

《微服务设计》

图书基本信息

书名：《微服务设计》

13位ISBN编号：9787115420262

出版时间：2016-5

作者：[英] Sam Newman

页数：228

译者：崔力强,张 骏

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：www.tushu000.com

《微服务设计》

内容概要

本书全面介绍了微服务的建模、集成、测试、部署和监控，通过一个虚构的公司讲解了如何建立微服务架构。主要内容包括认识微服务在保证系统设计与组织目标统一上的重要性，学会把服务集成到已有系统中，采用递增手段拆分单块大型应用，通过持续集成部署微服务，等等。

《微服务设计》

作者简介

作者简介：

Sam Newman

是ThoughtWorks公司的技术专家、ThoughtWorks内部系统架构师，同时还为全球的客户提供咨询服务。他在开发和IT运维方面与全球多个领域的公司有合作。

译者简介：

崔力强

阿里巴巴技术专家，目前专注于持续交付相关的产品开发。曾在ThoughtWorks任职多年，从事软件定制开发、敏捷软件开发的相关咨询等工作，帮助过数个团队和项目进行精益需求管理、软件设计、自动化测试和持续集成等实践。微信号：blade_1986

张骏

2010年加入ThoughtWorks公司。作为开发人员、项目经理、资深敏捷教练和资深咨询师，在金融、电信和能源服务行业的大型复杂业务系统的设计、开发、管理、咨询等方面有丰富的经验。曾为国内外诸多客户提供软件设计、开发以及咨询服务。拥有10年工作经验，在Scrum、看板、规模化敏捷等方法论，以及精益需求管理、自动化测试、持续集成、领域驱动设计、微服务等具体实践方面都有丰富的积累。微信号：zhangjun695339

书籍目录

前言	xiv
第1章 微服务	1
1.1 什么是微服务	2
1.1.1 很小，专注于做好一件事	2
1.1.2 自治性	3
1.2 主要好处	3
1.2.1 技术异构性	3
1.2.2 弹性	4
1.2.3 扩展	5
1.2.4 简化部署	5
1.2.5 与组织结构相匹配	6
1.2.6 可组合性	6
1.2.7 对可替代性的优化	6
1.3 面向服务的架构	7
1.4 其他分解技术	7
1.4.1 共享库	8
1.4.2 模块	8
1.5 没有银弹	9
1.6 小结	10
第2章 演化式架构师	11
2.1 不准确的比较	11
2.2 架构师的演化视角	12
2.3 分区	14
2.4 一个原则性的方法	15
2.4.1 战略目标	15
2.4.2 原则	15
2.4.3 实践	16
2.4.4 将原则和实践相结合	16
2.4.5 真实世界的例子	16
2.5 要求的标准	17
2.5.1 监控	18
2.5.2 接口	18
2.5.3 架构安全性	18
2.6 代码治理	18
2.6.1 范例	19
2.6.2 裁剪服务代码模板	19
2.7 技术债务	20
2.8 例外管理	21
2.9 集中治理和领导	21
2.10 建设团队	22
2.11 小结	23
第3章 如何建模服务	24
3.1 MusicCorp简介	24
3.2 什么样的服务是好服务	25
3.2.1 松耦合	25
3.2.2 高内聚	25
3.3 限界上下文	26

3.3.1	共享的隐藏模型	26
3.3.2	模块和服务	27
3.3.3	过早划分	28
3.4	业务功能	28
3.5	逐步划分上下文	29
3.6	关于业务概念的沟通	30
3.7	技术边界	30
3.8	小结	31
第4章	集成	32
4.1	寻找理想的集成技术	32
4.1.1	避免破坏性修改	32
4.1.2	保证API的技术无关性	32
4.1.3	使你的服务易于消费方使用	33
4.1.4	隐藏内部实现细节	33
4.2	为用户创建接口	33
4.3	共享数据库	33
4.4	同步与异步	35
4.5	编排与协同	35
4.6	远程过程调用	38
4.6.1	技术的耦合	38
4.6.2	本地调用和远程调用并不相同	39
4.6.3	脆弱性	39
4.6.4	RPC很糟糕吗	40
4.7	REST	41
4.7.1	REST和HTTP	41
4.7.2	超媒体作为程序状态的引擎	42
4.7.3	JSON、XML还是其他	44
4.7.4	留心过多的约定	44
4.7.5	基于HTTP的REST的缺点	45
4.8	实现基于事件的异步协作方式	46
4.8.1	技术选择	46
4.8.2	异步架构的复杂性	47
4.9	服务即状态机	48
4.10	响应式扩展	48
4.11	微服务世界中的DRY和代码重用的危险	49
4.12	按引用访问	50
4.13	版本管理	51
4.13.1	尽可能推迟	51
4.13.2	及早发现破坏性修改	52
4.13.3	使用语义化的版本管理	53
4.13.4	不同的接口共存	53
4.13.5	同时使用多个版本的服务	54
4.14	用户界面	55
4.14.1	走向数字化	56
4.14.2	约束	56
4.14.3	API组合	57
4.14.4	UI片段的组合	57
4.14.5	为前端服务的后端	59
4.14.6	一种混合方式	60

4.15	与第三方软件集成	61
4.15.1	缺乏控制	61
4.15.2	定制化	62
4.15.3	意大利面式的集成	62
4.15.4	在自己可控的平台进行定制化	62
4.15.5	绞杀者模式	64
4.16	小结	65
第5章	分解单块系统	66
5.1	关键是接缝	66
5.2	分解MusicCorp	67
5.3	分解单块系统的原因	68
5.3.1	改变的速度	68
5.3.2	团队结构	68
5.3.3	安全	68
5.3.4	技术	68
5.4	杂乱的依赖	69
5.5	数据库	69
5.6	找到问题的关键	69
5.7	例子：打破外键关系	70
5.8	例子：共享静态数据	71
5.9	例子：共享数据	72
5.10	例子：共享表	73
5.11	重构数据库	74
5.12	事务边界	75
5.12.1	再试一次	76
5.12.2	终止整个操作	77
5.12.3	分布式事务	77
5.12.4	应该怎么办呢	78
5.13	报告	78
5.14	报告数据库	78
5.15	通过服务调用来获取数据	80
5.16	数据导出	81
5.17	事件数据导出	82
5.18	数据导出的备份	83
5.19	走向实时	84
5.20	修改的代价	84
5.21	理解根本原因	84
5.22	小结	85
第6章	部署	86
6.1	持续集成简介	86
6.2	把持续集成映射到微服务	87
6.3	构建流水线和持续交付	90
6.4	平台特定的构建物	91
6.5	操作系统构建物	92
6.6	定制化镜像	93
6.6.1	将镜像作为构建物	94
6.6.2	不可变服务器	95
6.7	环境	95
6.8	服务配置	96

6.9	服务与主机之间的映射	97
6.9.1	单主机多服务	97
6.9.2	应用程序容器	99
6.9.3	每个主机一个服务	100
6.9.4	平台即服务	101
6.10	自动化	101
6.11	从物理机到虚拟机	102
6.11.1	传统的虚拟化技术	103
6.11.2	Vagrant	104
6.11.3	Linux容器	104
6.11.4	Docker	106
6.12	一个部署接口	107
6.13	小结	109
第7章	测试	110
7.1	测试类型	110
7.2	测试范围	111
7.2.1	单元测试	112
7.2.2	服务测试	113
7.2.3	端到端测试	114
7.2.4	权衡	114
7.2.5	比例	115
7.3	实现服务测试	115
7.3.1	mock还是打桩	115
7.3.2	智能的打桩服务	116
7.4	微妙的端到端测试	117
7.5	端到端测试的缺点	118
7.6	脆弱的测试	118
7.6.1	谁来写这些测试	119
7.6.2	测试多长时间	119
7.6.3	大量的堆积	120
7.6.4	元版本	120
7.7	测试场景，而不是故事	121
7.8	拯救消费者驱动测试	121
7.8.1	Pact	123
7.8.2	关于沟通	124
7.9	还应该使用端到端测试吗	124
7.10	部署后再测试	125
7.10.1	区分部署和上线	125
7.10.2	金丝雀发布	126
7.10.3	平均修复时间胜过平均故障间隔时间	127
7.11	跨功能的测试	128
7.12	小结	129
第8章	监控	131
8.1	单一服务，单一服务器	132
8.2	单一服务，多个服务器	132
8.3	多个服务，多个服务器	133
8.4	日志，日志，更多的日志	134
8.5	多个服务的指标跟踪	135
8.6	服务指标	135

8.7	综合监控	136
8.8	关联标识	137
8.9	级联	139
8.10	标准化	139
8.11	考虑受众	140
8.12	未来	140
8.13	小结	141
第9章	安全	143
9.1	身份验证和授权	143
9.1.1	常见的单点登录实现	144
9.1.2	单点登录网关	145
9.1.3	细粒度的授权	146
9.2	服务间的身份验证和授权	146
9.2.1	在边界内允许一切	146
9.2.2	HTTP(S) 基本身份验证	147
9.2.3	使用SAML或OpenID Connect	148
9.2.4	客户端证书	148
9.2.5	HTTP之上的HMAC	149
9.2.6	API密钥	149
9.2.7	代理问题	150
9.3	静态数据的安全	152
9.3.1	使用众所周知的加密算法	152
9.3.2	一切皆与密钥相关	153
9.3.3	选择你的目标	153
9.3.4	按需解密	153
9.3.5	加密备份	153
9.4	深度防御	154
9.4.1	防火墙	154
9.4.2	日志	154
9.4.3	入侵检测（和预防）系统	154
9.4.4	网络隔离	155
9.4.5	操作系统	155
9.5	一个示例	156
9.6	保持节俭	158
9.7	人的因素	158
9.8	黄金法则	158
9.9	内建安全	159
9.10	外部验证	159
9.11	小结	159
第10章	康威定律和系统设计	161
10.1	证据	161
10.1.1	松耦合组织和紧耦合组织	162
10.1.2	Windows Vista	162
10.2	Netflix和Amazon	162
10.3	我们可以做什么	163
10.4	适应沟通途径	163
10.5	服务所有权	164
10.6	共享服务的原因	164
10.6.1	难以分割	164

10.6.2	特性团队	164
10.6.3	交付瓶颈	165
10.7	内部开源	166
10.7.1	守护者的角色	166
10.7.2	成熟	166
10.7.3	工具	167
10.8	限界上下文和团队结构	167
10.9	孤儿服务	167
10.10	案例研究：RealEstate.com.au	168
10.11	反向的康威定律	169
10.12	人	170
10.13	小结	170
第11章	规模化微服务	171
11.1	故障无处不在	171
11.2	多少是太多	172
11.3	功能降级	173
11.4	架构性安全措施	174
11.5	反脆弱的组织	175
11.5.1	超时	176
11.5.2	断路器	176
11.5.3	舱壁	178
11.5.4	隔离	179
11.6	幂等	179
11.7	扩展	180
11.7.1	更强大的主机	181
11.7.2	拆分负载	181
11.7.3	分散风险	181
11.7.4	负载均衡	182
11.7.5	基于worker的系统	184
11.7.6	重新设计	184
11.8	扩展数据库	185
11.8.1	服务的可用性和数据的持久性	185
11.8.2	扩展读取	185
11.8.3	扩展写操作	186
11.8.4	共享数据库基础设施	187
11.8.5	CQRS	187
11.9	缓存	188
11.9.1	客户端、代理和服务器端缓存	188
11.9.2	HTTP缓存	189
11.9.3	为写使用缓存	190
11.9.4	为弹性使用缓存	190
11.9.5	隐藏源服务	191
11.9.6	保持简单	191
11.9.7	缓存中毒：一个警示	192
11.10	自动伸缩	192
11.11	CAP定理	193
11.11.1	牺牲一致性	194
11.11.2	牺牲可用性	195
11.11.3	牺牲分区容忍性	195

11.11.4	AP还是CP	196	
11.11.5	这不是全部或全不	196	
11.11.6	真实世界	197	
11.12	服务发现	197	
11.13	动态服务注册	199	
11.13.1	Zookeeper	199	
11.13.2	Consul	200	
11.13.4	构造你自己的系统	201	
11.13.5	别忘了人	201	
11.14	文档服务	201	
11.14.1	Swagger	202	
11.14.2	HAL 和HAL浏览器	202	
11.15	自描述系统	203	
11.16	小结	203	
第12章	总结	204	
12.1	微服务的原则	204	
12.1.1	围绕业务概念建模	205	
12.1.2	接受自动化文化	205	
12.1.3	隐藏内部实现细节	205	
12.1.4	让一切都去中心化	206	
12.1.5	可独立部署	206	
12.1.6	隔离失败	206	
12.1.7	高度可观察	207	
12.2	什么时候你不应该使用微服务	207	
12.3	临别赠言	208	
	关于作者	209	
	关于封面	209	

1、最近几年我一直在微服务相关的工作，编码、学习模式、寻找并使用开源工具，到大会做分享... 这本《微服务设计》我读起来还是有很多比较切身的感触的，这里记录下。首先这本书最后一章有一段说的特别好，“你越不了解一个领域，为服务找到合适的界限上下文就越难.....服务的界限划分错误，可能会导致不得不频繁地更改服务间的协作，而这种更改成本更高.....”，软件行业从业者，尤其是那些已经不写代码的从业者，总会期望有银弹，但银弹终究是没有的，微服务也是。我个人觉得微服务本质上是要解决一个 Scalability 的问题，这里的 Scalability 不仅仅是指应对用户量增加，还指业务复杂度增加，数据量增加，团队人员增加（具体参考《The Art of Scalability》一书）。微服务的一套方法论和具体实践方法给我们指了一条路，但路总是需要自己走的。这本书的大量内容是对其他书籍的提炼并用更现代化的语言阐述，例如，要理解微服务你不得不去阅读《领域驱动设计》（或者《实现领域驱动设计》），因为所有服务都是围绕领域来的；例如，要懂得如何应对大量服务的快速发布，你需要去阅读《持续交付》，再补充 Docker 相关的知识；例如，要学会如何保证大量服务交互时候的稳定性，你需要去阅读《Release It!》；此外，常见的分布式知识，如负载均衡、CAP理论，如何使用缓存，该学的你一样都不能拉下；对了，你还得知道 DevOps 以及如何监控你的系统和服务（可惜未见比较好的关于监控的书籍，有朋友知道的话请推荐）。书中的第4, 5章是我觉得比较有原创性的内容，怎么拆分一个巨型应用，第5章有很多切实的建议；而第4章告诉你，当你有一堆形状各异的服务的时候，他们之间集成可能会遇到什么麻烦，以及如何避免及应对。不过，我不得不说，此书完全不适合编程新手阅读，因此书中几乎没有有什么样例代码可以拿去实践的，而书中涉及了大量的开源工具，随便拉一个如 Hystrix, Dropwizard's Metrics 都可以去研究好几天，我是非常佩服作者的视野的。没打五星的另外一个原因是，我读毕没有感觉啊哈！意思就是，书的结构并没有给我一个清晰的系统或者脉络，我自认已经实践过了书中六七成的内容，心中也没有一个清晰的系统抽象，本来指望这本书能有所帮助，然而还是失望了。如果你在维护巨型应用并濒临死亡，那么还是应该走微服务这条活路的，但可以想象死而复生的道路比死亡本身难很多。

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:www.tushu000.com