FOLIO 中的模块化：原理、技术和工具*

迈克尔·泰勒（Index Data、布里斯托尔大学）
周江纯[1] 张春景[1] 胡小菁[2]（译）（1 上海图书馆 2 华东师范大学）

摘 要 从一开始，FOLIO 就被设想为一个由许多独立且相互依赖的模块组成的服务平台，而不是一个传统的 ILS（Integrated Library System，图书馆集成系统）。这一设想成了基于相关模块构建 ILS 或其他图书馆软件的基础。FOLIO 的这种模块化的构想降低了参与的门槛，因而对图书馆社区产生了十分强大的吸引力：每个图书馆都可以由自己或者雇佣开发人员来创建满足自己需求的模块，或者资助对广泛社区有用的模块——以上所有方式都不需要从权威机构处获得“许可”。FOLIO 的技术设计深受模块化需求的影响，包括标准规范的建立和强调对机器可读 API 的描述。虽然 FOLIO 的模块化设计已被证明是优越的，但同时也带来了一些困难，包括跨模块搜索和数据的一致性。已有一些约定来解决这些困难，另一些约定正在细化的过程中。随着建立于 FOLIO 平台之上的 ILS 的发展和成熟，以及其他应用程序套件的创建，不走单体系统快速开发成功的老路和保持模块化的愿景仍然是至关重要的，因为正是模块化成功地让 FOLIO 走到了今天。

关键词 FOLIO 服务平台 社区 模块化 软件工程

Modularity in FOLIO: Principles, Techniques and Tools

Michael P. Taylor (Index Data and University of Bristol)
Zhou Jiangchun[1] , Zhang Chunjing[1] , Hu XiaoJing[2] Trans. (1 Shanghai Library; 2 East China Normal University)

Abstract From its earliest inception, FOLIO was conceived not as an ILS (Integrated Library System), but as a true Services Platform, composed of many independent but interdependent modules, and forming a foundation on which an ILS or other library software could be built out of relevant modules. This vision of modularity is crucial to FOLIO's appeal to the library community, because it lowers the bar to participation: individual libraries may create modules that meet their needs, or hire developers to do so, or contribute to funding modules that will be of use to a broader community — all without needing "permission" from a central authority. The technical design of FOLIO is deeply influenced by the requirements of modularity, with the establishment of standard specifications and an emphasis on machine-readable API descriptions. While FOLIO's modular design has proved advantageous, it also introduces difficulties, including cross-module searching and data consistency. Some conventions have been established to address these difficulties, and others are in the process of crystallizing. As the ILS built on FOLIO's platform grows and matures, and as other application suites are built on it, it remains crucial to resist the shortcuts that monolithic systems can benefit from and retain the vision of modularity that has so successfully brought FOLIO this far.

Keywords FOLIO, Services Platform, Community, Modularity, Software Engineering

1 文章简介：FOLIO 理念

FOLIO① 是“The Future Of Libraries Is Open”（图书馆的未来是开放的）的首字母缩写，它通常被认为是一个开源的 ILS。虽然这

* 本文根据《国际图书馆学报》（International Journal of Librarianship, 简称 IJoL）2021 年 12 月 FOLIO 特刊发表的英文论文做全文翻译。原文出处：Taylor, M. (2021). Modularity in FOLIO: Principles, Techniques and Tools. International Journal of Librarianship, 6(2):3–12. https://doi.org/10.23974/ijol.2021.vol6.2.174.

① http://www.folio.org/

并非不正确，但这只是对 FOLIO 本身认识理解的一种观点。该项目是在理想主义和实用主义的考验中构思出来的，旨在创建一个全球图书馆社区自主拥有并能施加影响的模块化平台，从而共同构建许多面向图书馆的应用程序。虽然 ILS 是可以建立在这样一个平台上的应用之一，也是一个最具商业可见性的应用，但这远不是 FOLIO 唯一的可能性。事实上，在 FOLIO 基金会发展的早期，它被半正式地认为是一个 DLS（Disintegrated Library System）——一个去集成图书馆系统——因为它的目标是每个组件都独立存在。而集成则来自模块之间的相互依赖，并且这些模块需基于明确规定的接口。

因此，本文对两者做了区分：一个是 FOLIO 平台，下面将详细描述；另一个是建立于其上的 FOLIO ILS。前者曾被称为 LSP（Library Services Platform，图书馆服务平台），但这个术语似乎是单独生造的，指提供 web 服务 API 的 ILS（Breeding，2015），所以现在为了防止混淆而避免使用这个术语。

这种模块化的、基于组件的理念已经深刻地体现在 FOLIO 的各个项目之中。它源于这样一种愿望：让任何感兴趣的一方都能够为任何模块做出贡献，以此来建立整体的力量和灵活性，从而避免出现单一一个负责的机构承担所有的变化和附加责任的典型 ILS 情况。从一开始，我们的意图就是，比如一个 FOLIO 图书馆需要一个预订房间的模块，他们可以自己创建这个模块，并很轻松地将其连接到系统之中，也可以向其他 FOLIO 图书馆提供这个预订房间的模块。模块也可以被已有接口的新的实现方法所替代：例如一个图书馆使用一个扣分系统代替原有的罚款系统来进行逾期惩罚。这样，图书馆可以用他们自己的、与其他平台组件 API 期望匹配的模块来替换相关模块。FOLIO 更广泛的目标是创建一个民主公地，不仅在组织上不同利益相关者可以对高层决策做出贡献，而且任何人都可以因自己的特定需求而实际创建系统中缺失的那些部分（或聘请开发人员做），并将他们回馈给更广泛的社区。

必须指出的是，优先选择哪些模块进行开发是以在 FOLIO 平台上创建一个可运行的 ILS 的需求为指导的，因此现在已经为典藏管理、流通、用户管理等创建了模块。（下面是 FOLIO 平台用于非 ILS 应用的例子。）为了与 FOLIO 对社区的关注保持一致，早期 ILS 发展的优先次序是由一个广泛的社区推动的，这个社区代表了许多国家和机构不同大小的图书馆，并在社区内达成了广泛的共识。同样，每个核心 ILS 模块的设计也来自代表许多不同机构的专题事务专家之间的合作讨论，以确保核心 ILS 模块能满足不同种类图书馆的需求（FOLIO, 2021）。

2 FOLIO 的灵活架构

FOLIO 有时被描述为由微服务组成，但这其实是一个误导性的观点。在微服务体系结构中，一组被定义好的异构服务在 web 中相互调用，以执行截然不同的任务，包括系统逻辑和业务逻辑。相比之下，在 FOLIO 中有任意数量的模块，可能有许多不同的来源，所有这些模块都以符合一组标准的方式实现应用程序领域的各个功能。模块之间的依赖关系可以存在，但是一般被表示为同行之间的 Web 服务应用程序编程接口（Web Service Application Programming Interface，WSAPI）调用，并且必须以机器可读的形式进行描述（见下文）。

有两种模块，分别是前端（UI）模块和后端模块。一个“app”通常由一个 UI 模块和一个相关的后端模块组成，前者主要或专门调用后者：例如，UI 模块 UI –courses 和后端模块 mod–courses 一起组成了 Course Reserves（教学参考书，下同）应用程序。虽然这个例子表明前端和后端模块之间是一对一的对应关系，但实际上一个前端模块可以调用许多后端模块。后端模块还可以联系其他后端模块来实现其部分功能。

由于 UI 和后端模块运行在不同的环境之中，因而它们的规则是截然不同的。本文将主要关注后端模块，但在详细考虑这些模块之前，有必要简要考虑一下 UI 模块是如何工作的。

3 STRIPES[①] 框架和 UI 模块

每个 UI 模块都是在 JavaScript 中以 NPM 包[②] 的形式实现的（匿名，2020），使用 React 框架（Hunt, 2013），满足特定的需求并依赖于特定的条款。它运行在 FOLIO 的 web 应用框架 Stripes（Taylor et al., 2017）中，Stripes 是用于具体部署时 UI 模块集合的容器，为它们提供访问后端数据和实现国际化等方面的工具，并在符合 FOLIO 用户体验（User Experience，UX）指南的标准化布局中呈现。为了实现 Stripes 模块（也就是 FOLIO UI 模块）的功能，NPM 包必须满足以下要求：

- 它的程序包文件必须包含一个 Stripes 部分。
- 这个部分必须包含一个 actsAs 条目，一个数组来指定这个模块在 Stripes 中可以扮演什么角色：app、设置、插件、处理程序等。Stripes 使用它来了解如何调用该模块。
- Stripes 部分还必须指定模块使用的后端接口和版本（见下文）。这是对其后端需求的声明。
- 程序包文件可以声明分配给模块用户的权限，以允许访问模块功能的各个方面。这些权限可能包括后端模块定义的子权限。
- 翻译目录必须包含定义翻译标记及其表达方式的文件。该文件至少用英语表示，通常也用很多其他语言表示。
- 最高级别的源文件必须导出一个 React 组件，Stripes 会根据模块所执行的角色以各种不同的方式调用这些组件。

一般来说，UI 模块并不直接依赖于其他 UI 模块（尽管它们可能间接地依赖于具有插件或处理程序角色的 UI 模块）。相反，它们依赖于后端模块。

考虑到对 UI 模块的简要概述，从这里开始，本文将使用术语“module（模块）”来表示后端模块，下文将更详细地讨论这些模块。

4 什么是 FOLIO 模块?

就像 FOLIO UI 模块是满足特定需求的 JavaScript NPM 包一样，FOLIO 后端模块也是满足不同需求的程序。与 UI 模块不同，后端模块不必用任何特定的语言编写：这是因为它们与 UI 模块不同，并不作为程序包的一部分运行，而是作为单独的进程运行。FOLIO 模块可以用任何语言和任何工具编写，而且确实存在用 Java、Groovy、JavaScript、Perl 和其他编程语言编写的模块。目前，社区的工具（见下文）对 Java 的支持是最强的，因此出于实用主义的考虑，大多数模块都是用 Java 编写的。

所有对 folio 后端模块的访问都由一个被称为 Okapi[③]（OK API 的首字母缩略词）的 API 网关调节（Dickmeiss 等人，2016）。在典型的部署情况中，会通过阻塞除 Okapi 网关公开的端口之外的所有网络端口来强制执行。除此之外，Okapi 管理会话、处理用户身份、维护运行在同一网关上的多个图书馆之间的租户边界，并强制执行 FOLIO 权限模型，以禁止未获得对应权限的用户访问后端模块：Okapi 在后端模块监测到之前就拒绝了这些用户的请求。

对于一个作为 FOLIO 模块运行的程序，它必须通过接受和响应 WSAPI 请求与外界进行交互，并且它实现的请求必须在模块描述符中以机器可读的形式进行描述。为了使一个模块在 FOLIO 的运行实例中可用，它的模块描述符和一些部署信息必须发布到 Okapi 网关。

模块描述符是一个 JSON 文件（格式由 JSON Schema 定义），它描述了模块的功能和限制，以便 Okapi 能够适当地调节对它的访问。它还规定其他事项：

- 该模块的唯一标识符
- 模块定义的权限列表，包括其中含有哪些子权限。
- 该模块所需的接口列表（见下文）
- 该模块提供的接口列表（见下文），

① 译者注：Stripes 是一个使用最新的 Java 技术来构建 Web 应用的展现框架。
② 译者注：NPM 是 JavaScript 的包管理工具。
③ 译者注：Okapi 是 FOLIO 的多租户 API 网关。

以及这些接口中所包含的 WSAPI 操作和这些操作所需权限的规范。

依赖关系是通过接口而不是实施操作来确定的：例如，mod-courses（实现 Course Reserves 应用程序的后端）不依赖于 mod-inventory-storage 模块，而是依赖于 item-storage 接口。这一要求可以由任何在兼容版本上执行指定接口的模块来实现。目前，这可能是 mod-inventory-storage，但 FOLIO 架构的定义对于 Course Reserves 甚至没有任何要求。其他人可以执行 item-storage 接口的替代典藏模块，FOLIO 管理员可以选择部署该模块，而不是标准实施，Course Reserves 程序会乐于使用该模块。正如 Jason Skomorowski 所观察到的，接口可以让你更换一部分东西，而不必重新发明创造整个东西。

当模块描述符指定模块提供的接口时，它包含了关于哪些 HTTP 方法支持哪些路径的信息（GET、PUT、POST 等），以及每个请求需要哪些权限。Okapi 使用此信息将请求路由到适当的模块，并拒绝那些没有必要权限的请求。描述符本质上是关于模块的协议。

谁能创建一个 FOLIO 模块？答案是任何有技术知识的人，没有人需要获得准许。如果一个模块满足上述要求，那么它就可以被嵌入到一个正在运行的 FOLIO 系统中，并在该生态系统中发挥它的作用。对于 FOLIO 系统操作员来说，决定支持哪些模块将是一个策略问题，但是任何托管自己 FOLIO 系统的组织都可以自由地添加它所选择的任何模块。

5 FOLIO 模块的工具支持

在 FOLIO 中，所有从 UI 到后端模块以及后端模块之间的通信都是通过基于 HTTP 的 web 服务方式实现的。按照惯例，这些 web 服务都采用 RESTful 架构[①]，即涉及可寻址实体的状态（Fielding，2000），并以 JSON 格式返回响应。然而，Okapi 并没有强制执行这些约定，它在这些问题上是不可知的，并且很乐于支持 RPC 风格的 web 服务、XML 格式的响应等。（Okapi 本身由基于 RESTful JSON 的 WSAPIs 控制，它提供用嵌入或删除模块等操作。）

模块描述符是模块唯一需要提供的自我描述：它指定了支持哪些 HTTP 路径和方法，但没有涉及到 URL 查询参数、请求和响应的格式等方面。因此，模块通常会以更详细的机器可读的形式来描述 web 服务。服务本身通常在 RAML[②]（Heidenreich 等人，2016 年）或 OpenAPI[③]（Miller 等人，2020 年）中进行描述；JSON 的请求和响应使用 JSON Schema 进行描述（Hutton 等人，2020 年）。

框架可以帮助 FOLIO 模块开发人员从使用这些机器可读的服务描述中实现最大价值化。迄今为止使用最广泛的是 RMB（RAML 模块构建器），它将模块的 RAML 文件和相关的 JSON Schema 作为真理之源，并使用它们自动生成的 Java 代码，将描述的对象持续写入到 PostgreSQL 数据库。然后，模块开发者可以在生成的代码中添加业务逻辑。当然，使用像 RMB 这样的框架并不是强制的——首先，它不能直接用于由 Java 以外的语言编写的模块，但是对于许多模块，特别是那些专注于创建、更新、获取和搜索各种对象的模块，它提供了有力的支持，并大大减少了需要编写代码的数量。

RAML 和 JSON Schema 也被另一个重要的工具所使用：一个自动文档生成器使用这些文件在 https://dev.folio.org/reference/api/ 上创建 HTML 开发人员指南。当 JSON Schema 包含了有用的描述字段、示例和介绍性文本时，自动生成的文档可以为开发人员提供使用模块所需的所有信息：参见示例 https://s3.amazonaws.com/foliodocs/api/mod-courses/p/courses.html

6 挑战

虽然 FOLIO 所采用的模块化被证实有许多好处，但它同时也带来了一些挑战。对于从单体系统如 Koha[④] 转到 FOLIO 的开发人员，

① 译者注：RESTful 一种网络应用程序的设计风格和开发方式，基于 HTTP，可以使用 JSON 格式定义。
② 译者注：RAML 是 RESTful API 的建模语言。
③ 译者注：OpenAPI 是开放网站的应用程序接口。
④ 译者注：Koha 是一个开源的图书馆集成系统。

他们不能简单地对一个包含所有 ILS 功能的数据库编写一个大型 SQL 查询，而必须使用由负责不同功能领域的模块定义的 WSAPIs。这是一种两难的情况。虽然单一的大型 SQL 查询方法对快速完成任务很有吸引力，但在使用它的大型系统中，它总是会导致维护困难，一些对范式微小的更改甚至会影响到几乎不相关的模块中的代码。

使用 FOLIO 模块的一个更大的困难是跨模块查询。首先考虑一个模块内的情况：搜索名称满足某些条件的用户，其返回结果应该包括每个用户所属的读者组的名称。由低级 mod-users[①] 模块返回的用户对象只包含一个读者组 ID。如果想要显示读者组名称的 UI，则必须在一个单独的读者组 WSAPI（也是由 mod-users 提供的）中查找，以确定具有该标识符的组名称。但是业务逻辑模块 mod-users-bl 呈现相同用户数据的高层视图，并返回复合用户对象，其中包括扩展的读者组对象（以及其他信息，比如每个用户拥有的权限集）。因为用户注册和读者组设置都是由同一个底层模块维护，所以模块可以在连接相关数据库表的单个 SQL 查询中获取关于用户及其读者组的信息。

但是每当需要连接跨数据集维护不同的模块时，SQL 是不能使用的，因为这将违反每个模块负责自己的数据及其表示的限定。一般来说，它甚至不能假设两个模块使用相同的数据库或基于相同的 RDBMS[②]。例如流通模块 mod-circulation[③]，其中表示借出的单件包含单件的 ID 和借用用户的 ID。如果想看到单件的名称和借用者用户名，必须在各自的 WSAPI 查询，因为流通业务模块不可能在一个连接所有表的 SQL 语句中获取所有相关信息，流通模块只了解自己的数据库表。

当然，一个更高级别的模块仍然可以返回一个借出记录，其中包括在他们各自的 WSAPI 中查找的单件件和用户信息——这就是 mod-circulation 实际上所做的。但是，获取借出信息不仅涉及 SQL 查询，还涉及对其他模块的两次调用，每个模块都进行自己的 SQL 查询。这是为了实现模块化而必须付出的代价，而回报则是灵活性。一个新的用户管理模块（可能基于 LDAP 或 NCIP 的用户注册功能）可以代替 mod-users，而流通模块将像以往一样继续工作，不需要做任何改变。

mod-graphql 为跨模块搜索问题提供了一个更为通用的解决方案。该模块没有自己的存储，但可以由其他模块提供的 WSAPIs 之间的链接的及其可读的描述进行配置，并知道如何通过搜索多个 WSAPI 并将结果合并在一起，来响应以 GraphQL[④]（Facebook，2018）表达的查询。这有很多应用：例如，FOLIO Z39.50 服务器使用 mod-graphql 来搜索某个实例（书目记录）及其馆藏记录和这些馆藏中所包含的单件，返回包含所有这些信息经强化的书目记录。对于与主查询中相同或不同的模块中实现的子查询（以及子子查询和子子查询），这种方法同样适用。

最后必须承认的是，设置和管理一个完整的 FOLIO 系统比运行一个典型的单体 ILS 更具挑战。这是因为需要启动 Okapi 本身，加载它与所有相关模块的信息，并安排启动和停止这些模块。然而，这种复杂性是为了换取巨大的灵活性所付出的代价。它允许各种不同的编排方案，使用如 Kuderker 群或 Amazon 的弹性容器服务工具，并允许大量使用的模块、冗余和故障切换。

7 FOLIO 模块的经验

既然 FOLIO 开发面临着这些非常现实的挑战，那为什么还要坚持使用模块化架构呢？因为去中心化方法的优点不仅反映了实际需求，而且还反映了 FOLIO 项目的基本理念。就像图书馆本身是关于知识获取的民主化一样，模块也是关于图书馆技术的民主化，允许实际用户决定如何开发和在哪里开发，他们想要优先考虑哪些项目，他们想要雇佣或签约哪

① 译者注：mod-users，即用户管理模块后端，下同。
② 译者注：RDBMS，Relational Database Management System，关系数据库管理系统。
③ 译者注：mod-circulation，即流通模块的后端，下同。
④ 译者注：GraphQL 是一种针对图状数据进行查询的查询语言。

些技术人员来实现他们的目标，甚至模块之间交互什么能最好地满足它们的需求。

这其中的一个例子就是 Course Reserves 模块。这是一个特定的图书馆联盟的需求——芬威图书馆组织（Fenway Library Organization）——一个 Index Data 公司的客户。但是 FOLIO 路线图没有包括 Course Reserves 模块，因为它并不拥有足够高的优先级。在传统的单体 ILS 中，事情的结局会是这样的：产品供应商的开发团队将遵循技术路线图，而 Course Reserves 模块将不会被添加到 ILS 中。但在 FOLIO 中，Index Data 公司单独设计和实现一个 Course Reserves 模块（或者更准确地说，是一对模块：UI 和后端，一起组成一个应用程序）是一件相对简单的事情。这个模块与现有的 FOLIO 模块 WSAPIs 进行交互，尤其是那些由 Inventory[①] 模块提供的单件、馆藏地、借出类型等信息。这样做不需要对 Inventory 模块做任何更改，而只需要使用现有的接口，并声明它对这些接口的依赖。Inventory 模块甚至不知道 Course Reserves 模块的存在：从它的角度来看，Course Reserves 模块只是一个普通的客户端。在证明了它的价值之后，Course Reserves 模块现在已经被接受为 FOLIO 模块的核心集，这使得任何使用 FOLIO 的机构都可以轻松地安装使用。

关于 FOLIO 的模块化如何解决单体 ILS 难题的一个更重要的例子是 ReShare 项目，这是一个管理馆际互借（Inter-Library Loan，ILL）的系统。最近，学术图书馆合作与创新伙伴关系（the Partnership for Academic Library Collaboration & Innovation，PALCI）的 50 多家图书馆和 ConnectNY（CNY）的大约 15 家图书馆成功地实施了这一计划。ReShare 是作为一组 FOLIO 模块实现的，其中第一个用于管理 ILL 请求，第二个用于管理传入请求的实现，第三个用于目录管理等，所有这些模块都独立于 FOLIO ILS 模块开发，但都利用了 FOLIO 平台基础设施。ReShare 应用程序套件不仅构建在 Stripes 框架、Okapi API 网关和权限模型上，还构建在应用程序级 FOLIO 模块上，包括用户管理、租户设置和开发人员设置以及字段。此外，ReShare 的联盟配置包括一个共享索引，该索引是通过 FOLIO Inventory 模块实现的。

通过使用 FOLIO 平台和其他 FOLIO 模块，ReShare 应用程序套件的开发速度得到了大大加快。与此同时，模块化架构意味着 ReShare 的开发已经能够从 FOLIO ILS 的开发中分离出来，由一组开发人员驱动，其中只有部分人参与了 FOLIO。在此过程中，ReShare 开发者对 FOLIO 工具包的某些部分进行了改进，例如，可重用的 React 组件间接地帮助充实了 ILS 应用程序，这是一个意外的收获。

FOLIO 社区计划开发更多的应用程序套件和 ReShare：也许是一个内容管理系统，一个机构存储库，一个采购系统，或者其他什么东西。还可以创建应用程序套件以形成与其他系统的桥梁，如 ArchivesSpace 或 Canvas 学习管理系统。令人兴奋的是，FOLIO 社区的模块化架构和民主的社区氛围意味着这些贡献可以来自任何地方——甚至是目前与 FOLIO 无关的组织。衡量软件工具的一个标准是它能在多大程度上被应用到创造者没有想到的用途之上。FOLIO 的理念都是鼓励这种利用的。

8 结论

FOLIO 项目的每个方面都反映了它的社区起源和民主原则。需要特别指出的是，模块化架构不仅仅是一种战术选择（尽管其采用大大有助于消除单体系统中复杂性的指数增长）。相反，FOLIO 的模块系统是项目目标、态度和理念的技术表现。

这一选择是有一定代价的。这让 FOLIO ILS 的初始版本以一组模块的形式运行需要一些时间，但这些代价已经通过快速设计、原型和开发新添加和扩展的能力得到了充分回报。因此，FOLIO 现在实现新功能的速度比其他同等成熟的 ILS 更快。项目的现有状态只是一个开始，任何人都可以影响它未来的走向。

① Inventory，典藏，下同。

参考文献

[1] About npm[EB/OL].[2021-9-13].https://docs.npmjs.com/about-npm

[2] Breeding M. Library services platforms: A maturing genre of products[M]. American Library Association, 2015.

[3] Okapi guide and reference[EB/OL].[2021-8-24]. https://github.com/folio-org/okapi/blob/master/doc/guide.md

[4] GraphQL, June 2018 Edition[EB/OL].[2021-8-27]. https://spec.graphql.org/June2018/

[5] Fielding R. CHAPTER 5: Representational State Transfer (REST). 2000[J]. https://www. ics. uci. edu/~ fielding/pubs/dissertation/rest_arch_style. htm. Accessed, 2020: 6-25.

[6] Our community[EB/OL].[2021-9-14].https://www.folio.org/community/

[7] Heidenreich C, Hahn K, Stoikovitch J. RAML Version 1.0: RESTful API modeling language[J]. Retrieved August, 2016, 26: 2021.

[8] Why did we build React?[EB/OL].[2021-9-13]. https://reactjs.org/blog/2013/06/05/why-react.html

[9] JSON schema: A media type for describing JSON documents, draft-bhutton-json-schema-00[EB/OL].[2021-8-26].https://json-schema.org/draft/2020-12/json-schema-core.html

[10] OpenAPI specification, version 3.1.0. 2020[EB/OL].[2021-8-26].https://json-schema.org/draft/2020-12/json-schema-core.html

[11] The stripes module developer's guide[EB/OL]. [2021-8-24].https://github.com/folioorg/stripes/blob/master/doc/dev-guide.md

迈克尔·泰勒 Index Data 公司，软件专家；英国布里斯托尔大学恐龙古生物学，研究员；制定 FOLIO 技术设计核心团队的工程师。他深入参与了 FOLIO 的用户界面工具、GraphQL 服务、Z39.50 服务器以及其他对平台应用程序中 ReShare 应用程序套件的实现过程。在业余时间，他描述并命名了多种蜥脚类恐龙，包括色诺波塞顿(“外星地震之神”)和野马(“雷腿”)。

周江纯 上海图书馆（上海科学技术情报研究所），馆员。 上海 200031

张春景 女，上海图书馆（上海科学技术情报研究所），研究员。 上海 200031

胡小菁 女，华东师范大学图书馆，研究馆员。 上海 200062

（上接第 16 页）

黛布拉·豪厄尔 康奈尔大学图书馆 IT 运营主任，拥有超过 20 年的信息技术全领域项目管理和战略指导经验，在促进、变革、团建和领导方面有着广泛的背景。黛布拉于 2006 年获得康奈尔大学工业与劳动关系硕士学位，她是经过认证的项目管理专业人士(PMP)，是康奈尔大学 FOLIO 实施项目经理，同时也是国防部长办公室认证的主培训师，并担任纽约州美国教育委员会妇女网络董事会的信息官。

嵇 婷 女，上海图书馆(上海科学技术情报研究所)，副研究馆员。 上海 200031

张春景 女，上海图书馆（上海科学技术情报研究所），研究员。 上海 200031

胡小菁 女，华东师范大学图书馆，研究馆员。 上海 200062

（上接第 44 页）

[3] Newman S. Building microservices[M]. O'Reilly Media, Inc., 2021.

[4] Why libraries need to get with apps and APIs. [EB/OL]. [2021-12-21].https://opensource.com/article/18/6/folio.

刘铁威 中国 FOLIO 产品经理（PO）。在加入 FOLIO 项目之前，她曾在美国和中国的多家学术图书馆担任数字存储库、编目和元数据、采访和公共服务馆员。E-mail:twliu@utexas.edu