一种新型的构建数字内容管理基础设施的微服务方法

摘要

为了更好的满足加利福尼亚大学(UC)多元化的需求，加利福尼亚数字图书馆UC内容管理中心正在重新构建数字内容管理中心的基础设施通过将功能转化为一组细粒度的、独立的，但是彼此相连的微服务。因为每个服务都是小而且独立的，所以它们更容易开发、部署、维护和扩展；同时，复杂的内容管理功能得以通过将原子的服务组合的方式实现。出现的方法强调内容的持续性而不是整个内容管理系统，因此规范的文档不会过度的耦合到特定的技术环境。这个结论在一个内容管理的环境中的范围是全面的，但是尽管技术和用户期望会有颠覆性的变化，在局部的策略实际上还是灵活的且可持续的。

介绍

信息技术和资产已经成为加利福尼亚大学(UC)的教育任务里必不可少的一部分。UC社区的成员在教学、学习、研究过程中会产生和利用大量的各种各样的数字资产。这些资产代表了大学的知识资本；他们有固有的持久价值并且需要自习的管理以确保他们可以被将来的学者们使用。在加利福尼亚数字中心(CDL)UC内容管理中心(UC3)中有使命去确保大学数字资产的长期的可用性。

UC3在数字内容管理的使命越来越大，一组政策和实践关注于现在以及不确定的将来保留和增加数字内容的可信任性(Abbott, 2008)。一般来说，保存和使用被认为是不同的活动。然而，他们应该被认为是互补的功能：保存关注于能超出预期的时间使用(Rusbridge, 2008)。因此，内容管理中心的管理和丰富是数字资产的生命周期中的不断进行的过程(Higgins, 2008)。虽然内容管理的成功不仅仅是一项技术工作，例如高度依赖重要的人力资源、分析和决策，但是在一个健壮的基础设施内，越来越有效的管理有价值的数字内容是必要的基础。

内容管理基础设施

因为中央系统提供服务给10个UC校区，所以UC3承担数字内容不断地增加的责任。此外，这些内容常常在新的环境中被使用和重新改变用途。因此，UC3的编程准则是在总体范围内提供一个内容管理环境，但是局部规则和实现是灵活的，在技术和用户期望上必然会发生颠覆性的变化，并且在时间跨度上的内容管理的实现是一个中继器。(Janée, Frew, & Moore, 2008)。

为了达到这个目的，UC3反对将内容管理仓库作为中心地点(Abrams, Cruse, & Kunze, 2008)。新的UC3实现数字内容管理基础设施的方法是基于将必要的方法转移到一组独立的但是彼此协作的微服务，这包含内容管理的价值和策略。因为每个服务都很小，他们都是易于开发、部署、维护、扩展的(Denning, Gunderson, & Hayes-Roth, 2008)。同样重要的是，由于对任何给定服务的投资和承诺程度都很小，所以在超出他们的可用性的时候更容易被更换。尽管单个服务的作用很狭小，但是可以通过对原子的服务有战略的组合来实现内容管理的复杂功能(Fisher, 2006)。

微服务可以被部署在技术和管理最有意义的环境中。尽管UC3会使用微服务作为集中管理的基石，但这些服务也可以在当地的校园IT，研究组和部门环境中有效地部署和运行。数字内容不再需要转移到公共存储库，以便数字信息接受合适的内容管理。

内容管理微服务

UC3内容管理微服务旨在实现下列发展社区的最佳实践的战略目标：

1. 通过冗余提供安全(实现了“大量备份保持资料安全[lots of copies keeps stuff safe]”的原则；(Reich & Rosenthal, 2001))
2. 通过描述(description) 进行维护(“大量描述保证东西有意义[Lots of description keeps stuff meaningful]”)
3. 通过服务提供功能(“大量服务保证东西可用[Lots of services keeps stuff useful]”).
4. 通过使用增加价值(“大量功能保证东西有价值[Lots of uses keeps stuff valuable]”)

因此，整体基础设施框架是按照四个分层服务层安排的初步设置的12个微型服务来构思的，每个服务层建立在较低层次的必要基础功能之上，并且总体上趋于充分（参见表1）。虽然微型服务被分配了一种模式并将重点放在了分类的目的上，但实际上，服务在整个生产周期中具有广泛的适用性（见图1）。

|  |  |  |
| --- | --- | --- |
| Mode 模式 | Focus 关注点 | Layer / micro-service  层次/微服务 |
| Curation 内容管理 | Value 价值 | Interoperation 交互  • Annotation 注解  • Notification 通知 |
| Service 服务 | Application 应用  • Transformation 转换  • Search 查找  • Index 索引  • Ingest 摄取 |
| Preservation 保存 | Context 环境 | Interpretation 接口  • Characterization 描述  • Inventory 详细目录 |
| State 状态 | Protection 保护  •Replication 复制  •Fixity 固定性  •Storage 存储  •Identity 身份 |

表1. 内容管理微服务

保护层身份和存储服务是整个微服务框架的基础。Identity服务提供了一种持久的、明确的区分和指出给定部分的内容的方法。 Storage服务为该内容的持续管理提供了一个安全的环境。 Fixity服务提供了检测持久化内容的完整性的损害的方法，并且同步的复制服务管理内容副本。

注意Protection操作内容的四个组件状态没有任何表示理解的内容。策划内容的上下文意义由上级解释层管理。Inventory服务为保护层中管理的内容维护一个全面的，与模式无关的元数据目录。Characterization服务提供了一个自动化的方法来检查和提取的属性格式化字节流潜在重要的管理内容为目的的管理和保存分析、规划和干预（Abrams，Morrissey和Cramer，2008）。

发现/使用/重用 转变 创建/接收 评估/选择 取食 描述 商店 监控

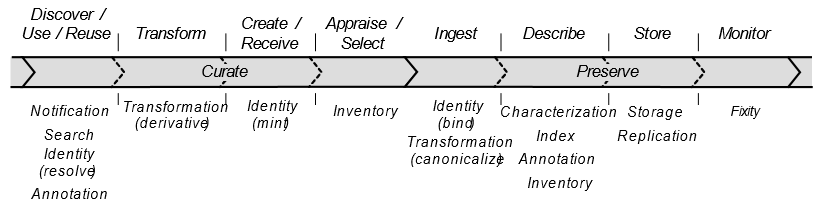


图1.微服务生命周期的适用性(改编自 Higgins, 2008).

保护和解释层集中地以后台保存模式运行，通常由存储库管理器（例如，UC3工作人员）直接管理。 上层服务层提供面向用户的内容管理服务。 应用层支持生产者和消费者用户的基线功能。 Ingest服务提供了将新内容加入到内容管理环境中的手段，其界面适用于手动和自动化工作流。 索引和搜索服务支持内容和元数据库的搜索，浏览和检索。 转换服务提供了将内容转码为所需形式的手段，用于摄取规范化，保存迁移和创建传递衍生工具。

上层互操作层支持通过消费者驱动的使用和丰富来增加设计内容的服务。 通知服务提供了通知用户社区新获取内容的可用性的方法。 注释服务提供了内容内容管理人和消费者可以描述微服务基础架构中管理的内容的重要属性的方法。

设计单个的内容管理微服务基于以下原则：

•粒度和正交性。

•通过组合而不是添加的复杂性。

•持久的界面，不断发展的实现。

•灵活的配置，但有意义的反射行为（“最不惊奇的原则”）。

•推迟实施决策，直到需求和结果得到明确的理解。

正如前面所说，复杂性是微服务方法紧急的属性。 换句话说，复杂的内容管理功能是通过个性化的原子服务的灵活组合产生的，而不是通过将功能添加到越来越大的整体系统中。 单片系统范围的不断扩大确实增加了功能，但是以复杂性为代价，使开发变得复杂化，抑制维护，并增加了错误行为的可能性。 UC3的优先考虑是将策略功能积极地转移到简单，集中，独立，但可互操作的微服务中。

微服务通过定义其良好的接口来揭示其公共服务的功能（Liegl，2007; O'Reilly，2005）。 关于UC3内容管理功能的持久性和可持续性的断言是相对于这些接口而进行的，而不是它们的底层实现，它们可以并且将随着时间的推移自由发展，而不会使较高级别的接口变得无效。 接口设计是基于给定服务的主要概念实体，它根据可以访问和操纵该状态的状态属性和行为来定义。 单独的状态属性是强类型的，并且被分配唯一的形式标识符，在适当的范围单元内保证是唯一的，使得实体状态定义可以被公开暴露为可重用的本体。

抽象接口映射到三个互动模式:程序api在不同语言绑定,命令行api支持的主要操作系统命令shell;和web api符合REST规范(Fielding＆Taylor，2002)，并结合主要浏览器支持的瘦客户端GUI（参见图2）。其目的是为数字资产管理员和内容管理平台的管理员提供与服务交互的手段，而不需要对已建立的工作流程和模式进行重大改变。

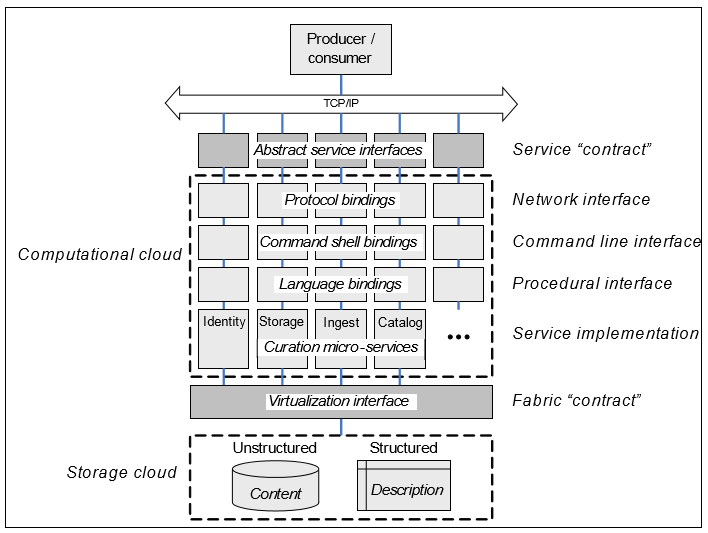


图2. 微服务栈

微服务程序API的初始语言是Java和Perl。 Java RESTful API是使用Jersey框架构建的，Jersey框架是在Jetty或Tomcat容器中运行的参照于RESTful Web Services的JSR 311，JAX-RS - Java API的实现。Perl和Java实现强调瘦的命令行工具,提供尽可能多的功能可行的shell命令, 但是它本身对基于语言的方法已经提供的功能增加了最小的功能; 以这种方式，最大功能被推入最低级别，在所有三种模式中可用。

作为这些设计原则的示例，存储服务将在以下部分中进行详细描述。 由于微服务正在开发中，下面描述的设备不包括其中更多的推测性组件。

存储服务

存储服务管理保存内容的数字表示的文件的非结构化存储（即，没有公共数据模型）。 （结构化存储由Inventory服务提供）通过设计，存储服务对于存储内容的底层语义是不透明的，由高级库存服务管理。 因此，存储服务对数字对象的定义很弱，这仅仅是从单个目录下降的一组相关文件，其状态可以通过一系列离散版本被随时修改。 根据策略，UC3加强了这一点，要求目录层次结构包含与数字对象相关的每个非衍生文件。

**概念建模**

存储服务基于五个概念实体，每个概念实体根据其状态属性和状态操纵行为进行定义。

1. 服务。 存储服务本身 存储服务充当多个定义的存储节点的中央代理，可以为管理或技术方便而定义。 全部服务状态包括：

o服务名称，标识符和版本。

o存储节点的枚举。

o对象，版本和文件数。

o总大小。

o访问和支持URI。

该服务包含任意数量的存储节点。

1. 节点。 负责管理服务已知的内容子集的实体。 节点通常基于其基础存储技术或策略制度来定义。 节点状态包括：

o节点名称，标识符和版本

o对象，版本和文件数。

o总大小。

o存储介质：硬盘，磁带，光盘，固态。

o访问模式：在线，近线，脱机。

o访问和支持URI。

存储节点包含任意数量的数字对象。

1. 对象。 一组版本化的文件，代表一个智力一致的内容单元。 对象状态包括：

o对象标识符。

o枚举版本。

o版本和文件数。

o总大小。

o创建，修改，上次验证和最后访问日期/'时间戳。

o访问URI。

一个对象包含任意数量的版本。

1. 版本。 一组代表数字对象在某个时间点的离散状态的文件。 版本状态包括：

o版本标识符。

o文件数。

o总大小。

o创建，修改，上次验证和最后访问日期/时间戳。

o访问URI。

版本标识符以数字顺序分配，从1开始。保留的版本号0不引用固定版本，而是作为始终代表当前版本的访问同义词。 一个版本包含任意数量的文件。

1. 文件。 一个命名的数字八位字节流。 请注意，文件八位字节流被命名，但不是键入; 存储服务不涉及表示为数字对象的抽象内容的含义。 文件状态包括：

o文件标识符。

o尺寸。

o创建，修改，上次验证和最后访问日期/时间戳。

o访问URI。

**方法**

存储服务支持许多访问和操纵概念实体及其状态的方法。 每种方法根据幂等和安全性的重要事务性质进行分类（Fielding等，1999）。

* *Help* [idempotent, safe]
* *Get-service-state* [idempotent, safe]
* *Get-node-state* [idempotent, safe]
* *Get-object* [idempotent, unsafe]
* *Get-object-state* [idempotent, safe]
* *Get-version* [idempotent, unsafe]
* *Get-version-state*[idempotent, safe]
* *Get-file* [idempotent, unsafe]
* *Get-file-state* [idempotent, safe]
* *Add-version* [non-idempotent, unsafe]
* *Delete-object* [idempotent, unsafe] • *Delete-version* [idempotent, unsafe]

帮助方法对所有微服务是通用的，并提供简短的描述性文本，所有支持的方法的枚举以及支持联系URI。 Getobject，Get-version和Get-file方法几乎不安全，因为它们使用当前的访问时间戳来修改它们各自的状态。 请注意，修改对象内容的唯一机制是引入新版本。 删除对象和删除版本方法是为了完整性而定义的，但作为策略的一个问题只能用于响应不寻常的内容管理环境。

每个方法首先被定义为抽象，然后映射到特定的协议。 例如，表2中总结了Get-file-state方法定义。该抽象方法定义映射到Web，命令行和过程API指定的具体语法，如图3所示。所有实现 细节隐藏在接口后面，构成公共服务合同。 可以请求支持的状态信息的响应表单是ANVL（Kunze，Kahle，Masanes，＆Mohr，2005），HTML，JSON，RDF / Turtle和XML。

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Obligation | Description |
| Node | Identifier | Mandatory | Storage node |
| Object | Identifier | Mandatory | Object identifier |
| Version | Identifier | Mandatory | Version identifier |
| File | Identifier | Mandatory | File identifier |
| Form | Enum | Optional | Response form |
| RETURN | State | Mandatory | File state |
| SIDE EFFECTS | *Not applicable* | |  |
| ERRORS | Badly formed request | |  |
| Node not found | |  |
| Object not found | |  |
| Version not found | |  |
| File not found | |  |
| Unsupported response form | |  |

表2. *Get-file-state* 方法.

Get /fileState/*node*/*object*/*version*/*file* HTTP/1.1

Accept: application/json

% store getFileState *node object version file* –f json File.getState(node, object, version, file, Form.JSON);

图3. *Get-file-state* 接口格式.

**实现**

在整个实施过程中应用了粒度和正交性的一般微服务原则。 因此，存储服务依赖于许多辅助规范，约定和系统（在http://www.cdlib.org/inside/diglib/>中更详细地描述）。

存储服务在文件系统中实例化，如图4所示。“0 = name\_version”形式的文件是一个Namaste标签（名称为文本），用作存储服务主目录的指示性签名; 在这种情况下，它指定此实例符合服务规范的版本0.8。 “admin /”目录包含各种管理声明，“log /”目录包含访问和诊断日志。 服务的全局状态属性由文件“store-info.txt”定义（参见图5）。

服务已知的存储节点由文件“nodes.txt”中的名称和访问URI定义（见图6）。 节点可以是运行Storage服务的主机的远程或本地。 本地互操作性假设存储节点在本地主机可安装的文件系统中实例化; 远程节点通过TCP / IP网络通过其访问URI进行访问。

store\_home/

0=store\_0.7 admin/ log/ nodes.txt store-info.txt

图4. Storage service 文件系统结构

Name: store

Service-scheme: Store/0.7

Node-scheme: CAN/0.8

Verify-on-read: true

Verify-on-write: true

Access-uri: http://store.cdlib.org/

Support-uri: email:store-support@cdlib.org

图5. Storage file 配置文件

can01 http://can01.cdlib.org/ can02 http://can02.cdlib.org/ can03 file:///home/can03

图6. Storage节点文件

存储节点的默认实现是内容接入节点（CAN，参见图7和图8），其基本上是存储库实例。 CAN管理其分层文件系统树中的对象。 树的分支结构的主要约定是Pairtree，它使用对象标识符的二进制分解来确定找到对象的内容的目录层次结构。 因此，在相对目录路径“ab / c1 / 23”的结尾处将找到具有标识符“abc123”的对象。 （Pairtree定义了转义规则，以防止标识符字符与文件系统语义之间的冲突。）符合微服务独立性原则，Pairtree已被多个外部机构和计划采用。 例如，HathiTrust（York，2009）正在使用它来存储数百万个扫描图书。 可用的开源Perl代码支持Pairtree，Namaste和ANVL（Kunze，2009）。

|  |
| --- |
| can\_home/  0=can\_0.8 admin/ can-info.txt log/ store/  pairtree\_root/  0=pairtree\_0.1 pairtree-info.txt  ab/ c1/  23/ abcd123/ |

图7. CAN 文件系统结构

Pairtree路径末端的叶存储数字对象，但其性质并未由Pairtree指定。 例如，它可以是Bagit包（Boyko，Kunze，Littman，Madden，＆Vargas，2009），HathiTrust数字化书或网络爬行。 在CAN的上下文中，控制该对象的结构的约定是Dflat。

Name: can01

Node-scheme: CAN/0.8

Branch-scheme: Pairtree/0.1

Leaf-scheme: Dflat/0.16

Media-type: magnetic-disk

Access-mode: on-line

Verify-on-read: true

Verify-on-write: true

Access-uri: http://can01.cdlib.org/

图8. CAN 配置文件

Dflat定义了用于管理代表数字对象的版本化文件集的结构（参见图9和图10）。 对象版本存储在“vnnn /”形式的编号目录中。 （对应于编号最多为999的版本的目录名称被左填充以对齐词法和数字排序;高于999的名称自然地每个数量级地扩展一个附加数字。）符号链接“current @”提供对当前版本的直接访问。

dflat\_home/

0=dflat\_0.16 admin/ current@ dflat-info.txt log/ v001/ v002/ V003/

图9. Dflat 文件系统结构

Object-scheme: Dflat/0.16

Manifest-scheme: Checkm/0.1

Full-scheme: Dnatural/0.12

Delta-scheme: ReDD/0.1

Current-scheme: symlink

图10. Dflat 配置文件

CAN是可能属于存储库实例的所有内容的容器。 虽然其规范仍在不断发展，但它捆绑了一个CAN存储库集合由一个或多个Pairtrees表示的位置，每个Pairtree的树叶是Dflats。 根据规定的设计原则，一些实施决策已被推迟，直到需要更清楚地了解为止; 目前缺席的方法是代表固定检查的频率，远程复制站点，注释的可接受性等的策略。

Dflat版本可以以完全实例化或delta压缩形式表示。 当前版本始终完全实例化; 所有以前的版本通常以三角形压缩形式保存，以最大限度地减少存储利用率。 不管表示类型如何，所有版本的目录都保存符合Checkm规范的清单文件（“manifest.txt”），它将大小和消息摘要与每个版本文件相关联。

完全实例化版本表示的结构由附属Dnatural约定（见图11）定义。 从对象的制片人或内容管理人收到的内容数据和元数据分别存储在“数据/”和“元数据”目录中。 “data /”目录的内容完全由制作人或内容管理人（例如，它可以是BagIt包）。 “enrichment”目录包含由Storage Service自身生成的附加元数据和衍生内容。 “注释”目录包含由内容消费者提供的附加元数据和衍生内容。

v003/

0=dnatural\_0.12 admin/ annotation/ data/ enrichment/ manifest.txt metadata/

图11. Dnatural 文件系统结构

压缩版本表示符合反向目录增量（ReDD）约定（见图12）。 ReDD是一个非常简单的与工具和平台无关的方案，它使用文件级反向增量来最大限度地减少总体存储利用率。 “add /”目录包含相对于后续版本需要添加的文件，以便重新实例化以前的版本; “delete.txt”文件列出了相对于后续版本需要删除的文件，以重新实例化以前的版本。换句话说，与版本2相关联的增量信息指示如何操纵版本3的文件，以便恢复版本2的完整形式。因此，更高版本的访问速度更快;按时间顺序早期重新实现版本将需要迭代应用三角洲。 Dflat维护版本的有序序列，并且可以应用于对当前版本和以前版本的概念进行操作的任何差异方案（例如Unix“diff”）。

v002/

0=redd\_0.1 d-manifest.txt delta/ add/ delete.txt manifest.txt

图12. ReDD 文件系统结构

存储服务底层的所有惯例和子系统都由单独的包空间中的过程API支持，因此可以轻松地在其他上下文中重新使用。对文件系统作为范式存储抽象的依赖是现代文件系统的设计和行为特征，例如ZFS（Bonwick＆Moore，2007），其基本上保持恒定的读写性能，独立于总数或大小档案（Abrams，Cruse，Kunze，＆Loy，2009）。由UC3策略部署的存储服务将作为所有关于数字内容单元的所有信息的记录副本。虽然该信息的一部分将由库存服务以结构化形式进行管理，但是为了优化常规管理和内容管理查询，这被认为是重复的，而不是权威的副本。如有必要，可以通过各种存储服务文件系统结构的详尽遍历来全面重新实现库存服务。

所有的微服务实现都被设计为完全独立的，并且以最少的人为干预轻松部署和操作。 虽然UC3将继续提供一个集中管理的内容管理信息库，但微服务方法的目的是促进有效的和有效的管理功能的分布新选区和上下文,包括校园数据中心,学术部门、实验室和野外测站计算集群,以及学者们的桌面。

开发过程

建立UC3微型服务基础设施是从传统和敏捷开发原则中获得的：

•参与用户社区驾驶需求评估和功能要求

•具有频繁重构的早期原型。

•连续构建和测试。

•作为可共同交付的文件，而不是事后的想法。

•一小群早期采用者。

12个微型服务正在按照六个发展阶段的顺序执行（见表3）。 第二，第四和第六波分别表示对应于最低限度，中等和完全功能的策展信息库的显着可交付的里程碑。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| First wave | Second wave  | Third wave | Fourth wave  | Fifth wave | Sixth wave  |
| *Identity* | *Inventory* | *Index* | *Search* | *Notification* | *Annotation* |
| *Storage* | *Ingest* | *Fixity* | *Replication* | *Characterization* | *Transformation* |
| *Object / collection modeling* | | | *Authentication / authorization* | | |
| *Policy and business model development* | | | | | |

表3. 微服务开发过程

第一至第三波伴随着数字对象和对象集合建模的标准和约定发展。 第四到第六波将伴随着共同的认证和授权机制发展。所有六波都将伴随着政策和商业模式。

身份和存储服务目前以工作形式提供; 第二波里程碑交付物将准备在2010年1月接受内容。初步内容将由电子论文和论文策划的多校区试点项目提供。 随后的内容项目将涉及人类学和动物博物馆收藏，环境领域数据和生物样本标本。

结论

为了便于将UC 内容管理服务提供给新的校园选区，以及数字内容的数量，大小和类型多样性的增加，基础策划基础设施必须易于适应当地的需求和做法。一种在一组小而且相互协作的微服务中实现内容管理功能的架构方法提供了必要的部署灵活性，同时也简化了开发和维护工作。通过严格遵守明确界定的公共接口来促进服务互相调用。这允许从个别原子服务的战略组合中产生全面的内容管理功能。