按需加载工作表

此功能是0.7.1版本中的新功能，由函数的 `on_demand` 参数控制 `open_workbook()` ，允许通过仅加载调用者感兴趣的那些工作表来节省内存和时间，并在不再需要时释放工作表。

`on_demand=False` **（默认）：**

没变。 `open_workbook()` 加载全局数据和所有工作表，释放不再需要的资源（主要是 包含工作簿流的对象 `str` 或 `mmap.mmap` 对象），并返回。

`on_demand=True` **和BIFF版本<5.0：**

发出警告消息， `on_demand` 记录为 `False` ，并遵循旧过程。

`on_demand=True` **和BIFF版本> = 5.0：**

`open_workbook()` 加载全局数据并返回而不释放资源。在这个阶段，有关表格的唯一信息是 `Book.nsheets` 和 `Book.sheet_names()` 。

`Book.sheet_by_name()` `Book.sheet_by_index()` 如果尚未加载，将加载请求的工作表。

`Book.sheets()` 将加载所有卸载的工作表。

调用者可以 `Book.unload_sheet()` 在完成工作表时通过调用来节省内存 。无论状态如何，这都适用 `on_demand` 。

调用者可以通过调用 `Book.sheet_by_name()` 或重新加载已卸载的表单 `Book.sheet_by_index()` ，除非已释放所需的资源（当 `on_demand` 错误时将自动发生）。这是唯一会引发异常的情况。

调用者可以使用查询工作表的状态 `Book.sheet_loaded()` 。

`Book.release_resources()` 可用于保存内存并关闭任何内存映射文件，然后再继续检查已加载的工作表。资源一旦发布，就无法再加载资料。

当按需使用时，建议 `Book.release_resources()` 始终调用，即使在您自己的代码中引发异常也是如此; 否则，如果输入文件已经过内存映射，则 `mmap.mmap` 对象将不会关闭，并且在Python进程终止之前您将无法访问物理文件。这可以通过 `Book.release_resources()` 在try / finally块的finally部分中显式调用来完成。

Book对象也是一个上下文管理器，因此您可以将代码包装在一个 `with` 语句中，以确保关闭底层资源。