业务规则要求远程用户数据被整合到“活动”系统之前需要进行转换或审核，我们建议部署系统中的同步脚本仅仅交换记录。随后，在单独过程中处理托管文件和/或部署文件接收到的数据。这样可以确保最大限度地缩短同步过程需要访问主机的时间，允许关闭托管文件之后继续进行数据处理。

在这种情况下，您需要在一个或两个文件中创建中间表。这些表可以位于托管和/或部署文件中，也可以位于专门建立的单独文件中。这些表应当与现有表和活动表准确匹配，自动输入功能需要被禁用的情况除外。因而，将数据被移至这些表时，ID、时间戳等都不会被修改。

记录创建

将记录移至中间表，而不是在同步过程中修改活动记录；使用“导入”脚本步骤或循环查找记录组；并使用“设置字段”脚本步骤（通过启用了“允许通过此关系在本表中创建记录”选项的关系）创建并填充新记录。请参阅下列“记录创建和修改”章节了解通过“关系图”上的“pivot”关系创建记录的方法。

其他数据交换方法

除了使用导入或关系在各个表之间移动记录之外，还可以将记录导出至或电邮至另一个位置然后再导入。

可将记录数据作为“执行脚本”步骤中的参数进行传递，也可将记录数据作为 FMP URL 协议中格式化字符串中的参数，并使用“打开 URL”脚本步骤进行传递。不论使用哪种方式传递，数据最终都被传回至脚本结果中的调用脚本中。以任意一种方式调用脚本都将在本地设备上打开托管文件。

在 FMP URL 协议中，不按字母顺序或数字顺序排列的字符不可使用 URL 编码。请注意，URL 协议不支持容器数据。此外，容器数据只能被导出至 FileMaker 文件，因此，FileMaker 导出格式需要能够交换包含容器字段的记录数据。

还可以使用基于 php 的系统交换记录数据。部署文件或连接器文件可借助已配置的可与托管文件兼容的网页向页面发送 URL 请求，以调用在托管文件中添加、编辑、删除和检索记录的脚本。进一步讲，可配置部署文件或连接器文件通过 php 与 SQL 系统协作，以提供强大的功能，或使用 ODBC/JDBC 技术交换数据。本文对这些选项不做详细讨论。

使用交换的记录实时操作

一旦记录被移至中间表，iOS 设备上的同步脚本将关闭连接器文件和托管文件。目前，托管文件拥有一组处理并与活动系统整合的新记录。FileMaker Server 中预设的脚本或机器人系统（专门的 FileMaker 客户端部署中运行的有特定目的的文件）中运行的脚本可以处理记录。无论哪种情况，脚本都将持续或按照预设时间间隔运行该过程，以便恰当地处理数据队列。

可设定处理器脚本查找中间表（一个或多个）中的新记录，这些中间表用来接收同步过程中传输的数据。当脚本检测到等待处理的中间记录时，会采用相关的业务规则调整中间数据，使之与主机中对应记录的内容相符。

同样，部署文件可能需要在完成与托管文件的数据交换后处理中间记录。

处理记录要求使用上述相同的逻辑，以确认同步过程中需要改写哪些数据 –“获胜”的记录是由业务规则确定的，比如在部署文件还是托管文件中修改记录，用户是否拥有特殊角色等等。如上所述，需要一个改写机制以确保被同步数据的修改时间戳与源记录的修改时间戳相匹配（而不是反应数据交换或后处理发生的时间）。

关于对比记录、保持修改时间戳的匹配以及保持恰当的主键和更新记录的详细内容，请参阅下节内容以及本文随后“逐步介绍同步进程”中的“处理记录”小节。

记录修改

记录数据可通过手动或脚本化过程被修改 – 使用与主要字段中数据匹配的导入或逐个字段修改。后者可以通过关系进行，或者直接导航至基于相关表的布局上待更新的记录。

可配置导入为添加新记录、更新匹配记录或更新现有记录。这是一种单向更新记录直接有效的方法，但无法根据业务规则选择并提供有待更新的记录。此外，需要一个导入程序更新所有字段，这一程序暗示字段是否被更改过，但最终可能导致（由索引和其他次要过程导致的）同步性能可能不是最优化的。

能够提供更有效控制且更为有效的方法是循环每条记录，对比每个字段中的值并仅仅更改那些与其他字段不同的字段。“逐步介绍同步进程”章节中的“处理记录”对这一方法做出了更详细的介绍。

使用关系对比和修改记录

关系是对比两个记录的有效方法。这需要一个基于当前上下文的表摹本，也就是依附于布局的表，对比开始时运行同步脚本，且基于表的第二个表摹本需要被检验（如下列图 5 所示）。如果当前上下文不是需要对比的表之一，当前上下文则需要第三个表摹本（如下列图 6 所示）。

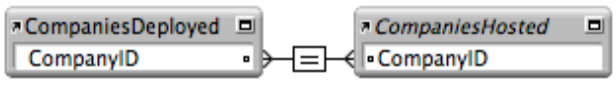

图 5：部署文件和托管文件中 Companies 表的表摹本，通过 CompanyID 关联。

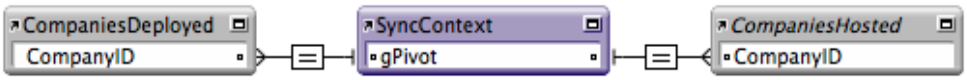

图 6：三个表之间关系的表摹本，其中中间的表摹本是同步脚本的上下文。

生成独有的主键

上述关系的建立基于待对比表中的主键。每条记录的主键必须是独有的，这样，每条记录才能保持各自的身份以及与其他记录的关系。比如，公司可能拥有相关联系人，这些联系人可能会有相关活动。公司主键将相关联系人记录与公司进行关联，而联系人主键将相关活动记录与联系人进行关联。

同步的目标是在部署文件和托管文件中建立一组匹配的记录，其中包括主键。也就是说，每条记录应当拥有独有的主键，部署系统中的记录和关系应当确切地反应托管系统中的记录和关系。

如果同步要求将一部设备作为移动新记录的来源，比如从主机移至托管文件，那么使用 FileMaker 的自动输入序列号选项即可生成独有的主键。然而，如果需要在托管系统和一个或多个部署文件中同时创建记录，那么所采取的方法应当避免在同一系统不同副本中创建的记录之间发生主键上的冲突。

FileMaker 的 Get(UUID) 函数是自动输入序列号选项的另一种方法。它在创建记录时生成一个唯一的 32 字符的值，这个值可作为主键使用。对于使用 FileMaker 12 创建的解决方案，我们推荐使用这种生成独有主键的方法。

使用时间戳字段

FileMaker 的字段类型之一是时间戳，包含日期和时间，可能以“4/18/2012 2:49:10 PM”格式显示。可配置时间戳字段自动输入创建或修改记录的日期和时间，也可以通过用户或脚本进行更新。为了追踪记录修改的时间，请确保设置一个自动输入修改时间戳的字段。为了便于同步，您可以创建一个改写机制，以确保同步记录中的修改时间戳与源记录中的修改时间戳互相匹配。

除单独记录时间戳之外，部署文件中应当包含一个用于存储最新成功同步时间戳的单个记录表，而同步中应包含部署文件中修改时间戳迟于该时间戳的记录。（为了高效使用，这个单个记录表还可执行其他功能，比如存储图片、实用工具字段以及其他解决方案数据。）如果您确信部署文件始终适用于单个用户，可以使用全局字段存储最后一次同步的时间戳。

不同时区的用户需确保在使用同步时间戳之前调整时差。比如，对于与主机不在同一时区的用户而言，您需要设置用户位置和主机之间的时差，并在部署文件中的修改时间戳值上进行加减，然后再与托管文件对应记录中的时间戳进行对比。当用户在不同时区移动时，部署文件将捕获每次修改的位置值，从而计算如何适当调整时区。

设定时差的可选方法有：

- 在部署文件的每个副本中存储用户时区的相关信息
- 提示用户在部署文件的用户偏好设置表中输入并更新时区值
- “擦除”返回主机当前时间的网页，并分析 html 源代码中的时间值
- 使用返回设备经纬度信息的 Location () 或 LocationValues () 函数，并引用与时区经纬度关联的存储表（或在线源代码）

事务完整性

基于网络中断的可能性，使同步脚本在一个或一组指定记录提交之前完成全部修改是非常重要的。修改应当通过以下方式完成：在最短的时间内锁定记录并在记录处于锁定状态时进行相关更改，这样可以一次性传输整套修改，确保事务的完整性。如果传输记录时网络连接丢失，记录将不会被提交，记录的所有修改将会丢失。同时，被锁定而无法修改的记录应当被添加至上述异常处理列表中，并在适当时间报告给用户。

FileMaker 提供记录锁定功能，因而两个用户同时更改记录时不会发生冲突。一旦一个用户首先提交了记录，另一个用户即可对记录进行修改。在将记录添加至异常处理列表之前，同步脚本可尝试多次修改被锁定的记录。

为了修改未被锁定的记录，脚本应被设置为锁定记录，并尽可能高效地更改并提交记录（从而解除锁定）。这样势必将每个字段与其在对应记录中的副本进行对比，并只更改那些不同于其他字段的字段。另一种方法是“预先加载”只包含字段名和字段值的变量，这些字段名和字段值将被更改，并使用变量中的值只对那些列出的字段进行更改。第三种方法是使用“更新匹配记录”选项导入一个或多个记录。

不论您使用哪种方法更改记录，都应使用改写机制确保修改时间戳字段的匹配，并显示最初进行更改的确切时间，而不是当前同步发生的时间。

这种方法设定了上述“识别待同步记录”中的标准 – 当前同步中应包括任意文件中修改时间戳等于或大于最后同步时间戳的记录。

注释：多数情况下记录更改需要作为一个整体被接受，以保持事务的完整性。然而，如果业务规则允许对单个字段值分别进行修改，您可以提供一个用来追踪每个记录中一个或多个单独字段修改时间戳的机制。下列相同的修改方法适用于同步字段级修改 。

打开和关闭托管文件

为了避免给用户造成不便，同步过程应在需要时打开托管文件，在不需要时关闭托管文件，从而尽可能缩短打开托管文件的时间。如果托管文件未被关闭而网络连接丢失或中断，FileMaker Go 将暂停工作，尝试重新连接网络，并提示用户在重新建立网络连接时进行重新验证。我们需要尽可能避免网络连接问题造成的延迟和提示。

正如本节开始所提到的，只有当其他打开文件在托管文件中都没有表摹本引用表时，FileMaker Go 才允许关闭托管文件。下表列出了关闭 iOS 设备上托管文件的方法：

如果部署文件...	如果...果托管文件将打开	如果...托管文件将关闭
拥有主机的文件参考	“打开文件”[托管文件] 脚本步骤或“执行脚本”[托管文件中的脚本] 脚本步骤被调用	“关闭文件”[托管文件] 脚本步骤或“执行脚本”[托管文件中关闭文件的脚本] 脚本步骤被调用
拥有基于托管文件中表的表摹本	上述操作或引用托管文件中字段的任意脚本步骤（比如“设置字段”[]）被调用	表摹本基于托管文件的当前文件和任何其他文件被首先关闭

拥有基于指向托管文件中表的表摹本的布局	上述操作、用户或脚本导航至布局	表摹本基于托管文件的当前文件和任何其他文件被首先关闭
包含引用基于托管文件中表的表摹本字段的脚本或计算，或拥有包含托管文件字段的布局	运行相关脚本，评估相关计算和/或显示相关布局	表摹本基于托管文件的当前文件和任何其他文件被首先关闭

关闭同步方案中托管文件的意思是必须在关闭托管文件之前关闭表摹本基于托管文件中表的任何文件。如果部署文件包含这些表摹本中的任意一个，则必须在关闭托管文件之前关闭部署文件。可以通过以下方式避免这种情况：只拥有一个指向部署文件中托管文件的文件参考（但没有表摹本），使用一个同时在部署设备上打开的单独连接器文件处理基于部署文件和托管文件中表的表摹本之间的关系。仅仅使用部署文件中该连接器文件的一个文件参考，部署文件即可在同步完成后关闭连接器文件，然后关闭部署文件。

按照需要打开并关闭托管文件，并按下列顺序打开、同步和关闭文件：

1. 部署文件打开托管文件。如果这一步成功，则同步过程继续。
2. 部署文件打开连接器文件，连接器文件包含托管文件和部署文件的表摹本。
3. 部署文件调用连接器文件中的脚本同步记录。
4. 部署文件关闭连接器文件。
5. 部署文件关闭托管文件。

用简单易懂的语言介绍同步进程

本节介绍了同步过程的要点，目的是为您的设计和制作提供帮助。整个过程中，请您记录操作、监控错误、必要时暂停同步过程并给用户适当的提示。本节后面的章节不仅在概念上介绍了同步过程的必要组成部分，也列举了 FileMaker 脚本步骤的具体实例。

第一部分 – 在 FileMaker Go 部署文件中

1. 用户点按部署文件中的按钮开始同步。
2. 部署文件中的脚本查看托管文件是否通过调用“握手”脚本打开了托管文件。
    a. 确认托管文件可用之后，部署文件中的脚本导出并打开连接器文件，然后等候连接器文件运行自己的同步过程。
    b. 如果托管文件不可用，脚本将通知用户并暂停。

第二部分 – 在连接器文件中

1. 被部署文件调用时运行同步脚本。
2. 连接器文件中的脚本收集记录 ID 列表或“查找”部署文件中每个表内被修改的记录。
3. 取决于记录是否将被这个脚本处理：
    a. 如果设置连接器文件仅用来交换记录，它会将记录数据推送至托管文件中的中间表。它还将收集并上传最后一次同步后部署文件中删除记录的主键列表。
    b. 如果设置连接器文件为处理记录同步，它将循环记录 ID 或记录，并执行需要对相关托管记录进行的修改，包括创建新记录以及删除最后一次同步后被删除的记录。
4. 连接器文件中的脚本重复（上述）步骤 2 和步骤 3，而采用逆向同步（也就是说，把主机文件当做数据源，把部署文件当做数据接收者）。
5. 脚本结束时，更新托管的同步日志记录并在部署文件中创建一个日志记录。

第三部分 – 在 FileMaker Go 部署文件中

1. 脚本关闭连接器文件。
2. 脚本关闭托管文件。如果脚本将在数据交换之后处理记录，脚本目前则处理部署记录，包括循环中间表中的记录、对比时间戳并按照业务规则更新、创建或删除部署记录。
3. 脚本更新部署文件的同步日志，同时，如果同步完成，脚本将把同步开始时间戳写入部署文件中的 System::gSyncTimeStamp 字段。
4. 脚本将在设备同步完成时通知用户（比如，显示自定义对话框或更新布局上的字段）。

第四部分 – 在 FileMaker Server 托管的文件或独立的 FileMaker 文件中（如果需要）

1. 预定脚本查看中间表中的新记录。可设定脚本在 FileMaker Server 或机器人中运行，机器人是有规律地运行脚本的独立 FileMaker 客户端文件。
2. 脚本执行其更新，包括循环中间表中的记录、对比时间戳，并根据业务规则更新、创建或删除活动的托管记录。
3. 脚本更新主机文件的同步日志表以及同步状态和同步完成时间戳。

逐步介绍同步进程

对于为 iOS 设备制定的同步策略来讲，现在正是为同步过程制定有效方案的时候。但因为同步的需求会发生很大的变化，没有一种方案能够适用于所有情况。考虑到这一点，本节介绍了常见同步方案中最主要的部分，并为变化情况提供了备选方案。关于同步中各部分的概要，请参阅上一节，“以简明语言描述同步过程”。

> **注释**：按复杂程度来讲，本节中的示例脚本步骤为中级到高级。本节还将涉及包括变量的使用、打开 URL 脚本步骤和 ExecuteSQL( ) 计算函数的高级技术。如果您不熟悉这些技术或下面示例中的其他技术，建议您参考一些培训资料或其他指导资源。您可以在“资源”小节中找到与这些技术相关的资料和下载链接。

本节中的同步进程开发基于以下假设：

- 需要同步的文件为一个部署文件、一个托管文件和一个用于管理同步任务的附加文件。以下示例中将这些文件称为 HostedOnServer.fmp12、DeployedOnGo.fmp12 和 SyncConnector.fmp12。
- 部署文件的用户在部署文件中修改、创建或删除记录。
- 托管文件的用户在托管文件中修改、创建或删除记录。
- 同步将执行记录的修改、创建和删除。
- 同步可能使用实时据或先将记录传输到一个中间表。
- 部署文件中的表和字段与托管文件中的表和字段完全匹配或是其子集，其中包括使用了与托管文件相同名称的表和字段。

“开发您的同步策略”章节中问题的解决方法如下：

- 从 iOS 设备角度进行规划
    - 该进程由部署文件启动。
    - 初始同步脚本包括“允许用户异常中断 [关闭]”和“设置错误捕获 [打开]”（子脚本继承这些设置）。关于同步脚本中这些选项使用方法的详细介绍，请参阅上述“开发您的同步策略”。
    - 同步进程中的所有脚本以完全访问权限运行。
- 网络连接
    - 设定服务器的可用性
        - 部署文件能否建立，取决于托管文件是否可用；如果托管文件不可用，则终止此过程；如果托管文件可供同步，允许此过程继续。
        - 如果托管文件不可用
            - 当服务器不可用时，部署文件将中断同步脚本。
        - 网络链接中断的后果
            - 同步脚本包括关键点测试以判断托管文件是否仍然可用。如果不可用，同步脚本将创建一个日志记录并中断同步。
        - FileMaker Go 休眠的后果
            - 如果 FileMaker Go 被重新启动，它将尝试重新创建与托管文件间的连接并恢复运行的脚本。如果托管文件再一次不可用，同步脚本将创建一个日志记录并暂停同步。
        - 重新连接和重新验证
            - fmreauthenicate 扩展权限被启用并将允许远程用户重新建立网络连接的时间设置为 60 分钟。在您的同步方案中，您可以根据用户工作要求和网络可用性对时间间隔进行合理设置。
- 追踪同步状态
    - 托管文件和部署文件均有“同步日志”表。同步脚本为进程中的每个操作创建日志记录。

- 业务规则
  - 对于已删除记录，脚本将每条记录中的主键与对应表的主键进行对比。如果没有匹配记录并且记录创建时间早于最后一次成功同步时间，记录将会被删除。
  - 示例中的其他业务规则包含“最近修改优先”，无论是在哪个文件中进行修改。
- 识别需要同步的记录
  - 同步脚本在两个文件中执行“查找”或使用 ExecuteSQL( ) 函数来收集记录 ID，隔离最后一次同步后在任何一个文件中被创建、修改或删除的记录。
- 记录的创建/修改顺序
  - 该示例同时处理同一表中的所有记录。比如在前文所述的时效性方案中，您可能选择由父记录开始处理相关记录。
- 交换数据
  - 部署文件启动一个本地“连接器”文件来管理通过关系进行的记录交换和修改。数据交换或同步完成后，托管文件即可关闭。要处理其他选项，还要对替代记录的创建和交换方法进行探讨。连接器文件储存在托管文件或部署文件的容器字段中，并作为同步进程的一部分部署在 iOS 设备中。这省去了用户分别安装和管理连接器文件的过程。
- 记录的修改
  - 关系
    - 连接器文件包含部署文件和托管文件中每个表格间的关系。
  - 主键
    - 部署文件和托管文件均使用 Get(UUID) 函数生成主键。或者，由自定义的计算生成唯一的主键。
  - 时间戳
    - 使用触发器或覆盖字段的修改时间戳被用于每一张表中，以确保在用户修改时得到的原始时间戳在同步过程中被保留（而不是被同步过程中更新记录的时间戳所覆盖）。该技术允许原始（源记录）的修改时间戳被放置在一个覆盖字段（“系统”表中的全局字段）中，并且优先于为当前记录返回 zModStamp 值的计算式中的当前时间戳值。常规的自动输入时间戳字段被用来作为生成修改时间戳计算式的触发器。此外，部署文件“系统”表中的全局时间戳字段被用来存储最后一次成功同步的起始时间戳。
  - 事务完整性
    - 同步脚本在每条记录的字段中循环，与匹配记录中的相关字段进行对比来判断需要修改的字段。“预加载”变量的附加选项及其需要修改的字段也在探讨范围之内。
- 打开和关闭托管文件
  - 实现该同步过程需要打开三个文件 – 部署文件、托管文件和部署在 iOS 设备上的连接器文件。
  - 部署文件先打开托管文件，再打开连接器文件。如果托管文件不可用，部署文件将在打开连接器文件前中断脚本。
  - 同步完成时，部署文件将关闭连接器文件和托管文件。托管文件仅在没有其他文件打开基于托管文件的表摹本时方可关闭，因此必须首先关闭连接器文件。同样，如果在部署文件中存在基于表的表摹本，连接器文件不能被关闭，所以需确认部署文件没有连接器文件的表摹本。当连接器文件关闭时，部署文件可以关闭托管文件。

接下来的章节包括示例脚本、其他选项和细节扩展，并提供基于上述概念的同步进程基本实施纲要。

联络主机

同步进程的第一步是实现部署文件与主机的联络。下列脚本步骤可联络主机，当主机因某些原因不可用时提醒用户并记录结果。为确立托管文件的可用性，该脚本尝试将托管系统中的用户 ID（在本示例中为

用户表的主键）设置到部署文件“系统”表中的全局字段。如果操作成功，托管文件可用且同步进程可开始；如果操作失败，脚本将提示用户网络连接不成功并创建一个日志条目来记录本次操作。

此脚本被设计在基于部署文件“系统”表布局上的部署文件中运行。

```
设置错误捕获 [打开]
允许客户异常中断 [关闭]
设置字段 [System::gPivot; “”]
设置字段 [SyncLog::Action; “联系托管文件”]
提交记录/请求 [跳过数据输入验证; 无对话框]
设置变量 [$token; 值: “Handshake_” & GetAsNumber(Get(当前系统时间))]
#SyncHandshake 脚本在 HostedOnServer 文件中，描述如下。
执行脚本 [“SyncHandshake” 来自文件 “HostedOnServer”; 参数: $token]
如果 [Let(
        $error =
        Case(
        Get(上一个错误) = 100; "托管文件丢失";
        Get(上一个错误) = 802; "无法打开托管文件";
        Get(脚本结果) ≠ $token; "托管文件当前不可用"
        );
        不为空($error)
        )]
    显示自定义对话框 [标题: "连接错误"; 消息: $error & “.” 默认按钮: “确定”, 提交: “否”]
    设置字段 [SyncLog::Result; $error]
    中断脚本
Else
    设置字段 [SyncLog::Result; “托管文件握手成功”]
End If
提交记录/请求 [跳过数据输入验证; 无对话框]
```

在上述步骤中，脚本试图在 HostedOnServer 文件中执行一个脚本。如果 Get(上一个错误) 得到的结果是非零的值，或者脚本结果不等于作为参数传递给托管文件脚本的标记的值，自定义对话框将通知用户同步无法继续，且脚本将在日志中创建一条同步失败的记录。同步日志的关系应设置为允许创建记录；要了解关于示例同步脚本中被广泛使用的强大关系功能的更多详情，请参阅上述“记录的创建与修改”章节。

以下是建议在 HostedOnServer 文件“SyncHandshake”脚本中包含的内容：

```
退出脚本 [结果: Get(脚本参数)]
```

编写脚本注意事项：通过使用变量的计算式可以写出精简的脚本，从而处理条件的变化。在这些示例脚本中，变量一般以一或两个字符命名，如：错误变量 $e 和时间戳变量 $ts。变量有多种用途，如：存储时间戳和记录 ID 的列表，为“设置字段（按名称）”脚本步骤计算字段名。变量不但可以在“设置变量”脚本步骤中，而且还能在 Exit Loop If[ ]、If[ ]、Else If[ ] 和“设置字段[ ]”脚本步骤的选项中设置与修改。

部署一个连接器文件

如果您正在使用被导出至 iOS 设备并被打开的连接器文件，它可能被部署在 iOS 设备上或存储在部署文件或托管文件的容器字段中。假设连接器文件被存储在部署文件“系统”表中名为 gContainer 的字段中，部署文件中的如下脚本步骤将导出并打开连接器文件：

```
设置字段 [System::gPivot; “”]
设置字段 [SyncLog::Action; “打开并导出连接器文件”]
提交记录/请求 [跳过数据输入验证; 无对话框]
设置变量 [$path; 值: "file:" & Get(文档路径) & "SyncConnector.fmp12"]
导出字段内容 [System::gContainer; “$path” ]
设置变量 [$e1; 值: Get(上一个错误)]
If [$e1 = 0]
    打开 URL ["FMP://~/SyncConnector.fmp12?script=Run Sync";无对话框]
    设置变量 [$e2; 值: Get(上一个错误)]
End If
If [$e1 or $e2]
    设置字段 [SyncLog::Result; Case($e1; “导出失败报错” & $e1 & “.”; 连接器文件打开命令失败
        报错 “ & $e2 & “.”)]
    显示自定义对话框 ["系统错误"; Case($e1;“文件导出失败 [error = “ & $e1 & “]”; “无法打开连
        接器文件 [error = “ & $e2 & “]”)]
    中断脚本
Else
    设置字段 [SyncLog::Result; “连接器文件已打开”]
End If
提交记录/请求 [跳过数据输入验证; 无对话框]
```

开始的几个脚本步骤将清除管理 SyncLog 表关系的 gPivot 字段，然后使用 SyncLog 表摹本的“允许创建…”关系创建一个新的日志记录来追踪导出和打开连接器文件的状态。随后的“设置变量”脚本步骤将创建带有 FileMaker Go 自身文档文件夹路径的变量。“导出字段内容”脚本步骤将从部署文件“系统”表中的全局容器字段中导出文件。在以其结果设置其变量后（在以下两个 if[ ] 部分中用于错误的处理、记录和报告），“打开 URL”步骤将使用 FMP URL 协议给最近导出的文件发送一个请求，从而运行一个名为 OnFileOpen 的脚本。通过这种方式调用脚本能够有效地在 FileMaker Go 中打开文件。

在“打开 URL”步骤后，脚本将存储来自“打开 URL”脚本步骤的“上一个错误”结果，如果需要则显示一个错误对话框，并且设置日志记录的结果。

提示：为确保同步进程在最新版本的连接器文件上运行，您可以在连接器文件的 OnFileOpen 脚本中添加脚本步骤，在同步进行前，如果发现主机的容器字段中有可用的更新版本，则进行替换。

连接器文件被设计为数据交换（以及同时执行两个函数时进行记录处理）的中心。它的用途是识别待同步的记录、比较记录数据、交换和/或修改记录以及创建日志记录。无需执行打开和关闭托管文件的操作，因为这两项任务直接由启动进程的部署文件中的脚本执行。

若要执行分配的任务，连接器文件必须具备：

- 部署文件和托管文件的文件引用
- 在部署文件和托管文件中连接对应表的表摹本
- 带有适当字段的表，其中包括创建与其他表摹本关系的全局存储文本或数值字段以及全局存储时间戳字段
- 连接器文件表的表摹本与其他表摹本之前的关系（见图 7）
- 基于连接器文件表的表摹本的布局

- 基于部署文件和托管文件表摹本的布局（如果同步脚本中使用“转到布局”或“转到相关记录”脚本步骤）
- 一个由部署文件调用的、用于识别符合条件的记录的脚本；根据业务规则导出、导入或修改记录并创建日志记录的脚本

下图显示了如何排列关系图，以满足上述条件:

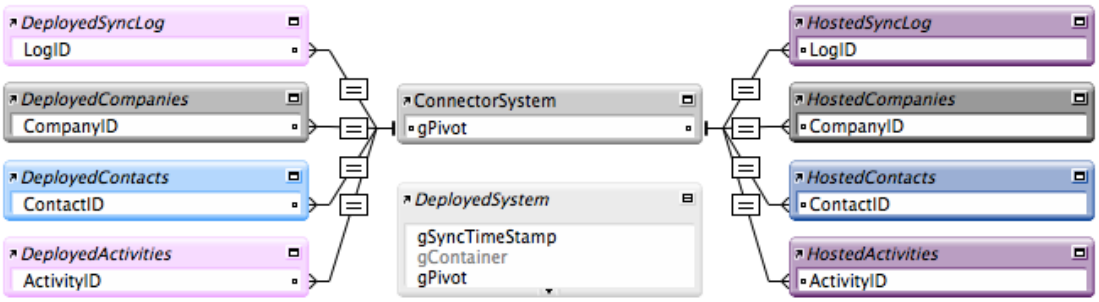

图 7：连接器文件的关系图

在 ConnectorSystem 表摹本中，gPivot 是一个全局存储的文本字段。它将作为关键字段来运行，从而在部署表与托管表之间创建“pivot”键连接。在 gPivot 字段中设置一个主键能够同时为一个或多个表创建有效的关系。

DeployedSystem 表摹本显示在关系图中，所以同步脚本可以与存储最后一次成功同步的时间戳字段一起运行 – 在本例中，部署文件中的全局字段被称为 gSyncTimeStamp。gSyncTimeStamp 字段的全局存储功能无需关系即可允许最后一次成功同步开始时的时间戳用于整个解决方案。

SyncConnector.fmp12 文件中的 Run Sync 脚本应该先定位至相应的布局并为整个同步进程引用的适当时间戳的值设置变量：

```
转到布局 [“SyncCentral” (“ConnectorSystem”)]
设置字段 [ConnectorSystem::gPivot; “”]
设置字段 [DeployedSyncLog::Action; “检索同步运行的时间戳值”]
设置变量 [$SyncTimeStamp; 值: DeployedSystem::gSyncTimeStamp]
设置变量 [$CurrentSyncTimeStamp; 值: Get(当前系统时间)]
设置字段 [DeployedSyncLog::Result; “同步开始时间戳储存为一个变量”]
提交记录/请求 [跳过数据输入验证; 无对话框]
设置字段 [ConnectorSystem::gPivot; “”]
设置字段 [HostedSyncLog::Action; "同步执行人： " & DeployedSystem::gUserID & " 于 " &
    $CurrentSyncTimeStamp]
设置变量 [$HostSyncLogID; 值: ConnectorSystem::gPivot]
提交记录/请求 [跳过数据输入验证; 无对话框]
```

上述时间戳部分提到的 DeployedSystem::gSyncTimeStamp 字段是一个持有最后一次成功同步时间戳的全局储存字段。它被设置到该同步脚本稍后所用到的变量中，并且根据成功同步的时间重新设置 $CurrentSyncTimeStamp 中的值。如果同步未成功完成，DeployedSystem::gSyncTimeStamp 中的值将不被更新，下一次同步将处理上一次成功同步后被更新的记录，而不是未成功同步后可能已被删除或更改的记录。

执行这些步骤之后，变量被设置为带有最后成功同步时间戳和当前时间戳。设置一个带有最后成功同步时间戳的变量意味着值只能从字段中被提取一次，并能够作为脚本的持续时间从变量中引用。$CurrentSyncTimeStamp 变量在创建标志真正同步开始的日志记录之前立即存储时间戳值。同步脚本的最后，如果同步成功，该值将被写入 DeployedSystem::gSyncTimeStamp 字段，作为下一次删除和修改的起点。

创建日志记录需要先清除 gPivot 字段并在日志表中设置一个字段的值（日志表关系中“允许通过这个关系在本表中创建记录”选项处于启用状态）。托管同步日志中也可创建日志记录，且日志记录的 ID 储存在一个变量中，因此它可以在脚本最后用来记录托管日志表中的同步结果（即是否成功完成）。

识别部署文件中的同步记录

部署文件和托管文件中所有已修改的记录必须完成同步。这可以通过将每条记录中 zModStamp 字段与 $SyncTimeStamp 值进行对比来完成。在任意一个文件中，记录的修改时间戳若迟于最后一次同步开始时间，则该记录将作为当前同步中包含的候选记录。如果用户仅需要主机中记录的一个子集，系统将根据定义用户子集的条件（修改时间戳条件除外）选择主机上的同步候选记录。

在您的解决方案中，为使条件的应用可行，您可以选择依次从每个文件（主机文件和部署文件）中表的上下文对比每条记录。或者您可以选择从一个单独的有利位置切入，通过一个来自上下文的关系处理两个文件中的记录。

以下三个示例提供三种识别同步记录的方法：执行“查找”、收集使用特定变量和表名的变量中的记录 ID、收集间接使用变量（当脚本运行时确定变量和表）的记录 ID。

执行“查找”来隔离记录

以下脚本步骤提供了一种示例方法，您可以基于最近成功同步时间戳隔离部署文件中一个单独表中的记录。斜体字步骤可以被以下两种方法中的步骤替代：

冻结窗口
设置字段 [ConnectorSystem::gPivot; “”]
设置字段 [DeployedSyncLog::Action; “在部署文件中查找已修改记录”]
提交记录/请求 [跳过数据输入验证; 无对话框]
转到布局 [“DeployedCcompanies” (DeployedCompanies)]
执行查找 [恢复; DeployedCcompanies::zModStamp: “> $SyncTimeStamp”]
设置字段 [System::gPivot; “”]
设置字段 [DeployedSyncLog::Result; “已找到 ” & Get(查找个数) & “条记录”]
设置字段 [DeployedSyncLog::Table; Get(布局表名称)]
提交记录/请求 [跳过数据输入验证; 无对话框]

或者，使用一个基于全局时间戳字段与 zModStamp 字段间“>”关系的过滤记录关系用“转到相关记录”脚本步骤代替“转到布局”和“执行查找”步骤（在“转到相关记录”脚本步骤之前，您的脚本需要将 $SyncTimeStamp 变量的值设置到全局键字段中 ）。无论使用哪种方法，每个需要同步的表都将执行隔离同步候选记录的脚本步骤。

在一个变量中收集记录 ID

要从整个上下文，而不是在每个表中执行查找或在每个表的上下文中循环查找记录，您的脚本可以收集待同步记录的 ID。首先，脚本应该捕捉修改记录的关键值。一旦脚本收集了待同步记录的主

键值，则不需要使用“转到相关记录”脚本步骤，因为脚本可以通过一次放置一个 ID 到 System::gPivot 字段来实现源表到目的表间的关系，从而依次处理每条记录。在这种情况下，资源表是 DeployedCompanies 且对应表为 HostedCompanies。

下面是一个捕获 ID 的脚本序列示例（可用于替换上述斜体脚本步骤）：

```
设置变量 [$CompanyIDs; 值: ExecuteSQL("Select CompanyID from DeployedCompanies
    where zModStamp > ?"; ""; ¶; $SyncTimestamp)
设置字段 [DeployedSyncLog::Result; “已找到 ” & ValueCount($CompanyIDs) & “条记录”]
设置字段 [DeployedSyncLog::Table; “Companies”]
提交记录/请求 [跳过数据输入验证; 无对话框]
```

如果您使用的是上面示例中描述的方法，请务必修改变量名和每个表的日志“表”字段设置值。

收集使用间接变量的记录 ID

作为“硬编码”步骤的替代方法来为每个表的 ID 设置一个特定的变量，您可以使用一个循环来收集每个表的 ID 以及每个循环的附加计算变量为每个表创建唯一的变量。如下步骤使用变量为每个表积累 ID 并创建本身的变量名称：

```
设置变量 [$refList; 值: "DeployedCompanies|CompanyID¶
    DeployedContacts|ContactID¶DeployedActivities|ActivityID" ]
循环
    Exit Loop If [
        Let([
        v1 = Substitute(GetValue($tables; 1); "|"; ¶);
        $table = GetValue(v1; 1);
        $field = GetValue(v1; 2);
        $refList = RightValues($refList; ValueCount($refList) - 1)];
        IsEmpty($table)
    ) ]
    设置字段 [ConnectorSystem::gPivot; “”]
    设置字段 [DeployedSyncLog::Action; “定位部署表中已修改记录: ” & $table]
    设置变量 [$IDs; 值: Evaluate("ExecuteSQL(\"Select " & $field & " from " & $table & " where
        zModStamp < ?\"; \"\"; \¶; $SyncTimeStamp)")]
    设置变量 [$countVar; 值: ValueCount($IDs)
    设置变量 [$calculatedVar; 值: Evaluate(
        "Let($" & $table & "IDs = \"" & Substitute($IDs; ¶; "\¶") & "\"; \"\")")]
    设置字段 [DeployedSyncLog::Result; “已定位 “ & $countVar & “ 条记录”]
    设置字段 [DeployedSyncLog::Table; $table]
    提交记录/请求 [跳过数据输入验证; 无对话框]
循环结束
```

当这部分脚本完成时，您将会得到一组变量，这组变量是以含有待同步记录的部署文件中的每个表命名的。在此示例中，变量为 $DeployedCompaniesIDs、$DeployedContactsIDs 和 $DeployedActivitiesIDs。这些变量的值可以通过在 gPivot 字段中依次设置每个键值设定的关系单独解决和处理记录。为使关系图更清晰，该示例中的表摹本以关联表的复数形式命名；但是，使用单数形式命名可以让它们更易阅读且能充分利用上述的计算变量。

交换数据

如果您的同步方案需要在处理记录数据之前首先交换记录数据，那么您的脚本应该只在最开始的时候交换数据。您可以选择在修改记录和即将同步的文件之间转移所有修改过的记录，或者查找符合条件的记录并移动那些“赢得”副本的记录。检查每条记录，移动记录到中间表或者在同步脚本过程中进行修改等过程，将在之后的“处理记录”中描述。

想要简单地移动所有修改过的记录，下面这组脚本步骤能够从部署 Companies 表中将记录导入至托管的中间表中，然后将托管的 Companies 表中被修改的记录隔离，并移至部署的中间表中。之前示例中“执行查找”[ ]或“转到相关记录”[ ] 脚本步骤之后的例子：

```
导入记录 [“SyncConnector.fmp12”]; [从 “DeployedCcompanies”
    导入 “IntermediateHostedCompanies”; 添加；匹配名称；自动输入关闭]
转到布局 [“HostedCompanies” (HostedCompanies”]
执行查找 [Restore; HostedCompanies::zModStamp: [≥ $SyncTimeStamp]]
导入记录 [“SyncConnector.fmp12”]; [从 “HostedCompanies”
    导入 “IntermediateDeployedCompanies”; 添加；匹配名称；自动输入关闭]
```

请记住要包含创建日志记录的脚本步骤（如之前示例中所述的步骤）并将 Action、Result 和 Table 字段适当地设置到日志记录中。

注释：如果任何一个系统允许用户删除记录，那么在每张表中交换最后一次成功同步之前创建的、且对应表中不存在的记录列表是非常有必要的。请参阅“识别待删除记录”获取收集这些主键的方法。这些列表之后将由“正在处理”的脚本用来删除记录。

创建记录的其他方法

另一个导入记录的方法是在查找结果或者记录 ID 列表中循环并使用“pivot”关系创建记录并填充字段。欲知如何通过“pivot”关系创建记录，详见下节“记录创建和修改”。确保所有字段都填写完成后之后提交记录，并且在为下条记录设置字段之前清空 gPivot 字段。

其他数据交换方法

如上文所述，FMP URL 协议允许您发送请求给托管文件来运行脚本。这就提供了另一种将记录移至托管文件的方法。例如，如下形式的 OpenURL 脚本步骤调用“HostedOnServer”托管文件中名为“CreateNewRecord”的脚本，并且为表名称、记录 ID 和公司名称传递变量，以下列字段中的数据填充：

```
打开 URL [ "FMP://{ServerIPAddress}/HostedOnServer.fmp12?
    script=CreateNewRecord&$Table=Companies&$ID=" &
    DeployedCcompanies::CompanyID & "&$Company=" &
    DeployedCompanies::Company]
    [无对话框]
```

使用交换的记录实时操作

如上所述，同步脚本应该在记录被交换或处理托管记录准备就绪时通知托管文件或中间文件。这可能会以在托管文件中创建日志记录的形式表示：

```
设置字段 [SyncConnector::gPivot; “”]
设置字段 [HostedSyncLog::Action; “记录交换完成 (user ” & System::gUserID & “)”]
设置字段 [HostedSyncLog::TimeStampEnd; Get(当前主机时间)]
```

```
提交记录/请求 [跳过数据输入验证; 无对话框]
```

仅就交换记录来看，这些步骤应该在连接器文件同步脚本的结尾。接着，*部署*文件中的同步脚本应该关闭连接器文件和托管文件：

```
设置字段 [System::gPivot; “”]
设置字段 [SyncLog::Action; “关闭连接器文件”]
If [Position(¶ & DatabaseNames & ¶; ¶ & "SyncConnector" & ¶; 1; 1)]
    关闭文件 [“SyncConnector”]
End If
设置字段 [SyncLog::Result; “连接器文件已关闭”]
提交记录/请求 [跳过数据输入验证; 无对话框]
设置字段 [System::gPivot; “”]
设置字段 [SyncLog::Action; “关闭托管文件”]
If [Position(¶ & DatabaseNames & ¶; ¶ & "HostedOnServer" & ¶; 1; 1)]
    关闭文件 [“HostedOnServer”]
End If
设置字段 [SyncLog::Result; “托管文件已关闭”]
提交记录/请求 [跳过数据输入验证; 无对话框]
```

If [ ] 语句用于检测 SyncConnector 和 HostedOnServer 文件是否已经打开。虽然这个检测不是必须的，但是仅在文件可用时进行操作是比较正确的做法。

在部署文件中，该脚本应处理其自身中间表中的记录。这部分脚本以及在主机中处理记录的脚本都应遵循下一节所描述的策略。

处理记录

最后，同步进程来到更新记录这一步。它包括修改现有记录、删除待删除记录和创建新记录。同步脚本应该循环所有记录，比较匹配的修改时间戳（如果存在），并采用业务规则判断哪些记录是待修改记录。循环可以单步调试所查找到的记录或者使用从 $IDs 变量中收集的记录 ID 列表，也可以使用在之前章节“识别部属文件中的同步记录”中介绍的变量。

适当处理托管记录

“适当”处理是一种灵活有效的方法，它能够在不改变上下文的情况下处理多张表内的记录（这是非常有效的，并且为了维持跨越多条记录的事务完整性而允许您在记录集被提交时进行控制）。该方法需要将 SyncConnector“系统”表的表摹本连接到每个部署表和托管表的表摹本，如图 7 所示。任意或所有部署与托管表摹本都能指向中间表而不是活动的表，这取决于您是否想在中间表中创建记录或在活动的系统中处理记录。在 SyncConnector::System 端，全局文本字段必须与每个部署和托管表摹本中的自动输入 ID 字段相关联，并且必须在每个关系的部署或托管端启用“允许通过此关系在本表中创建记录”选项。

使用 $IDs 变量或其他变量中的 ID 列表，随后的脚本步骤从 SyncConnector 文件“系统”表上下文建立与部署文件中一条记录和托管文件中匹配记录的关系。接下来，脚本会比较部署记录的修改时间戳和相对应的托管记录的修改时间戳。如果部署记录的修改比托管记录的修改更新，并且托管记录没有被锁定，那么脚本将为部署文件中已经更改的值修改托管记录中的字段。如果托管记录被锁定，那么脚本会将记录 ID 移至列表底部，并把它分配到一个“锁定的 ID”临时列表，如果在三次尝试后记录仍无法被修改，则将它移至异常处理列表。

另外，如果没有任何匹配的托管记录，下列脚本步骤将通过“允许通过此关系在本表中创建记录”这一关系特性，在主机中创建新的记录（在托管记录获得最后一次成功同步后创建日期的地方）；如果创建记录的时间戳早于最后一次成功同步的时间戳，那么将删除部署文件中的记录（这表明该记录不是自最后一次同步之后创建的新记录，并且已从主机中删除）。

从“设置变量”脚本步骤开始，该步骤建立“识别部署文件中的同步记录”章节介绍的 $IDs 变量：

```
设置变量 [$IDs; 值: ExecuteSQL("Select CompanyID from DeployedCcompanies where
    zModStamp > ?"; ""; ¶; $SyncTimestamp)]
设置字段 [SyncConnector::gPivot; “”]
设置字段 [DeployedSyncLog::Action; “更新托管记录”]
设置变量 [$SyncLogID; 值: SyncConnector::gPivot]
提交记录/请求 [跳过数据输入验证; 无对话框]
Loop
    Exit Loop If [Let([
        $id = GetValue($IDs; 1);
        $IDs = RightValues($IDs; ValueCount($IDs) - 1)];
        IsEmpty($id)
        )]
    设置字段 [SyncConnector::gPivot; $id]
    If [DeployedCompanies::zGenStamp > $SyncTimestamp or
    (HostedCcompanies::zModStamp and DeployedCcompanies::zModStamp >
    HostedCompanies::zModStamp)
        #如果当前记录的创建迟于上一次同步时间戳或有对应的托管记录，则需设置适当的字段（创建一个新记录或修改一个现有记录）。
        设置字段 [HostedSystem::gModOverride; DeployedCompanies::zModStamp]
        设置字段 [HostedCompanies::Company; DeployedCompanies::Company]
        If [Get(LastError) = 301]
            设置变量 [$variables; 值: Let([
            $IDs = $IDs & If(PatternCount($LockedIDs; $id) < 3; Left(¶; not IsEmpty($IDs)) &
                $id);
            $LockedIDs = $LockedIDs & Left(¶; not IsEmpty($LockedIDs)) & $id;
            $ExceptionsList = $ExceptionsList & If(PatternCount($LockedIDs; $id) > 3; Left(¶;
                not IsEmpty($ExceptionsList)) & $id)];
            ""
            )]
        Else
            设置变量 [$modCount; 值: Let([
                    $modCount = If(HostedCompanies::Serial#; 1) + $modCount;
                    $createCount = If(IsEmpty(HostedCompanies::Serial#); 1) + $createCount];
                    "")]
            设置字段 [HostedCompanies::Location; DeployedCompanies::Location]
            提交记录/请求 [跳过数据输入验证; 无对话框]
            设置字段 [HostedSystem::gModOverride; “”]
        End If
    Else If [DeployedCompanies::zGenStamp < $SyncTimeStamp and
    IsEmpty(HostedCompanies::CompanyID]
        #如果当前记录的创建早于上一次同步且未被修改，此外没有对应的托管记录，则需删除部署记录。
        #确认已命名的入口在当前布局中
        转到对象 [对象名称: CompaniesPortal]
```

```
        转到入口行 [选择; 第一个]
        设置字段 [DeployedCompanies::Company; DeployedCompanies::Company]
        If [Get(上一个错误) = 301]
            设置变量 [$variables; 值: Let([
                $IDs = $IDs & If(PatternCount($LockedIDs; $id) < 3; Left(¶; not IsEmpty($IDs)) &
                $id);
                $LockedIDs = $LockedIDs & Left(¶; not IsEmpty($LockedIDs)) & $id;
                $ExceptionsList = $ExceptionsList & If(PatternCount($LockedIDs; $id) > 3;
                    Left(¶; not IsEmpty($ExceptionsList)) & $id)];
                ""
                )]
        Else
        删除入口行 [无对话框]
        设置变量 [$deleteCount; 值: Let($deleteCount = (Get(上一个错误) = 0) + $deleteCount;
            "")]
        End If
    End If
循环结束
设置字段 [SyncConnector::gPivot; $SyncLogID]
设置字段 [DeployedSyncLog::Result; $modCount & " 条记录被修改, " & $createCount & " 条
记录被创建, " & $deleteCount & " 条记录被删除"]
提交记录/请求 [跳过数据输入验证; 无对话框]
```

此示例脚本中，用 Exit Loop If 脚本步骤中的 Let 语句设置变量，而不是使用多个单独的“设置变量”脚本步骤。对于每个循环，第一个 $IDs 变量的值将被删除。当列表中没有剩余的值时（即独立的值 $id 为空），循环将会结束。当然，如果您正在使用某个表中特定的 $ID 变量，请确保在上述步骤中将 $IDs 改成正确的变量名。

> **注释**：设置字段也许可以“间接”实现，当您需要计算被设置的字段名称和值时，使用“设置字段（按名称）[ ]”脚本步骤。这样可以创建更加精简的脚本，因为脚本不需要为每张表设置单独的“设置变量”脚本步骤。

“设置字段”脚本步骤（在对比部署记录修改时间戳和托管修改时间戳的 If 脚本步骤后面）尝试为 Company 字段赋值。如果记录没有被锁定，将打开（并锁定）记录；如果记录被锁定，将返回错误 301（“记录正在被其他用户使用”）。您可以为数值硬编码来修改记录，或者将数值设置到实用工具或系统表的字段中。如果您将数值设置到字段中，该字段当脚本运行时存储变量中的值，那么在整个同步过程中，数字只能被提取一次。

> **注释**：只要脚本是在将每条部署记录的修改时间戳与托管记录的修改时间戳进行比较，您就可以选择用其他方法修改记录。例如，如果一条部署记录的修改时间戳早于托管记录的修改时间戳，您就可以假设托管记录是在部署记录后修改的。如果您的业务规则表明在这种情况下托管记录“胜出”，您可以向同步脚本中添加一些步骤，来将部署记录的字段设定为托管记录的值。设定字段的操作可以被硬编码到特定的表中，或者使用上文提到的“设置字段（按名称）[ ]”来计算。

最后，脚本将检测记录是否应该被删除，前往合适的入口行并删除入口行。请确保将入口添加至布局，每个入口对应您想从中删除记录的表。每个入口需要一个对象名称，您可以在布局模式中使用检查器“定位”选项卡指定对象名称。

要将记录移至中间表，请移除用于测试错误 301 的 If 语句，和与其相连的“设置变量”、Else、End If 脚本步骤，并将斜体字的脚本步骤替换为下列脚本步骤：

```
设置字段 [HostedIntermediateCcompanies::CompanyID;
    DeployedCcompanies::CompanyID]
设置字段 [HostedIntermediateCcompanies::Company;
    DeployedCcompanies::Company]
设置字段 [HostedIntermediateCcompanies::Location;
    DeployedCompanies::Location]
```

此外，“设置字段（按名称）[ ]”步骤可以被用来在中间表中创建记录，以避免分别指定每个目标字段。

识别需要删除的记录

因为上述脚本部分仅检查了自上次同步后被修改的记录，没有考虑到可能已经在托管文件中删除但没有在部署文件中修改的记录。如果您的同步进程分为数据交换和数据处理阶段，您可以配置“交换”部分来传递一个需要删除的记录列表。该列表可以使用下面的“设置变量[ ]”脚本步骤计算出来，之后的步骤将被整合到“处理”部分中。

如果交换步骤和处理步骤在同一个脚本中，下列脚本步骤将识别并删除那些自最后一次同步后没有被修改但已经从相应系统中删除的记录。此示例使用了与上述脚本相同的模型，从连接器文件系统表的上下文中运行，并使用连接器文件中脚本上下文布局上已命名的入口对象来删除记录：

```
设置字段 [SyncConnector::gPivot; “”]
设置字段 [DeployedSyncLog::Action; “识别需删除的记录”]
设置变量 [$SyncLogID; Value: SyncConnector::gPivot]
提交记录/请求 [跳过数据输入验证; 无对话框]
#收集先于上一次同步时间戳创建的、且没有对应托管记录的部署记录 ID。
设置变量 [$IDs; 值: 值: ExecuteSQL("Select CompanyID from DeployedCompanies where
    zGenStamp < ? and not exists (select * from HostedCompanies where
    DeployedCompanies.CompanyID=HostedCompanies.CompanyID)"; ""; ¶;
    $SyncTimeStamp)]
Loop
    Exit Loop If [Let([
        $id = GetValue($IDs; 1);
        $IDs = RightValues($IDs; ValueCount($IDs) - 1)];
        IsEmpty($id)
        )]
    设置字段 [SyncConnector::gPivot; $id]
    #确认已命名的入口在当前布局中
    转到对象 [对象名称: DeployedCompaniesPortal]
    转到入口行 [选择; 第一个]
    设置字段 [DeployedCompanies::Company; DeployedCompanies::Company]
    If [Get(上一个错误) = 301]
        设置变量 [$variables; 值: Let([
            $IDs = $IDs & If(PatternCount($LockedIDs; $id) < 3; Left(¶; not IsEmpty($IDs)) &
                $id);
            $LockedIDs = $LockedIDs & Left(¶; not IsEmpty($LockedIDs)) & $id;
```

```
            $ExceptionsList = $ExceptionsList & If(PatternCount($LockedIDs; $id) > 3; Left(¶;
                not IsEmpty($ExceptionsList)) & $id)];
            ""
            )]
    Else
        删除入口行 [无对话框]
        设置变量 [$deleteCount; 值: Let($deleteCount = (Get(上一个错误) = 0) + $deleteCount;
            "")]
    End If
循环结束
提交记录/请求 [跳过数据输入验证; 无对话框]
设置字段 [SyncConnector::gPivot; $SyncLogID]
设置字段 [DeployedSyncLog::Result; $deleteCount & " 条记录被删除"]
提交记录/请求 [跳过数据输入验证; 无对话框]
```

缩短记录锁定间隔

上述脚本步骤通过“接触”每个字段来完成修改匹配记录的任务。如果需要做少数的修改且被锁定的记录超过了需求，则锁定记录的时间总值可能会增加。为了缩短时间并避免触及无需修改的字段，应收集需要更改的字段列表和字段的值。借助该列表，您可以使用“设置字段（按名称）[ ]”脚本步骤设置仅在列表中指定的字段。

处理部署记录

托管记录被修改后（或修改的部署记录已经被推送至托管文件的中间表后），脚本能够识别托管记录，修改或删除匹配的部署记录，或者在中间表中创建记录。该过程与处理托管文件的过程相同 – 执行“查找”并在记录中循环，或收集主键列表并依次对值进行操作。

注释：如果您选择让脚本执行“查找”并循环记录，可以使用相同的“pivot”关系来添加或更新记录。为了在对应其他文件中的表的表摹本基础上，对布局上下文文件记录进行创建和修改，至少需要 ConnectorSystem 表中的一条记录 。若要修改现有记录，必须首先将相关的键值写入 ConnectorSystem 表的全局键字段（“gPivot”），来建立当前上下文和目标表摹本间的关系路径。

主机上的预设脚本

如果您在数据交换后处理记录，可以使用上述脚本步骤的修改版本来处理中间表。您可以在 FileMaker Server 或独立的机器人系统中设置一个适合您方案的时间间隔脚本来处理中间表的记录。例如，如果用户实现全天候同步，您可能想要每隔几分钟就运行一次该脚本。如果用户通常在早晨或晚上同步，该脚本可能计划在这些时间段中运行。

该脚本应该遵循与连接器文件中同步脚本相似的格式 – 识别适当的记录、使其与活动表中的副本进行循环对比，并根据需要进行更新。虽然您设计该脚本可能仅为了更新活动记录，但如果托管记录在部署记录被移至中间表后被修改，我们建议您对比来自中间表记录的修改时间戳和托管记录的修改时间戳。

部署文件的处理

同样，如果您在数据交换后处理记录，可以使用上述脚本步骤的修改版本来处理部署表。在记录被交换并且连接器文件和托管文件被关闭后，脚本可以继续运行或者调用一个单独的脚本来

处理来自中间表的记录。在这种情况下，gModOverride 机制应该用来确保每条记录的 zModStamp 被设定为存储中间表记录数据的修改时间戳字段（请记住中间表中没有任何应被设置为自动输入的字段，但是应该利用同步脚本填入源记录中的数据）。

删除中间记录

在您处理的脚本需要更新状态字段以标记被处理的记录时，或者向存档传输记录时，中间记录可能被不同的用户作为更改的历史记录被保留。在其他情况下，中间记录应该在处理完毕后立即删除。

断开和辅助

在记录进行完交换和处理（如果这也是同步的一部分）之后，就应该创建日志记录来记载交换或者同步的结果：

设置变量 [$EndSyncTimeStamp; 值: Get(当前系统时间)]
设置字段 [SyncConnector::gPivot; $HostSyncLogID]
设置字段 [HostedSyncLog::Result; “同步执行人: “ & DeployedSystem::UserID & “ 完成于: ” & $ EndSyncTimeStamp]
提交记录/请求 [跳过数据输入验证; 无对话框]
设置字段 [SyncConnector::gPivot; “”]
设置字段 [DeployedSyncLog::Action; “同步完成于 ” & $ EndSyncTimeStamp]
提交记录/请求 [跳过数据输入验证; 无对话框]
退出脚本 [Result: $CurrentSyncTimeStamp]

最后，正如在“交换记录”章节中所提到的，一旦连接器文件中的同步脚本执行完成，被*部署*的文件应该关闭连接器文件和托管文件，然后通知用户同步已经完成。

注释：在给用户的最后通知中，您可以选择包含被删除、修改、创建和/或未更新的记录信息。如果是这样的话，确保在连接器文件脚本的适当步骤收集信息，比如，在作为脚本结果传回至调用脚本的变量中，或者在全局字段或日志记录中。

总结

本文描述的方案旨在提供指导，为您自己的同步系统奠定了基础并为您采取的方法提供思路。同步的概念非常重要但又非常简单 – 改变一组或多组记录使它们与其他组的记录保持一致。实际的过程可能会细化和复杂，这取决于活动记录和中间记录间是否会发生对比和修改。然而，复杂性问题可以通过良好的规划，以及正确使用 FileMaker 的功能和特性创建强大稳定的同步系统来解决。

资源

FileMaker TechNet: http://www.filemaker.com/technet/
FileMaker 培训教程: http://www.filemaker.com/support/training/fts.html
FileMaker 论坛: http://forums.filemaker.com/
FileMaker 顾问: http://developer.filemaker.com/search/
GoZync 商务解决方案: http://www.gozync.com
MirrorSync 商务解决方案: http://www.360works.com/mirrorsync

关于作者

凯瑟琳·卢瑟尔（Katherine Russell）是 NightWing 公司（NightWing Enterprises）在澳大利亚墨尔本的高级顾问兼团队领导。想了解更多关于凯瑟琳的信息，请参阅：
https://fmdev.filemaker.com/people/katherinerussell