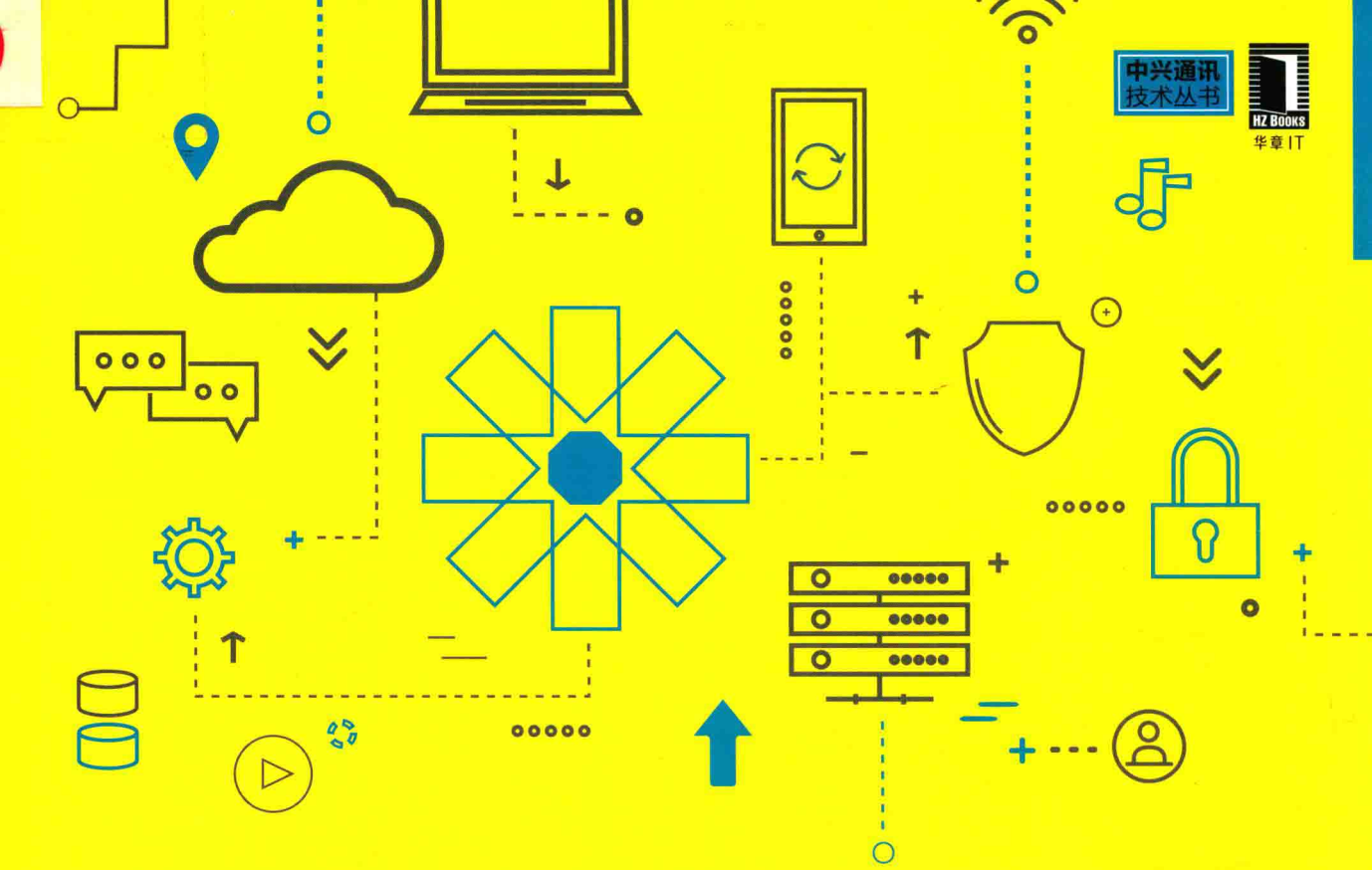


中兴通讯
技术丛书

HZ BOOKS
华章IT

ODL技术内幕

架构设计与实现原理

ODL INTERNALS: ARCHITECTURE, DESIGN AND PRINCIPLE

耿兴元◎著

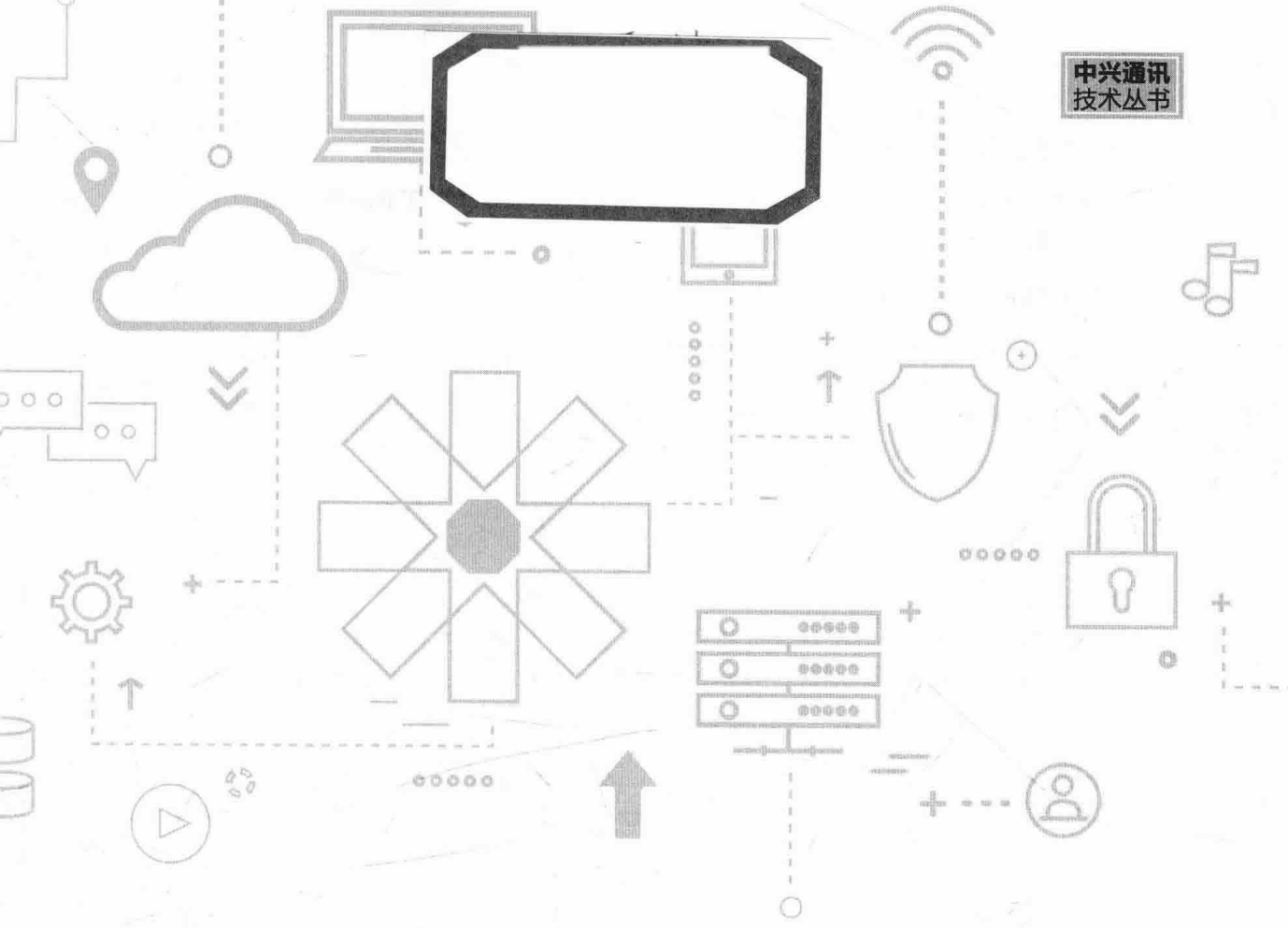
中兴通讯拥有15年以上通信类软件研发及系统设计经验的专家级工程师撰写

SDNLAB强烈推荐

深入分析ODL源码，洞悉ODL先进架构、设计思想和实现原理，实现高效SDN开发

机械工业出版社
China Machine Press

中兴通讯
技术丛书



ODL技术内幕

架构设计与实现原理

ODL INTERNALS: ARCHITECTURE, DESIGN AND PRINCIPLE

耿兴元◎著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

ODL 技术内幕：架构设计与实现原理 / 耿兴元著 . —北京：机械工业出版社，2019.9
（中兴通讯技术丛书）

ISBN 978-7-111-63509-3

I. O… II. 耿… III. 程序设计 IV. TP311.1

中国版本图书馆 CIP 数据核字（2019）第 175764 号

ODL 技术内幕：架构设计与实现原理

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：张锡鹏 责任校对：李秋荣
印 刷：北京文昌阁彩色印刷有限责任公司 版 次：2019 年 9 月第 1 版第 1 次印刷
开 本：186mm×240mm 1/16 印 张：16.25
书 号：ISBN 978-7-111-63509-3 定 价：79.00 元

客服电话：（010）88361066 88379833 68326294 投稿热线：（010）88379604
华章网站：www.hzbook.com 读者信箱：hzit@hzbook.com

内容介绍

这是一本从源代码层面深入剖析ODL的著作，旨在帮助读者在透彻理解ODL的先进架构、设计思想和实现原理后，能更高效地进行SDN开发。

作者是资深的ODL专家和SDN领域的布道者，在通信类软件研发和系统设计领域有超过15年的经验，对ODL及其源码有深入的研究和理解。ODL架构演进极快，核心模块和接口变动频繁，ODL子项目众多，功能和接口碎片化严重，学习门槛很高，面对数百万行的源代码更是让人无从下手。

作者根据自己的经验，对ODL的核心功能及其源代码（氟版本和氖版本）进行反复提炼、抽丝剥茧，不仅让读者理解ODL的系统架构、设计思想、实现原理，更能让读者领略ODL实现源码中的优秀代码和设计模式，最终实现让读者更高效地使用SDN的目的，掌握SDL的精髓。

全书13章，分为三个部分：

第一部分 基础环境篇（第1~2章）

主要介绍了ODL的核心概念、架构、设计目标、编译构建环境的搭建、源码阅读的方法，以及ODL社区对众多子项目的管理实践。

第二部分 核心架构篇（第3~10章）

从源代码的角度详细分析了ODL的基本对象、数据树、MD-SAL DataStore、MD-SAL RPC、MD-SAL Notification、MD-SAL Mount、MD-SAL Cluster Service的工作机制与实现原理。

第三部分 公共组件篇（第11~13章）

从源代码角度详细分析了ODL的AAA、RESTCONF、Blueprint等公共组件的设计、实现与扩展。

作者简介

耿兴元

资深ODL技术专家，目前就职于中兴通讯，是操作系统及支撑平台的软件专家级工程师，在通信类软件研发及系统设计领域已有超过15年的工作经验。在2015~2017年间，负责基于ODL的商用SDN控制器平台的设计和研发管理工作。

在ODL领域有多年的研究和实践经验，曾与SDNLAB一起创建了开源项目Jaguar(基于ODL的Kubernetes网络解决方案)，是该开源项目的主要管理者与贡献者。在SDNLAB上录制了ODL的系列课程，深受读者欢迎。

投稿快速通道

投稿邮箱 / 微信：15693352@qq.com

联系人：杨福川

華章圖書

一本打开的书，一扇开启的门，
通向科学殿堂的阶梯，托起一流人才的基石。

Preface 前　言

OpenDaylight（简称 ODL）项目是 Linux 基金会旗下的一个开源合作项目，致力于推进软件定义网络（SDN）和网络功能虚拟化（NFV）的发展，目的是寻求以一种更透明的方式促进该领域的创新。该组织由行业领先者建立，旨在制定一个统一、开放的平台，驾驭合作开发的力量，以驱动产业和生态圈的创新。ODL 是针对企业、服务提供商、数据中心、WAN 的模块化的开放的 SDN 平台，其基于 OSGi 的微服务架构让用户能够按需部署网络服务、应用、协议和插件。ODL 平台基本每半年发布一次大版本，从 2013 年开始，项目经历了多个版本的迭代已趋于成熟和稳定，几乎成为网络创新应用场景的默认选择。

ODL 把被控制的网络看作是一个消息驱动的巨大状态机，因此该平台核心架构即其模型驱动的状态保存机制与消息转发机制，称为模型驱动的业务抽象层（MD-SAL）。该架构的魅力源于其架构的前瞻性、可塑性和长期演进能力。

ODL 在架构设计上的先进性和灵活性，使其成为 SDN 领域最有影响力的开源平台。灵活的插件机制与在规模上的弹性可扩展，使 ODL 可应用于智慧城市和其他 IoT 应用等涉及多种设备类型以及多种网络技术的场景，包括光交换、IP/MPLS、LTE 或 5G 无线网络。ODL 对这些技术的可编程性可以做到设备无关。如今，ODL 在各场景的网络创新中的应用越来越广泛，利用和贡献 ODL 项目的中国公司数量也在不断增长，包括华为、联想、瑞斯康达、腾讯、Zenlayer、中兴、阿里巴巴和百度等。2017 年 3 月，中兴通讯成为国内首家 ODL 白金会员。

为什么写这本书

ODL 不仅仅是一个 SDN 控制器平台，它还是一个优秀的模型驱动架构实现，以及一个典型的分布式系统设计范例。通过 ODL，我们能学习的不仅仅是 SDN，也能学到其通用的编程技术及软件架构设计，其分布式系统设计实现也非常值得我们借鉴。

但是，ODL 作为一个开源项目，也有令人诟病的地方，主要有两点：一是虽然 ODL

架构设计比较先进，代码实现也比较优秀，但 ODL 缺乏较为系统性的文档，而且仅有的一些文档更新也较为滞后，内容陈旧，容易误导新手；再加上语言和文化背景的差异，足以让大量国内 ODL 初级玩家望而却步。二是 ODL 的架构演进非常快，核心模块和接口变动频繁，再加上 ODL 子项目众多，导致 ODL 功能和接口碎片化严重，开发者一开始面对几百万行代码，确实有点“老虎吃天，无从下口”的感觉。

我作为一个一直从事软件研发工作的工程师，深知学习与应用 ODL 的不易，所以我想是不是可以把我之前学习 ODL 的笔记——对 ODL 源码的分析和对 ODL 架构设计的理解整理成书，进而帮助大家深入理解 ODL 设计原理和思想，把握其核心源码实现，以不变应万变。这对大家基于 ODL 平台进行业务研发及应用都能有所裨益。

本书的读者对象

本书适合所有有志于解决现有网络问题并促进网络变革的通信、网络及计算机行业相关从业者，特别是希望掌握软件定义网络热门技术的开发人员，还适合具有一定工作经验、关注网络热门技术并希望查漏补缺继续成长的程序员，以及具备一定软件开发能力的网络技术领域从业者。

具备一定的 Java 语言开发基础，了解网络基础知识及分布式系统基础知识将有助于读者更好地理解本书中的内容。

本书的主要内容

对于广大有志于投身网络变革大潮的从业者而言，ODL 依然具有很高的门槛——100 多个子项目，几百万行源代码，OSGi、Maven、Akka、YANG 等背景知识，都成了相关从业者应用 ODL 平台的“拦路虎”。面对这些障碍，我们需要抓住 ODL 平台框架的本质与核心——MD-SAL，只有真正理解 ODL 的这个核心框架设计，理解 MD-SAL 的核心源码实现及其设计思想，才能基于 ODL 进行高效的 SDN 开发实战，就如同习武练功，招式是外壳，内功心法是核心，二者要相辅相成。本书结合作者 15 年的通信软件研发从业经验，从 ODL 核心框架 MD-SAL 的实现源码入手进行解构，剖析代码中的设计模式，总结 ODL 中的软件架构设计思想。本书可谓一本重在讲授内功心法，并指导内功和招式结合的“武功秘籍”。

本书能帮助入门级程序员深入、直观地理解 ODL 技术原理，构建精准的知识框架；帮助有一定工作经验的程序员填补知识漏洞，打通知识体系；帮助正在应用 ODL 构建商用产品和应用的同仁们客观认识并分析 ODL 中现存的问题。本书还分享了作者在基于 ODL 进行商用开发时总结的若干实战经验。

勘误与支持

本书主要内容来源于本人研究学习 ODL 源码的笔记和在应用 ODL 开发项目过程中的实践经验。我们知道 ODL 社区非常活跃，版本发布频繁，架构调整及源码变动也比较大，这不可避免地会导致书中引用的源代码与 ODL 社区最新源代码有一定出入，请读者在阅读过程中务必注意。同时，由于作者写作和认知水平有限，问题在所难免，欢迎读者朋友们通过电子邮箱 yfc@hzbook.com 进行指正。

致谢

ODL 官方社区的源代码是创作本书的原始素材，因此我首先感谢 ODL 官方社区。

其次，我要感谢未来网络学院给予我与大家分享 ODL 技术的平台，以及在开源项目 Jaguar 的成立、运作管理和基础设施资源上的支持。

最后，感谢中兴通讯 IT 学院的闫林老师的鼓励和巨大帮助。

耿兴元

2019 年

目　录 *Contents*

前言

第一部分　基础环境篇

第 1 章　阅读源代码前的准备 …… 2

1.1　ODL 项目介绍 …… 2

1.1.1　ODL 框架之争 …… 3

1.1.2　SAL 的演进 …… 3

1.1.3　ODL 的子项目及分类 …… 4

1.1.4　ODL 项目的管理 …… 6

1.2　搭建 ODL 编译构建环境 …… 6

1.2.1　安装 JDK …… 6

1.2.2　安装及配置 Maven …… 8

1.3　阅读和调试 ODL 源代码 …… 9

1.3.1　ODL 项目源码下载 …… 9

1.3.2　IntelliJ IDEA 安装 …… 10

1.3.3　IntelliJ IDEA 调试 ODL 的项目源码 …… 11

1.4　ODL 设计目标 …… 12

1.5　ODL 总体架构 …… 13

1.6　本章小结 …… 15

第 2 章　ODL 项目管理设计详解 …… 16

2.1　问题的提出 …… 16

2.2　解决思路 …… 17

2.3　实现详解 …… 20

2.3.1　基础 parent 设计 …… 20

2.3.2　模块构建 …… 23

2.3.3　feature 组织 …… 24

2.3.4　版本打包 …… 25

2.4　项目模板 …… 26

2.4.1　项目目录布局设计 …… 26

2.4.2　ODL 模板项目 …… 27

2.5　本章小结 …… 28

第二部分　核心原理篇

第 3 章　ODL 基本对象的设计与实现 …… 30

3.1　QName …… 30

3.1.1　QName 定义 …… 30

3.1.2　QName 对象比较 …… 36

3.1.3　QName 对象创建 …… 37

3.2 YangInstanceIdentifier ····· 38
3.2.1 Path 接口定义 ····· 38
3.2.2 YangInstanceIdentifier 的类定义 ····· 39
3.2.3 YangInstanceIdentifier 的比较 ··· 42
3.2.4 InstanceIdentifier 类 ····· 44
3.3 NomalizedNode ····· 44
3.3.1 NormalizedNode 类的定义 ····· 45
3.3.2 NormalizedNode 实例的创建 ··· 48
3.4 本章小结 ····· 49

第 4 章 数据树的设计与实现 ····· 50

4.1 基本概念 ····· 50
4.1.1 配置树与状态树 ····· 51
4.1.2 标识与定位 ····· 51
4.1.3 快照与 MVCC ····· 52
4.2 数据树的设计与实现 ····· 52
4.2.1 Tree 结构的设计 ····· 52
4.2.2 DataTree 相关接口定义 ····· 55
4.2.3 DataTree 的创建 ····· 57
4.3 数据树的读写过程 ····· 59
4.3.1 快照实现原理 ····· 61
4.3.2 数据校验的实现 ····· 61
4.4 MVCC 机制与实现 ····· 63
4.4.1 版本号变更规则 ····· 63
4.4.2 并发控制 ····· 65
4.5 本章小结 ····· 67

第 5 章 MD-SAL DataStore 接口设计 ····· 68

5.1 基本概念 ····· 69
5.1.1 事务和事务链 ····· 70
5.1.2 数据分片 ····· 70
5.1.3 三阶段提交 ····· 71
5.2 DataStore SPI 设计 ····· 72
5.2.1 DOMStore ····· 73
5.2.2 DOMStoreThreePhase-CommitCohort ····· 75
5.2.3 DOMStoreTreePublisher ····· 76
5.3 DataStore DOM API 设计 ····· 77
5.3.1 DOMDataBroker ····· 77
5.3.2 DOMDataTreeSharding-Service ····· 78
5.3.3 DOMDataTreeChange-Service ····· 80
5.4 DataStore Binding API 设计 ····· 82
5.4.1 Binding 基本对象接口 ····· 82
5.4.2 DataBroker ····· 84
5.4.3 DataTreeChangeService ····· 87
5.5 本章小结 ····· 87

第 6 章 MD-SAL DataStore 的实现原理 ····· 88

6.1 概述 ····· 89
6.1.1 背景知识 ····· 89
6.1.2 实现原理 ····· 91
6.2 Raft 算法及其实现 ····· 92
6.2.1 Raft 算法介绍 ····· 93
6.2.2 RaftActor 设计与实现 ····· 98
6.3 DataStore 后端实现详解 ····· 106
6.3.1 Shard 的实现 ····· 106
6.3.2 ShardManager ····· 110
6.3.3 ShardStrategy 及实现 ····· 112
6.4 DataStore 前端实现详解 ····· 113

6.4.1 DOMStore 的实现 …… 113
6.4.2 DOMDataBroker 的实现 …… 121
6.4.3 事务链实现 …… 124
6.5 Binding DataBroker 的实现 …… 125
6.5.1 Adapter 设计 …… 125
6.5.2 BindingDOMDataBroker-Adapter 的初始化 …… 126
6.6 本章小结 …… 130

第 7 章 MD-SAL RPC 的设计与实现 …… 131

7.1 一个实例 …… 131
7.1.1 RPC 的 YANG 模型定义 …… 131
7.1.2 RPC 的生成接口 …… 133
7.1.3 RPC 的实现与调用 …… 135
7.2 RPC 机制的总体设计 …… 136
7.2.1 Binding 接口设计 …… 136
7.2.2 DOM 接口设计 …… 137
7.2.3 总体实现流程 …… 139
7.3 RPC 机制实现详解 …… 141
7.3.1 DOMBroker 实现详解 …… 141
7.3.2 BindingBroker 实现详解 …… 144
7.4 Remote RPC 实现详解 …… 149
7.4.1 Gossip 协议的实现 …… 150
7.4.2 远程 RPC 注册及调用 …… 152
7.4.3 Actor 设计实现总结 …… 154
7.5 本章小结 …… 155

第 8 章 MD-SAL Notification 的设计与实现 …… 156

8.1 一个实例 …… 156
8.1.1 YANG 模型定义 …… 156
8.1.2 生成的接口 …… 157
8.1.3 消息发布 …… 157
8.1.4 消息订阅 …… 158
8.2 MD-SAL Notification 接口设计 …… 158
8.2.1 DOM 接口 …… 159
8.2.2 Binding 接口 …… 160
8.3 MD-SAL Notification 实现剖析 …… 161
8.3.1 DOM 层实现详解 …… 161
8.3.2 Binding 适配实现 …… 169
8.4 本章小结 …… 171

第 9 章 MD-SAL Mount 机制与 NETCONF …… 172

9.1 Mount 服务接口设计 …… 172
9.1.1 DOM 接口 …… 173
9.1.2 Binding 接口 …… 174
9.2 Mount 机制的实现 …… 175
9.2.1 DOM 接口实现 …… 176
9.2.2 NETCONF 南向插件的实现 …… 178
9.3 本章小结 …… 186

第 10 章 MD-SAL Cluster Service … 187

10.1 EntityOwnershipService …… 187
10.1.1 基本概念 …… 187
10.1.2 接口设计 …… 188
10.1.3 实现说明 …… 192
10.2 ClusterSingletonService …… 195
10.2.1 接口设计 …… 195
10.2.2 实现说明 …… 196
10.3 本章小结 …… 198

第三部分　公共组件篇

第 11 章　AAA …… 200

11.1　Shiro 框架介绍 …… 201

11.1.1　Shiro 是什么 …… 201

11.1.2　Shiro 的架构 …… 202

11.1.3　Shiro 核心处理流程 …… 204

11.2　AAA 实现原理 …… 210

11.2.1　Shiro 配置优化 …… 210

11.2.2　Realm 的 8 个实现 …… 212

11.2.3　Filter 的实现 …… 214

11.2.4　加解密服务 …… 216

11.2.5　数字证书管理 …… 218

11.3　本章小结 …… 219

第 12 章　RESTCONF …… 220

12.1　RFC 8040 解读 …… 220

12.1.1　操作 …… 221

12.1.2　消息 …… 222

12.1.3　资源 …… 223

12.2　RESTCONF 的实现 …… 226

12.2.1　Jersey 框架简介 …… 226

12.2.2　RESTCONF 资源接口定义 …… 228

12.2.3　Wrapper 设计模式 …… 231

12.2.4　初始化过程 …… 233

12.2.5　客户端访问 …… 235

12.3　本章小结 …… 236

第 13 章　Blueprint 及其扩展 …… 237

13.1　Blueprint …… 238

13.1.1　基础知识 …… 238

13.1.2　运行原理 …… 240

13.1.3　命名空间扩展 …… 241

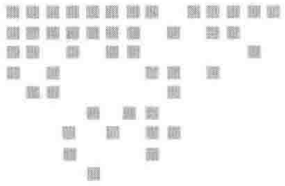
13.2　Blueprint 的使用 …… 244

13.3　本章小结 …… 247

第一部分 Part 1

基础环境篇

- 第 1 章　阅读源代码前的准备
- 第 2 章　ODL 项目管理设计详解



Chapter 1 第 1 章

阅读源代码前的准备

OpenDaylight（简称 ODL）是在 2013 年由 18 家网络巨头发起成立的，旨在推动 SDN 发展及促进网络领域创新的开源控制器平台项目。ODL 项目的目标是成为 SDN 领域的开发、运行、创新的框架和平台。开源项目自有其天生的优势和劣势，也必须遵循自身的发展演进的规律。开源项目优势可总结为具有兼容并包的框架性设计，聚集了众多贡献者大胆创新的思想，版本迭代快速等。开源项目的劣势也很明显，那就是开源项目一般缺乏系统性、完善的文档，部署、升级和运维监控等方面考虑的也都有所欠缺，所以自身很难提供针对具体场景的端到端的商用解决方案。另外，对于一个比较活跃的开源项目，需要 5～7 年才能做到框架的成熟和稳定。ODL 项目成立至今已有 6 年多，我们可以说其核心框架已经稳定和成熟了。因此，理解并掌握 ODL 的最新核心框架的设计与实现原理，对于我们利用 ODL 这个框架和平台进行商用产品的研发是有很大帮助的。

本章主要为后续章节做准备，将指导大家搭建 ODL 的编译开发环境，也会向大家简单介绍 ODL 的诸多子项目以及这些项目的组织管理方式，并对 ODL 核心的 MD-SAL 框架的设计目标和设计原则做一个简单的介绍，让大家对 ODL 项目及其核心架构有一个整体的了解。

1.1 ODL 项目介绍

ODL 项目成立之初，发起成立该项目的众多网络巨头都拿出了自己在 SDN 领域或网络虚拟化方面的一些项目代码贡献到 ODL 社区，包括 Cisco 的 OnePk（ODL 中的 controller

子项目）、Big Switch 的控制器与网络虚拟化项目（ODL 中的 OSCP 子项目）、IBM 的 DOVE（ODL 中的 OpenDove 子项目）、Radware 的安全防御系统（ODL 中的 Defence4All 子项目）、NEC 的 Virtual Tenent Network（ODL 中 VTN 子项目的）等。这些项目本身都是由各公司自己开发的，有些还是已经商用的项目。虽然这些项目本身质量还可以，但是因为缺乏统一的框架和设计思路，有些项目还具有竞争性质，导致这些项目没法组合成一个有机的整体，因此，我们看到的 ODL 的初始发布版本（氢版本），就是一个拼凑起来的大杂烩。随着近年来的不断完善，ODL 逐渐形成了统一的架构和合理的项目层次。

1.1.1　ODL 框架之争

在最初的 ODL 众多子项目中，Cisco 主导的 controller 子项目和 Big Switch 主导的 OSCP 项目代表了 SDN 控制器框架设计的两种思路，二者属于竞争关系。

controller 项目中，基于平台化的设计思路，提出了业务抽象层（Service Abstraction Layer，SAL）的概念，把控制器分为南向（SB）协议插件、北向（NB）业务 / 应用插件和 SAL 三层。通过 SAL 层，实现了多种南向协议插件与多样化的业务应用的解耦，这种设计借鉴了早期的 SDN 开源控制器 Beacon 的基于 OSGi 的模块化框架设计，符合 ONF 提出的典型的 SDN 三层架构，每一层都暴露大量的 API 供不同层次的业务应用进行调用，使系统具有很强的扩展性和灵活性。

OSCP 项目中，主要采用了产品化的设计思路，南向协议绑定在 OpenFlow 上，提供集成度高的、功能比较完善的网络基础功能管理，但这将导致整个系统围绕着 OpenFlow 协议紧紧地耦合在一起，限制了控制器自身的扩展性与适用性。

当然，我们知道，Cisco 的思路在 ODL 社区最终胜出，成为 ODL 的核心框架。而 Big Switch 与 Cisco 在 ODL 社区角力失败也导致了其在 ODL 社区运作了还不到 3 个月的时间就退出了。

1.1.2　SAL 的演进

最初的 SAL 是以 AD-SAL（API-Driven SAL）为主的，也就是需要分别定义南向插件的 Java 编程接口（API）和业务应用的 Java 编程接口（API），并需要为两种接口做大量的适配编码。另外，北向的 REST 接口也要手工定义。显而易见的，作为一个开放的开发平台来说，增加一个新的插件功能，就需要通过手工编码来定义 SAL API 和适配代码，使其具有很大局限性。一方面 API 的定义非常困难，因为这需要非常专业的网络领域的知识，还

要考虑规范性、通用性和扩展性；另一方面，大量的手工编码对于开发者来说非常不友好。因此，从 ODL 的氦（Hellium）版本和锂（Lithium）版本开始，一直到氟（Fluorine）版本，MD-SAL（Model-Driven SAL）架构逐步发展并成熟，而 AD-SAL 从铍（Berryllium）版本即被完全废弃。截至现在，ODL 社区加入了越来越多的基于 MD-SAL 框架设计的南向协议类的项目和应用类项目（几十个），而 ODL 成立时建立的一些不太符合该设计思路的项目，比如 OSCP、Defence4All、OpenDove 等现在已经被社区归档，不再继续维护。

1.1.3 ODL 的子项目及分类

MD-SAL 架构采用 YANG 语言作为数据及接口的建模语言，通过 YangTools 工具提供了编译期数据模型与接口的解析和代码的自动生成，采用 Binding Broker、Binding-Independent Broker 两套接口实现了运行期的数据模型与接口的适配转换，简化并规范了南向插件与业务应用间调用接口的定义，为南向插件和业务应用的开发者提供了统一的开发模式。MD-SAL 框架的主体代码原本在 controller 和 yangtools 两个项目中，特别的是，controller 项目最初不仅包含 SAL 框架，还包括配置子系统、netconf/restconf、版本公共配置及版本打包、项目模板等功能。随着 ODL 项目的发展演进，这些功能有的单独成立子项目（mdsal、odlparent、netconf/restconf、archetypes），有的被废弃（配置子系统、AD-SAL），当前仍保留在 controller 项目中的功能代码主要是 ODL 的分布式集群功能（分布式 datastore 和 remoterpc）和 blueprint 扩展。以上 controller、mdsal、odlparent、netconf/restconf 等子项目加上 yangtools 项目基本上就构成了 ODL 的核心框架，再加上 ODL 的鉴权框架 AAA 项目，这些子项目就是 ODL 的核心项目。除了决定 ODL 基础架构的这些核心项目，ODL 的项目中还包括协议类项目、业务应用类项目及支持类项目。协议类项目包括多种南向协议插件项目如 OpenFlow、OVSDB、NETCONF、BGP、BMP、PCEP、LISP、SNMP、P4、SXP、OCP、P4、Telemetry 等，还包括北向的 RESTCONF、NEMO Intent 等。业务应用类项目包括 NetVirt、COE、FaaS、BIER APP、DetNet、SFC、Transport PCE、IOT 等，支持类项目包括 Documentation、Inte-gration、RelEng 等。图 1-1 显示了 ODL 社区现在的活跃的项目及项目间的依赖关系。

注意 因为 ODL 社区中不断接受申请成立新的子项目，原来的子项目也会根据情况进行调整，甚至被废弃而归档，因此图 1-1 中的项目不一定与最新的 ODL 子项目一一对应，但从几个核心项目来说，图 1-1 还算准确。