REST API

第一章 产生原因

1.1 面对的问题

现代软件系统的复杂性要更加强调模块化，一个大型的项目被分为很多模块，而模块之间也需要进行**相互通信**以执行所需的任务。 软件架构研究确定如何最好地划分系统、组件之间相互识别和通信、信息如何通信、系统的元素如何独立的发展以及如何使用正式和非正式符号来描述上述所有内容。

1.2 REST的提出及示例

REST(Representational State Transfer)采用了从整体系统需求开始，没有约束，然后逐步识别和约束系统中的元素，用来区分系统设计空间并且注意系统内部行为的流程。 示例以下大概可以分为以下七个阶段：

Figure 5-1. Null Style

阶段一：**NULL。**没有约束，没有样式：

阶段二：**客户端——服务器样式**：

Figure 5-2. Client-Server

阶段三：**客户端——服务器(无状态)。**将服务器端约束为无状态响应。从客户端到服务器的每个请求都必须包含理解请求所需的所有信息，并且不能利用服务器上的任何存储上下文。因此，会话状态完全保留在客户端上。

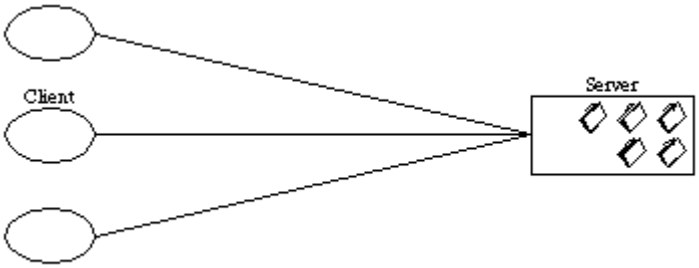

Figure 5-3. Client-Stateless-Server

阶段四：**客户端(缓存)——服务器(无状态)。**存约束要求对请求的响应中的数据被隐式或显式标记为可缓存或不可缓存。如果响应是可缓存的，则客户端缓存有权在以后的等效请求中重用该响应数据。

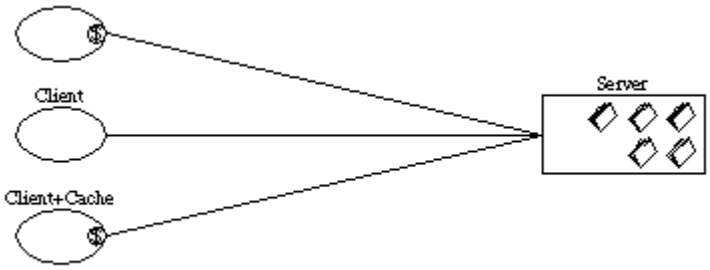

Figure 5-4. Client-Cache-Stateless-Server

阶段五：**客户端(缓存、统一接口)——服务器(无状态)，将REST架构风格与其他基于网络的风格区分开来的核心特征是它强调组件之间的统一接口。**

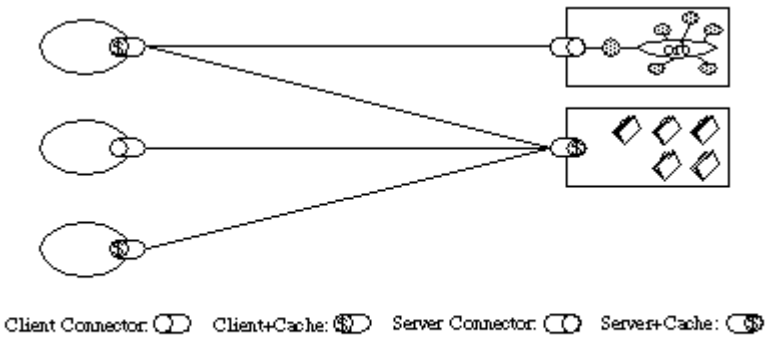

Figure 5-6. Uniform-Client-Cache-Stateless-Server

阶段六：**客户端(缓存、统一接口、分层)——服务器(无响应)。**通过将系统的知识限制在单层，我们限制了整个系统的复杂性并促进了基底的独立性。层可用于封装遗留服务并保护遗留客户端的新服务，通过将不常用的功能移至共享中介来简化组件。

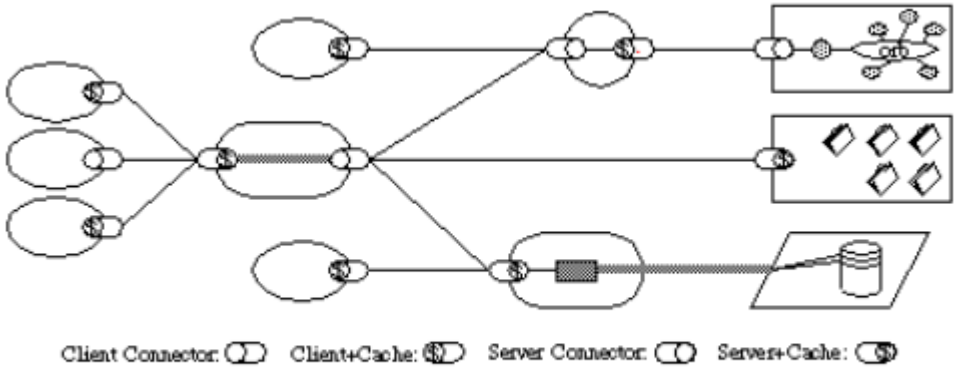

Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

阶段七：**REST。**按需编码：REST 允许通过下载和执行小程序或脚本形式的代码来扩展客户端功能。这通过减少需要预先实现的功能数量来简化客户端。允许在部署后下载功能提高了系统的可扩展性。然而，它也会降低可见性，因此只是 REST

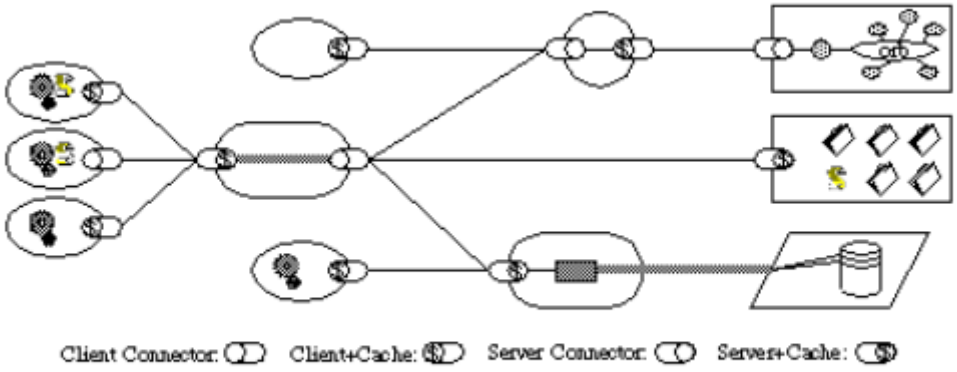

Figure 5-8. REST

中的一个可选约束。

1.3 REST架构元素

1.3.1 数据元素

REST重点关注对数据类型与元数据的共同理解（即客户端与服务端对元数据的理解），但这同样限制了标准化接口所揭示内容的范围。RESt中关键的抽象信息是资源。这种对资源的抽象定义实现了 Web 体系结构的关键特性。首先，**它通过包含许多信息源来提供通用性，而无需人为地按类型或实现方式区分它们。**其次，它允许对表示形式的引用进行后期绑定，从而能够根据请求的特征进行内容协商。REST 使用资源标识符来标识组件之间交互所涉及的特定资源。

1.3.2 连接器

REST 使用各种连接器类型来封装访问资源和传输资源表示的活动。连接器为组件通信提供了一个抽象接口，通过清晰地分离关注点并隐藏资源和通信机制的底层实现，增强了简洁性。 由于连接器管理组件的网络通信，因此可以在多个交互中共享信息，以提高效率和响应能力。所有 REST 交互都是无状态的。也就是说，每个请求都包含连接器理解该请求所需的全部信息，与之前的请求无关。

1.3.3 组件

Table 5-3: REST Components

Component	Modern Web Examples
origin server	Apache httpd, Microsoft IIS
gateway	Squid, CGI, Reverse Proxy
proxy	CERN Proxy, Netscape Proxy, Gauntlet
user agent	Netscape Navigator, Lynx, MOMspider

1.4 REST架构视图

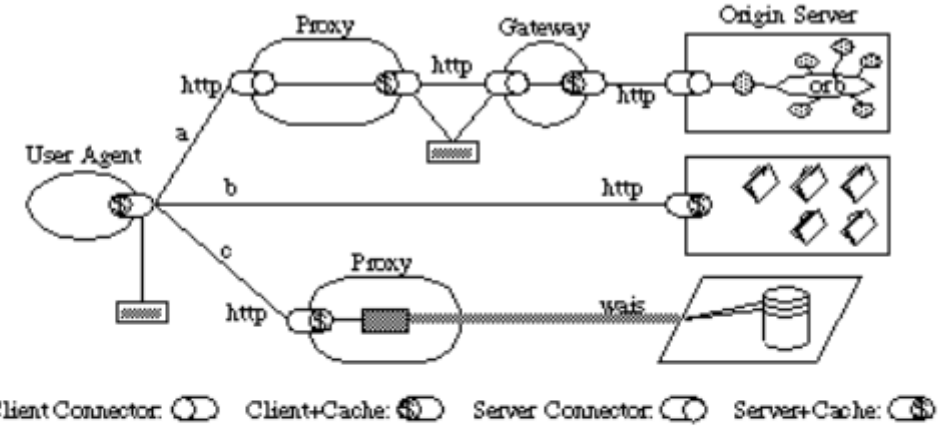

Figure 5-10. Process View of a REST-based Architecture

A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

REST 的客户端-服务器关注点分离简化了组件实现，降低了连接器语义的复杂性，提高了性能调优的有效性，并增加了纯服务器组件的可扩展性。分层系统约束允许在通信的各个点引入中介（代理、网关和防火墙），而无需更改组件之间的接口，从而允许它们通过大规模共享缓存来协助通信转换或提高性能。 REST 通过将消息限制为自描述来实现中间处理：请求之间的交互是无状态的，标准方法和媒体类型用于指示语义和交换信息，响应明确指示可缓存性。（流程视图、连接器视图、数据视图）

第二章 REST两个核心、六个约束

2.1 REST的两个核心

Resource 和 Representation （用url定位资源，用HTTP描述操作）

2.2 六个约束

- **统一接口 (Uniform Interface)**: RESTful API 的关键约束之一是通过统一的接口对资源进行访问和操作,包括识别资源、处理资源的请求以及传输资源的表示。这有助于简化客户端与服务器之间的通信,并提高可移植性。
- **无状态性 (Stateless)**: REST 要求服务器在处理客户端请求时不会保存任何状态信息,每个请求都必须包含足够的信息供服务器理解。这样设计有助于提高可伸缩性和减少服务器负担。
- **以资源为中心 (Resource-Centric)**: RESTful API 设计要以资源为中心,每个资源都有唯一的标识符,并且客户端通过资源的标识符来操作资源。这种设计有利于简化API的设计和理解。
- **使用超媒体 (Hypermedia)**: RESTful API 应该利用超媒体(如链接、表单等)来控制应用的状态转移,而不是依赖外部文档。这样可以提高API的可发现性和可扩展性。
- **分层系统 (Layered System)**: RESTful API 应该遵循分层系统的原则,允许中间层(如负载均衡器、缓存服务器等)在不影响客户端的情况下介入和优化请求-响应的流程。这有助于提高可扩展性和安全性。
- **缓存 (Cacheable)**: RESTful API 应该支持缓存机制,以提高性能和减轻服务器负载。这需要响应消息包含足够的信息来确定其是否可缓存,以及如何进行缓存。

第三章 自我总结

REST是一种系统架构格式，常用于网络应用程序，对于客服——服务器模式很合适。URL在REST中可以很好表示“物理资源”和“适当的操作+响应”（既**Resource**）。HTTP在web架构中扮演着很特殊的角色，即是Web组件之间通信的主要应用程序级协议，也是专门为资源表示（既**Representation**）传输而设计的唯一协议，并REST架构系统可以跟随HTTP的扩展而扩展，升级而升级。通过分层的架构，可以使用连接器来单独的管理组件之间的通信，对于组件来说只要接口不变，那么连接器的具体实现的更改并不会影响到组件之间的通信。而无状态和缓存可以提高组件的处理某些资源的性能。