title: 精尽 Dubbo 源码分析 —— 项目结构一览 date: 2018-01-04 tags: categories: Dubbo permalink: Dubbo/intro

---

摘要: 原创出处 http://www.iocoder.cn/Dubbo/intro/ 「芋道源码」欢迎转载，保留摘要，谢谢！

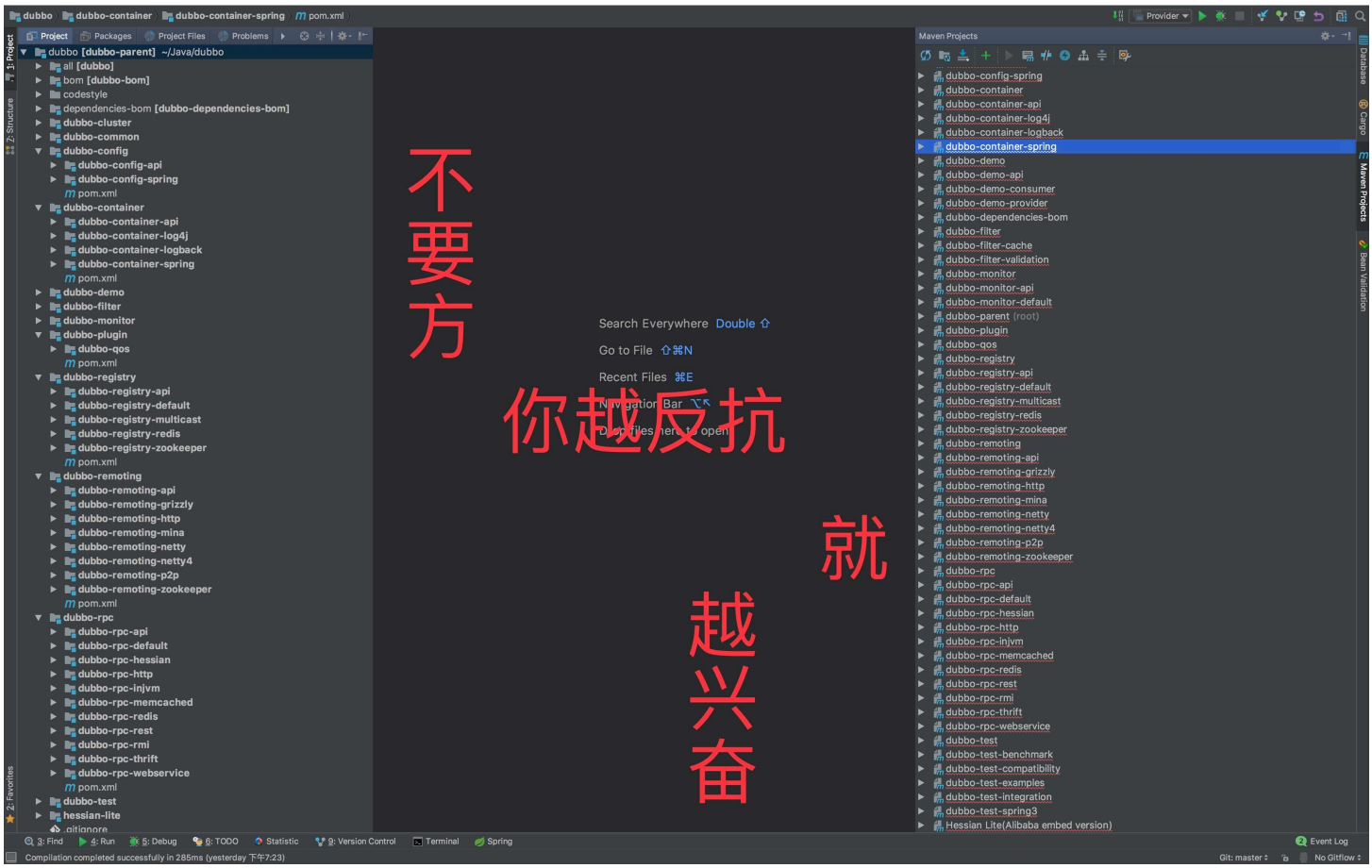
---

1. 概述

本文主要分享 **Dubbo 的项目结构**。
希望通过本文能让胖友对 Dubbo 的整体项目有个简单的了解。

另外，笔者会相对大量引用 《Dubbo 用户指南》 和 《Dubbo 开发指南》 ，写的真的挺好的。🙂
ps：限于排版，部分地方引用会存在未标明的情况。

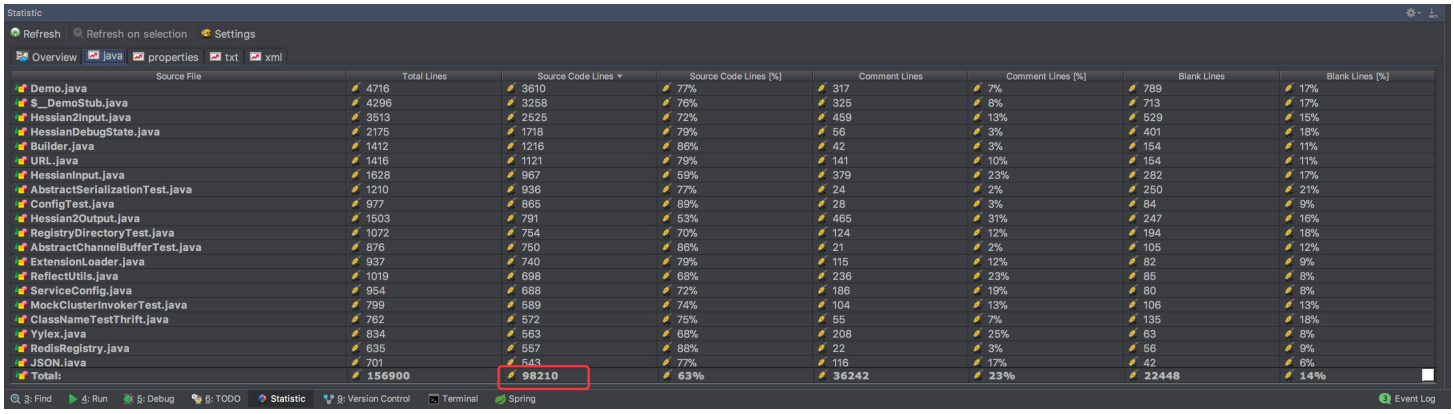
在拉取 Dubbo 项目后，我们会发现拆分了**好多** Maven 项目。是不是内心一紧，产生了恐惧感？不要方，我们就是继续怼。



2. 代码统计

这里先分享一个小技巧。笔者在开始源码学习时，会首先了解项目的代码量。

第一种方式，使用 IDEA Statistic 插件，统计整体代码量。



Source File	Total Lines	Source Code Lines	Source Code Lines [%]	Comment Lines	Comment Lines [%]	Blank Lines	Blank Lines [%]
Demo.java	4716	3610	77%	317	7%	789	17%
$__DemoStub.java	4296	3258	76%	325	8%	713	17%
Hessian2Input.java	3513	2525	72%	459	13%	529	15%
HessianDebugState.java	2175	1718	79%	56	3%	401	18%
Builder.java	1412	1216	86%	42	3%	154	11%
URL.java	1416	1121	79%	141	10%	154	11%
HessianInput.java	1628	967	59%	379	23%	282	17%
AbstractSerializationTest.java	1210	936	77%	24	2%	250	21%
ConfigTest.java	977	865	89%	28	3%	84	9%
Hessian2Output.java	1503	791	53%	465	31%	247	16%
RegistryDirectoryTest.java	1072	754	70%	124	12%	194	18%
AbstractChannelBufferTest.java	876	750	86%	21	2%	105	12%
ExtensionLoader.java	937	740	79%	115	12%	82	9%
ReflectUtils.java	1019	698	68%	236	23%	85	8%
ServiceConfig.java	954	688	72%	186	19%	80	8%
MockClusterInvokerTest.java	799	589	74%	104	13%	106	13%
ClassNameTestThrift.java	762	572	75%	55	7%	135	18%
Yylex.java	834	563	68%	208	25%	63	8%
RedisRegistry.java	635	557	88%	22	3%	56	9%
JSON.java	701	543	77%	116	17%	42	6%
Total:	156900	98210	63%	36242	23%	22448	14%

我们可以粗略的看到，总的代码量在 98210 行。这其中还包括单元测试，示例等等代码。所以，不慌。

第二种方式，使用 Shell 脚本命令逐个 Maven 模块统计 。

一般情况下，笔者使用 `find . -name "*.java"|xargs cat|grep -v -e ^$ -e ^\s*\/\/.*$|wc -`

`l` 。这个命令只过滤了**部分注释**，所以相比 IDEA Statistic 会**偏多**。

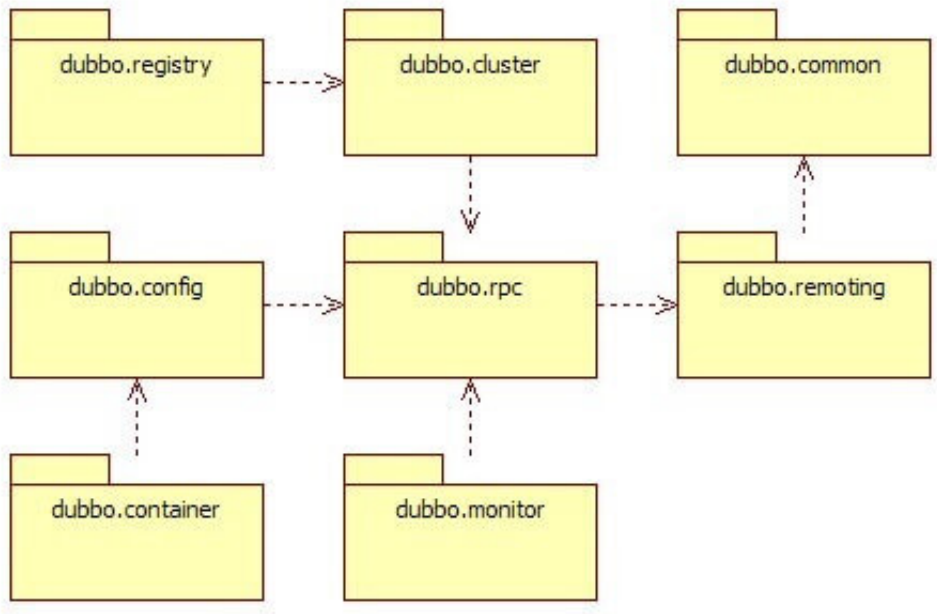
当然，考虑到准确性，胖友需要手动 `cd` 到每个 Maven 项目的 `src/main/java` 目录下，以达到排除单元测试的代码量。

Maven 第一级	Package 第一级	代码量1	代码量2	代码是否阅读	博客是否覆盖
dubbo-cluster		3957			
dubbo-common		23757			
dubbo-config-api		5867			
dubbo-config-spring		3346			
dubbo-container-api		140			
dubbo-container-log4j		98			
dubbo-container-logback		87			
dubbo-container-spring		61			
dubbo-demo-api					
dubbo-demo-consumer					
dubbo-demo-provider					
dubbo-filter-cache		434			
dubbo-filter-validation		529			
dubbo-monitor-api		426			
dubbo-monitor-default		458			
dubbo-plugin-qos		2137			
dubbo-registry-api		2792			
dubbo-registry-default		247			
dubbo-registry-multicast		461			
dubbo-registry-redis		669			
dubbo-registry-zookeeper		325			
dubbo-remoting-api		9195			
dubbo-remoting-grizzly		682			
dubbo-remoting-http		650			
dubbo-remoting-mina		640			
dubbo-remoting-netty		1139			
dubbo-remoting-netty4		1707			
dubbo-remoting-p2p		1434			
dubbo-remoting-zookeeper		695			
dubbo-rpc-api		5445			
	filter		1416		
	listener		292		
	protocol		651		
	proxy		366		
	service		128		
	support		582		

	[root] invocation		2010		
dubbo-rpc-default		3182			
dubbo-rpc-hessian		283			
dubbo-rpc-http		156			
dubbo-rpc-injvm		208			
dubbo-rpc-memcached		117			
dubbo-rpc-redis		168			
dubbo-rpc-rest		900			
dubbo-rpc-rmi		161			
dubbo-rpc-thrift		1416			
dubbo-rpc-webservice		151			
dubbo-test-benchmark		1232			
dubbo-test-compatibility		163			
dubbo-test-examples		2746			
dubbo-test-integration		0			
hessian-lite		18518			

3. 项目一览

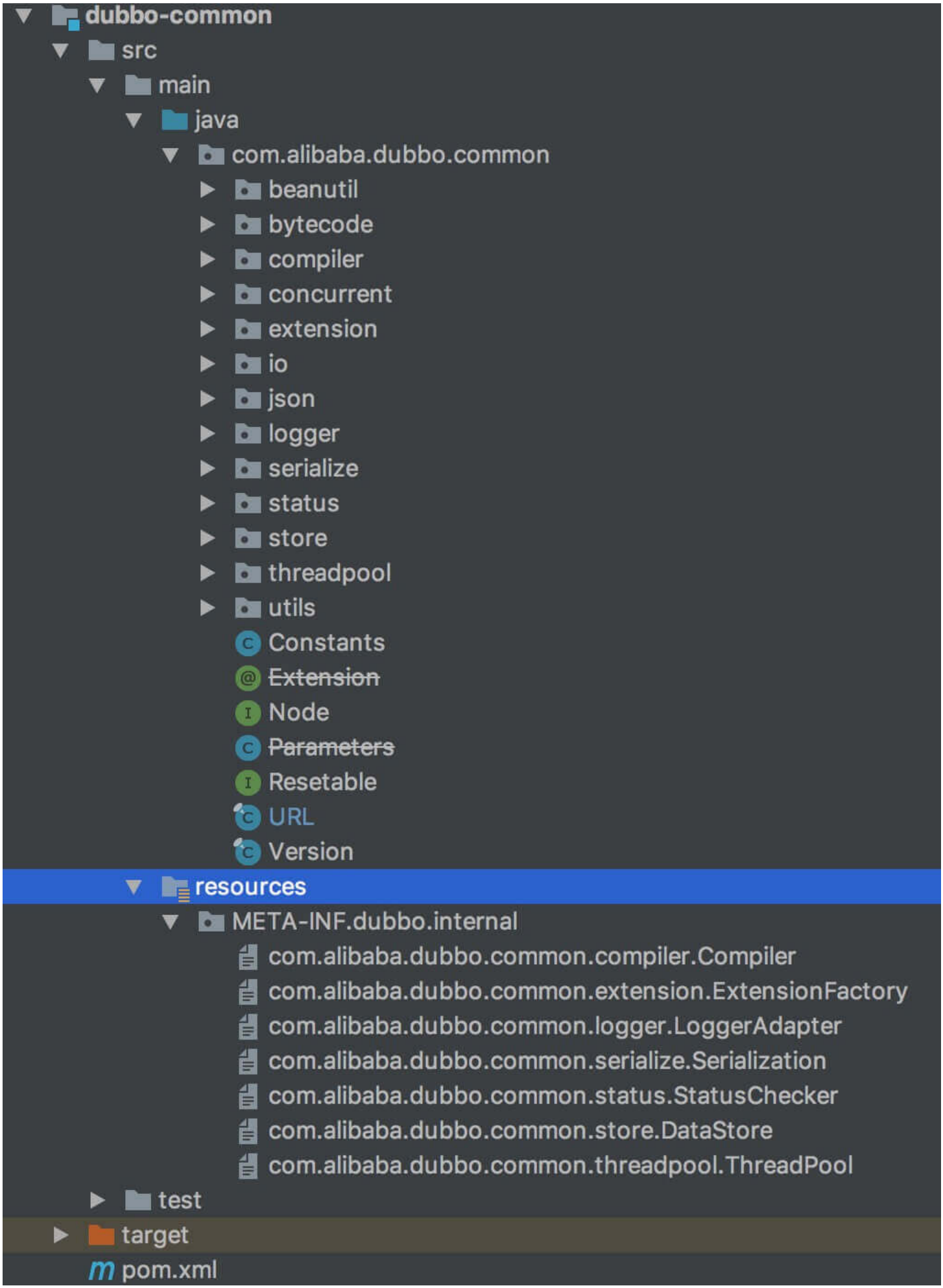
如果胖友看过 《Dubbo 框架设计》 ，就会发现有下面这张图。

> 
>

通过这图，我们可以很清晰的知道几个 Maven 模块的依赖关系。

3.1 dubbo-common

> `dubbo-common` **公共逻辑模块**：提供工具类和通用模型。



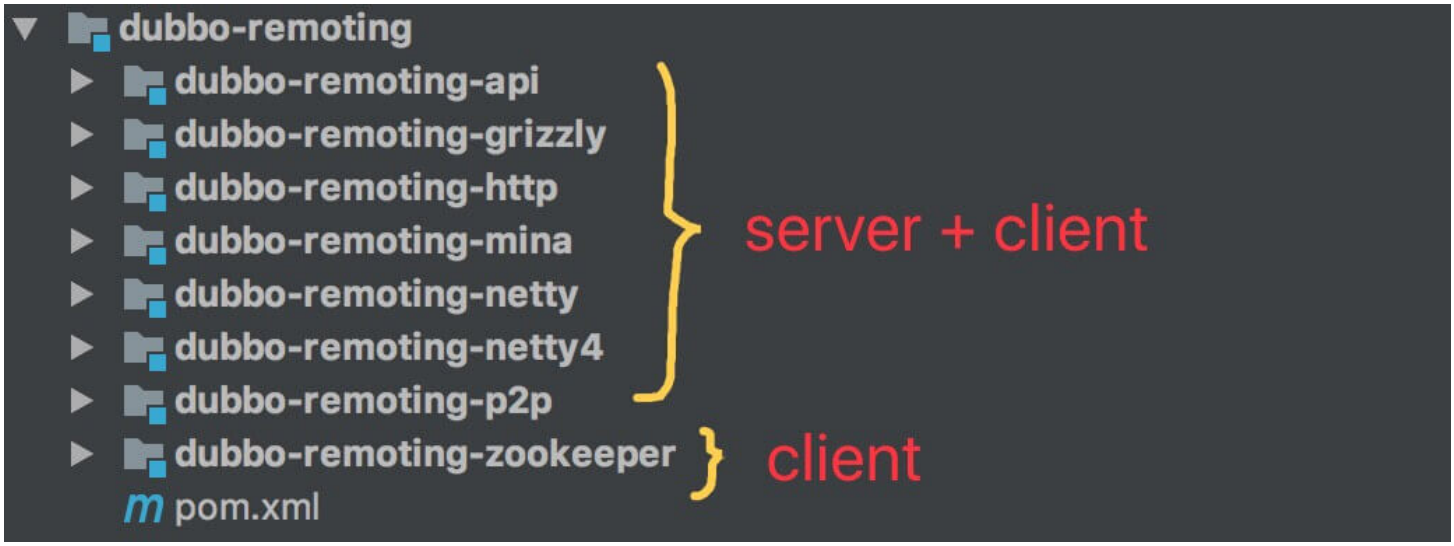
工具类比较好理解，**通用模型**是什么？举个例子，`com.alibaba.dubbo.common.URL`：

> FROM 《Dubbo 开发指南 —— 公共契约》
>
> - 所有扩展点参数都包含 URL 参数，URL 作为上下文信息贯穿整个扩展点设计体系。
> - URL 采用标准格式：`protocol://username:password@host:port/path?key=value&key=value` 。

那么 URL 有什么用呢? 😈 请见后续文章。

3.2 dubbo-remoting

> `dubbo-remoting` **远程通信模块**：提供**通用**的客户端和服务端的通讯功能。



- `dubbo-remoting-zookeeper` ，相当于 Zookeeper Client ，和 Zookeeper Server 通信。
- `dubbo-remoting-api` ， 定义了 Dubbo Client 和 Dubbo Server 的**接口**。
- **实现** `dubbo-remoting-api`
  - `dubbo-remoting-grizzly` ，基于 Grizzly 实现。
  - `dubbo-remoting-http` ，基于 Jetty 或 Tomcat 实现。
  - `dubbo-remoting-mina` ，基于 Mina 实现。
  - `dubbo-remoting-netty` ，基于 Netty 3 实现。
  - `dubbo-remoting-netty4` ，基于 Netty 4 实现。
  - `dubbo-remoting-p2p` ， 【TODO 8000】dubbo-remoting-p2p

从**最小化**的角度来看，我们只需要看：

- `dubbo-remoting-api` + `dubbo-remoting-netty4`
- `dubbo-remoting-zookeeper`

3.3 dubbo-rpc

> `dubbo-rpc` **远程调用模块**：抽象各种协议，以及动态代理，只包含一对一的调用，**不关心集群的管理**。

- 集群相关的管理，由 `dubbo-cluster` 提供特性。

在回过头看上面的图，我们会发现，`dubbo-rpc` 是整个 Dubbo 的**中心**。

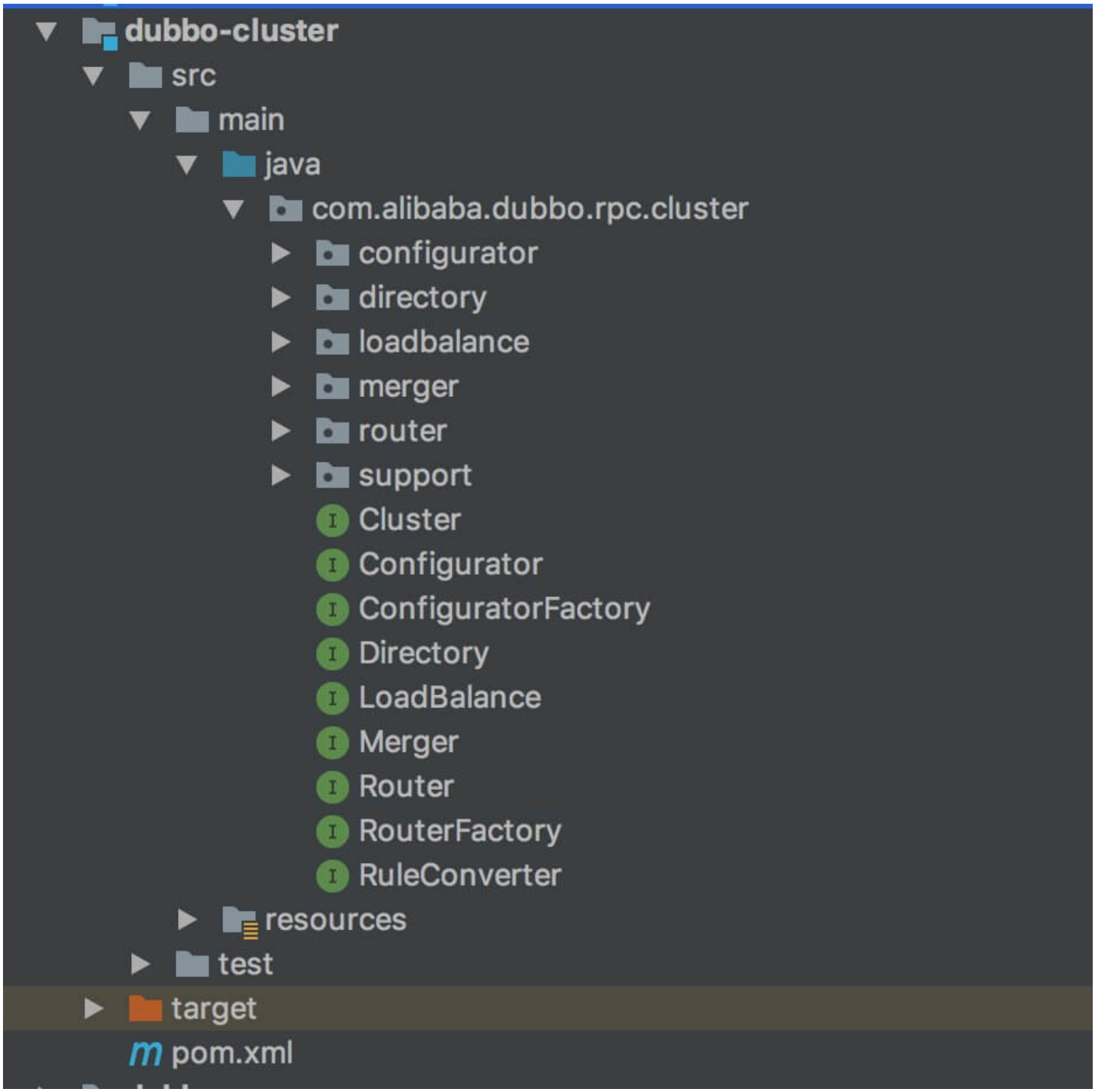
- `dubbo-rpc-api` ，**抽象**各种协议以及动态代理，**实现**了一对一的调用。
- 其他模块，实现 `dubbo-rpc-api` ，提供对应的协议实现。在 《用户指南 —— 协议参考手册》 中，可以看到每种协议的介绍。
- 另外，`dubbo-rpc-default` 对应 `dubbo://` 协议。
- 拓展参见 《Dubbo 开发指南 —— 协议扩展》 文档。

进一步的拆解，见 《精尽 Dubbo 源码分析 —— 核心流程一览》 文章。

3.4 dubbo-cluster

> `dubbo-cluster` **集群模块**：将多个服务提供方伪装为一个提供方，包括：负载均衡, 集群容错，路由，分组聚合等。集群的地址列表可以是静态配置的，也可以是由注册中心下发。

- 注册中心下发，由 `dubbo-registry` 提供特性。

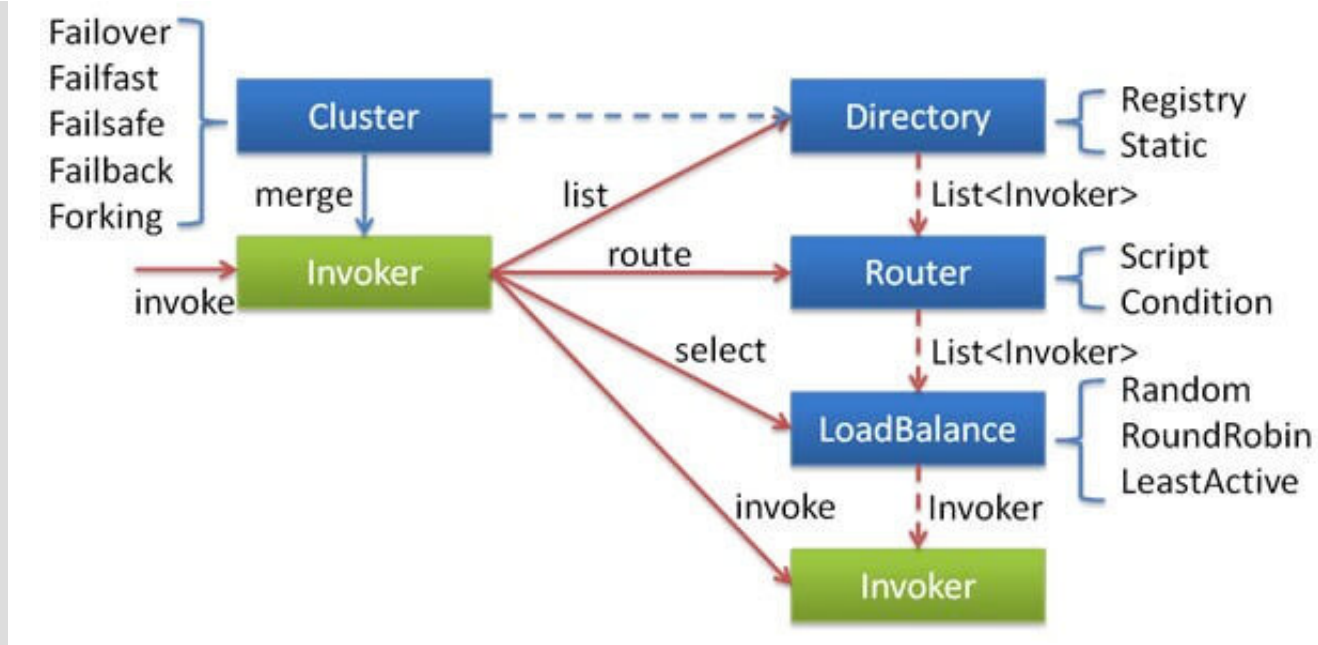


- 容错
  - `com.alibaba.dubbo.rpc.cluster.Cluster` 接口 + `com.alibaba.dubbo.rpc.cluster.support` 包。
  - Cluster 将 Directory 中的多个 Invoker 伪装成一个 Invoker，对上层透明，伪装过程包含了容错逻辑，调用失败后，重试另一个。
  - 拓展参见 《Dubbo 用户指南 —— 集群容错》 和 《Dubbo 开发指南 —— 集群扩展》 文档。
- 目录
  - `com.alibaba.dubbo.rpc.cluster.Directory` 接口 + `com.alibaba.dubbo.rpc.cluster.directory` 包。
  - Directory 代表了多个 Invoker ， 可以把它看成 List ， 但与 List 不同的是，它的值可能

是动态变化的，比如注册中心推送变更。

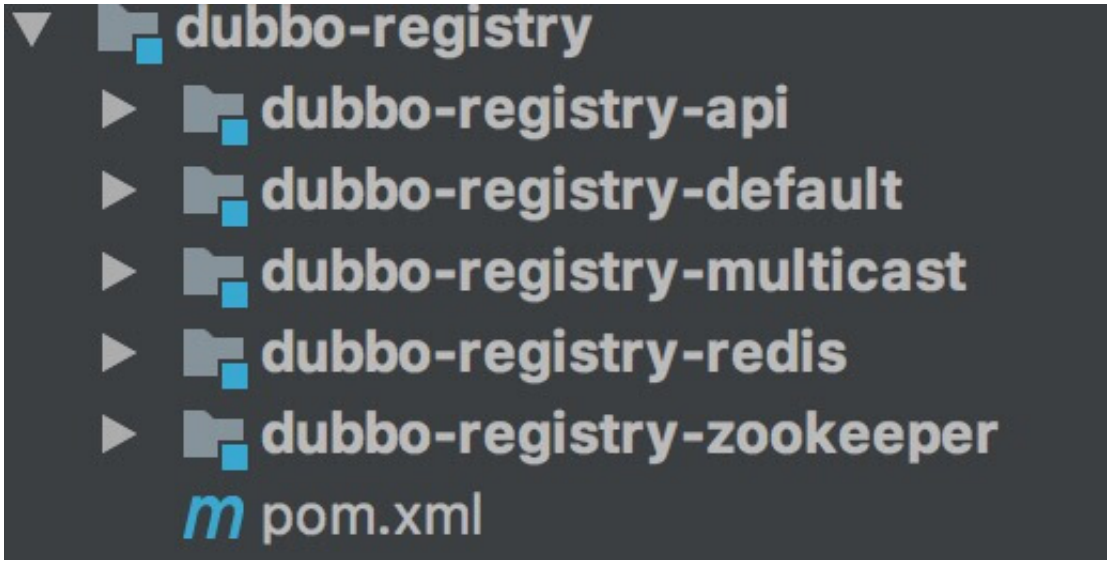
- 路由
  - `com.alibaba.dubbo.rpc.cluster.Router` 接口 + `com.alibaba.dubbo.rpc.cluster.router` 包。
  - 负责从多个 `Invoker` 中按路由规则选出子集，比如读写分离，应用隔离等。
  - 拓展参见 《Dubbo 用户指南 —— 路由规则》 和 《Dubbo 开发指南 —— 路由拓展》 文档。
- 配置
  - `com.alibaba.dubbo.rpc.cluster.Configurator` 接口 + `com.alibaba.dubbo.rpc.cluster.configurator` 包。
  - 拓展参见 《Dubbo 用户指南 —— 配置规则》 文档。
- 负载均衡
  - `com.alibaba.dubbo.rpc.cluster.LoadBalance` 接口 + `com.alibaba.dubbo.rpc.cluster.loadbalance` 包。
  - LoadBalance 负责从多个 Invoker 中选出具体的一个用于本次调用，选的过程包含了负载均衡算法，调用失败后，需要重选。
  - 拓展参见 《Dubbo 用户指南 —— 负载均衡》 和 《Dubbo 开发指南 —— 负载均衡拓展》 文档。
- 合并结果
  - `com.alibaba.dubbo.rpc.cluster.Merger` 接口 + `com.alibaba.dubbo.rpc.cluster.merger` 包。
  - 合并返回结果，用于分组聚合。
  - 拓展参见 《Dubbo 用户指南 —— 分组聚合》 和 《Dubbo 开发指南 —— 合并结果扩展》 文档。

整体流程如下：



3.5 dubbo-registry

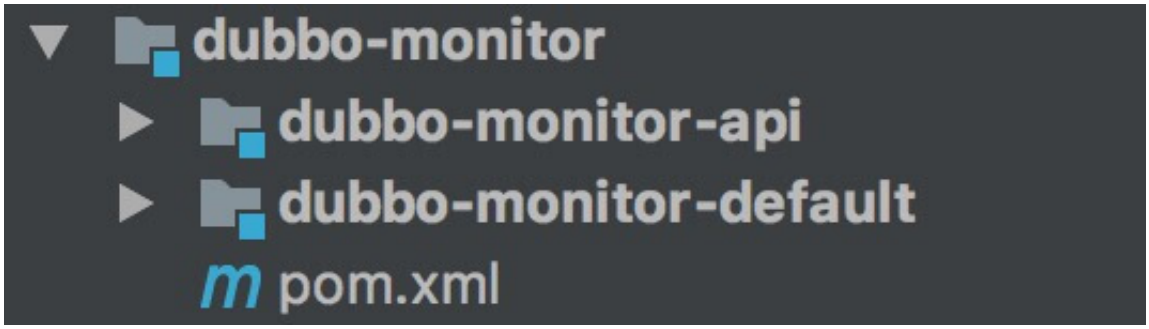
> `dubbo-registry` **注册中心模块**：基于注册中心下发地址的集群方式，以及对各种注册中心的抽象。



- `dubbo-registry-api` ，**抽象**注册中心的注册与发现接口。
- 其他模块，实现 `dubbo-registry-api` ，提供对应的注册中心实现。在 《用户指南 —— 注册中心参考手册》 中，可以看到每种注册中心的介绍。
- 另外，`dubbo-registry-default` 对应 Simple 注册中心。
- 拓展参见 《Dubbo 开发指南 —— 注册中心扩展》 文档。

3.6 dubbo-monitor

> `dubbo-monitor` **监控模块**：统计服务调用次数，调用时间的，调用链跟踪的服务。



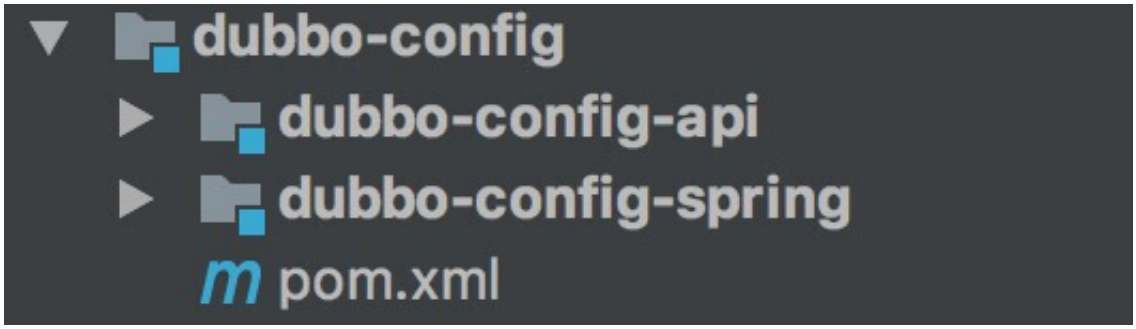
- 拓展参见 《Dubbo 开发指南 —— 监控中心扩展》 。

目前社区里，有对 Dubbo 监控中心进行重构的项目，例如：

- https://github.com/handuyishe/dubbo-monitor
- https://github.com/zhongxig/dubbo-d-monitor

3.7 dubbo-config

> `dubbo-config` **配置模块**：是 Dubbo 对外的 API，用户通过 Config 使用Dubbo，隐藏 Dubbo 所有细节。

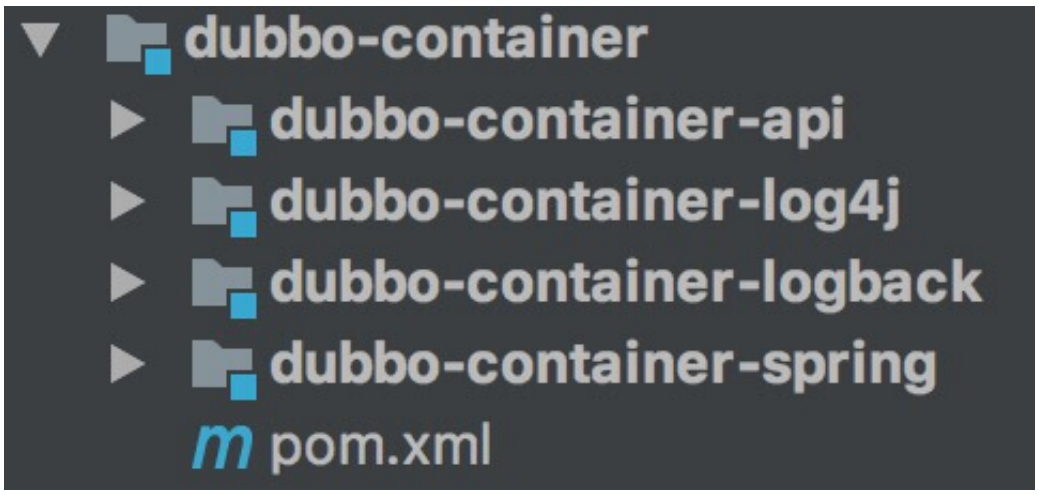


- `dubbo-config-api` ，实现了 API 配置 和 属性配置 功能。
- `dubbo-config-spring` ，实现了 XML 配置 和 注解配置 功能。

推荐阅读 《Dubbo 开发指南 —— 配置设计》 。

3.8 dubbo-container

> `dubbo-container` **容器模块**：是一个 Standlone 的容器，以简单的 Main 加载 Spring 启动，因为服务通常不需要 Tomcat/JBoss 等 Web 容器的特性，没必要用 Web 容器去加载服务。

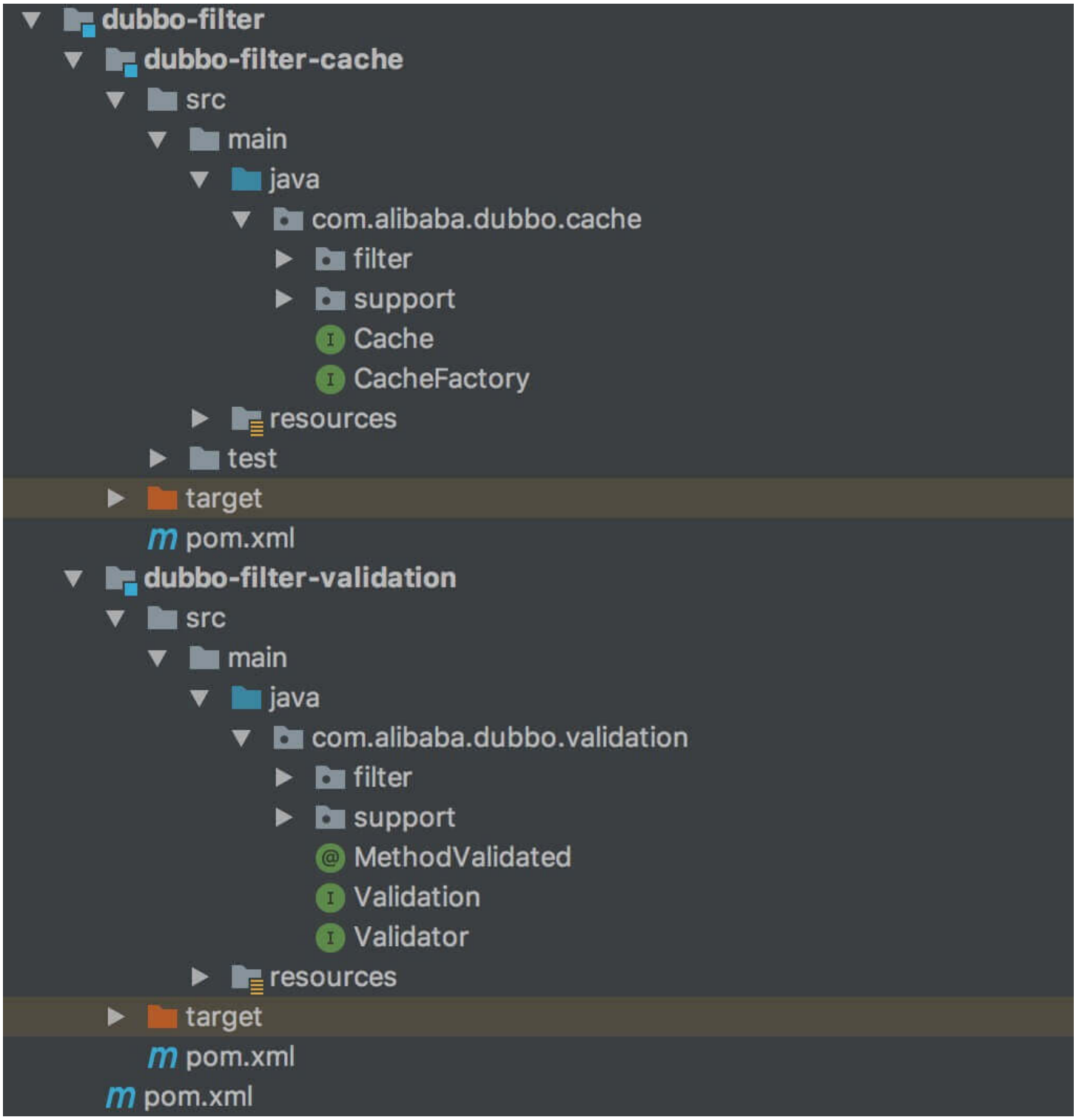


- `dubbo-container-api` ：定义了 `com.alibaba.dubbo.container.Container` 接口，并提供 加载所有容器启动的 Main 类。
- 实现 `dubbo-container-api`
  - `dubbo-container-spring` ，提供了 `com.alibaba.dubbo.container.spring.SpringContainer` 。
  - `dubbo-container-log4j` ，提供了

`com.alibaba.dubbo.container.log4j.Log4jContainer` 。
    - `dubbo-container-logback` ，提供了 `com.alibaba.dubbo.container.logback.LogbackContainer` 。
- 拓展参考 《Dubbo 用户指南 —— 服务容器》 和 《Dubbo 开发指南 —— 容器扩展》 文档。

3.9 dubbo-filter

`dubbo-filter` **过滤器模块**：提供了**内置**的过滤器。



- `dubbo-filter-cache` ，缓存过滤器。
  - 拓展参考 《Dubbo 用户指南 —— 结果缓存》 和 《Dubbo 开发指南 —— 缓存拓展》 文档。
- `dubbo-filter-validation` ，参数验证过滤器。
  - 拓展参考 《Dubbo 用户指南 —— 参数验证》 和 《Dubbo 开发指南 —— 验证扩展》 文档。

3.10 dubbo-plugin

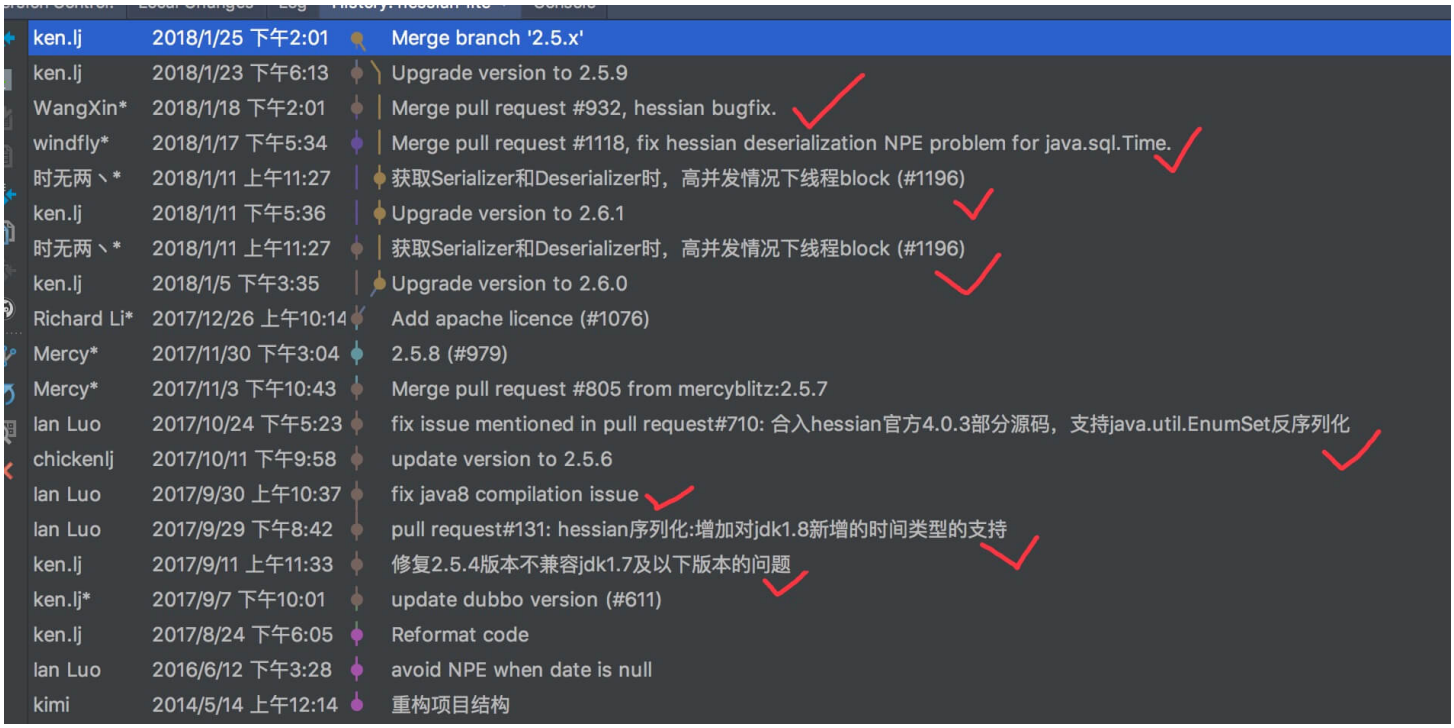
`dubbo-plugin` **过滤器模块**：提供了**内置**的插件。

- `dubbo-qos` ，提供在线运维命令。
  - 拓展参考 《Dubbo 用户指南 —— 新版本 telnet 命令使用说明》 和 《Dubbo 开发指南 —— Telnet 命令扩展》 文档。

3.11 hessian-lite

`hessian-lite` ： Dubbo 对 Hessian 3 的精简、改进、BugFix 。

提交历史如下：



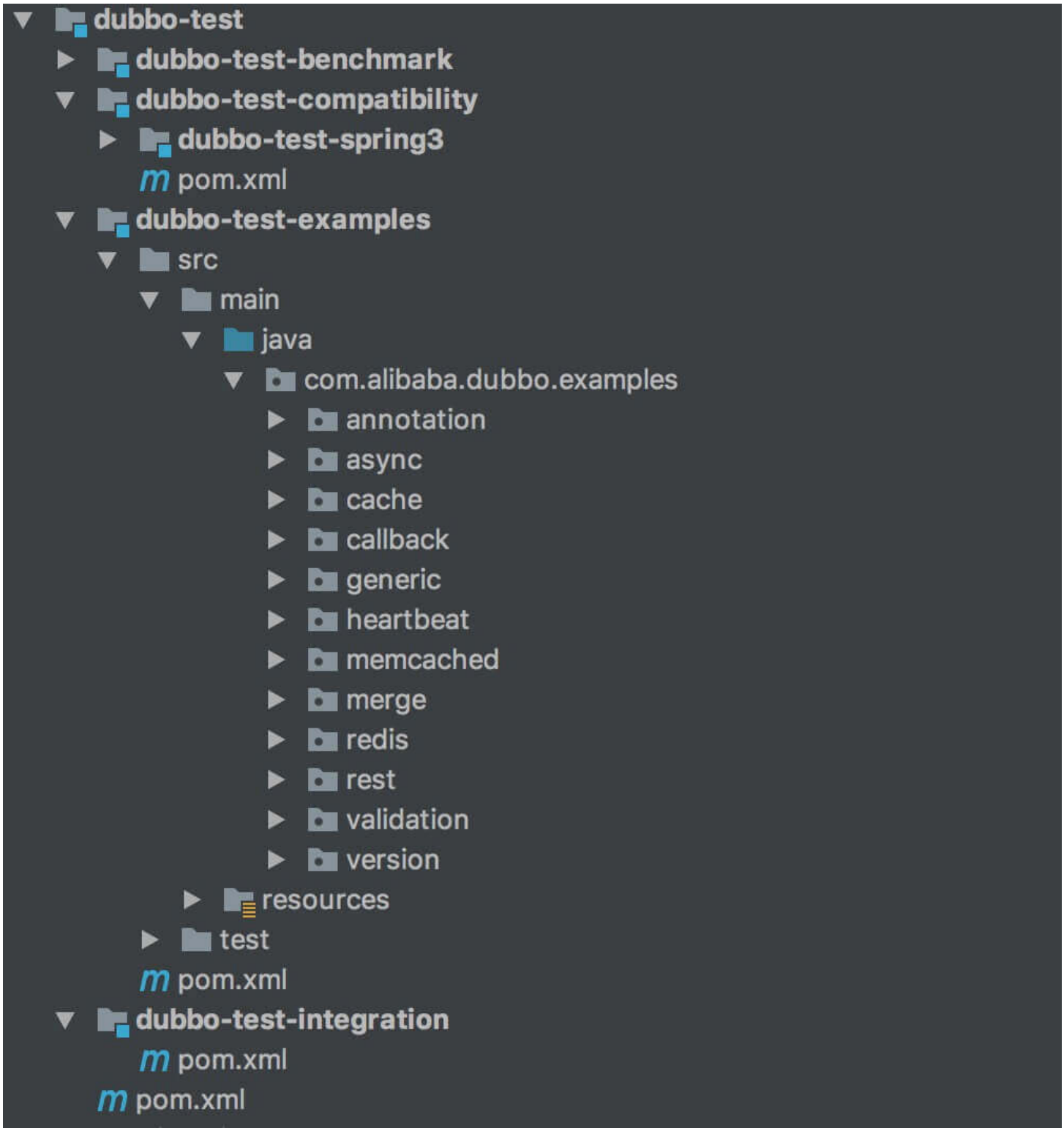
3.12 dubbo-demo

`dubbo-demo` **快速启动示例。**

参见 《Dubbo 用户指南 —— 快速启动》 文档。

3.13 dubbo-test

`dubbo-test` **测试模块。**



- `dubbo-test-benchmark` ，性能测试。
  - 参考 《Dubbo 用户指南 —— 性能测试报告》 文档。
- `dubbo-test-compatibility` ，兼容性测试。
  - `dubbo-test-spring3` ，测试对 Spring 3 的兼容性。
- `dubbo-test-example` ，**使用示例**。

3.14 Maven POM

3.14.1 dubbo-dependencies-bom

`dubbo-dependencies-bom/pom.xml`，Maven BOM(Bill Of Materials)，**统一**定义了 Dubbo 依赖的三方库的版本号：

```
    <properties>
        <!-- Common libs -->
        <spring_version>4.3.10.RELEASE</spring_version>
        <javassist_version>3.20.0-GA</javassist_version>
        <netty_version>3.2.5.Final</netty_version>
        <netty4_version>4.0.35.Final</netty4_version>
        <mina_version>1.1.7</mina_version>
        <grizzly_version>2.1.4</grizzly_version>
        <httpclient_version>4.5.3</httpclient_version>
        <fastjson_version>1.2.46</fastjson_version>
        <zookeeper_version>3.4.9</zookeeper_version>
        <zkclient_version>0.2</zkclient_version>
        <curator_version>2.12.0</curator_version>
        <jedis_version>2.9.0</jedis_version>
        <xmemcached_version>1.3.6</xmemcached_version>
        <cxf_version>3.0.14</cxf_version>
        <thrift_version>0.8.0</thrift_version>
        <hessian_version>4.0.38</hessian_version>
        <servlet_version>3.1.0</servlet_version>
        <jetty_version>6.1.26</jetty_version>
        <validation_version>1.1.0.Final</validation_version>
        <hibernate_validator_version>5.4.1.Final</hibernate_validator_version
        <jel_version>3.0.1-b08</jel_version>
        <jcache_version>1.0.0</jcache_version>
        <kryo_version>4.0.1</kryo_version>
        <kryo_serializers_version>0.42</kryo_serializers_version>
        <fst_version>2.48-jdk-6</fst_version>

        <rs_api_version>2.0</rs_api_version>
        <resteasy_version>3.0.19.Final</resteasy_version>
        <tomcat_embed_version>8.0.11</tomcat_embed_version>
        <!-- Log libs -->
        <slf4j_version>1.7.25</slf4j_version>
        <jcl_version>1.2</jcl_version>
        <log4j_version>1.2.16</log4j_version>
        <logback_version>1.2.2</logback_version>
        <commons_lang3_version>3.4</commons_lang3_version>
    </properties>

    <dependencyManagement...>
```

`dubbo-parent` 会引入该 BOM：

dubbo-parent

```xml
92
93      <dependencyManagement>
94          <dependencies>
95              <dependency>
96                  <groupId>com.alibaba</groupId>
97                  <artifactId>dubbo-dependencies-bom</artifactId>
98                  <version>2.6.1</version>
99                  <type>pom</type>
100                 <scope>import</scope>
101             </dependency>
102         </dependencies>
103     </dependencyManagement>
```

更多 Maven BOM 的知识，可以看下 《Maven 与Spring BOM(Bill Of Materials)简化Spring版本控制》 文档：

> 通俗解说： 为了防止用 Maven 管理 Spring 项目时，不同的项目依赖了不同版本的 Spring ，可以使用 Maven BOM 来解决者一问题。

3.14.2 dubbo-bom

`dubbo-bom/pom.xml` ， Maven BOM(Bill Of Materials) ， **统一定义了** Dubbo 的版本号：

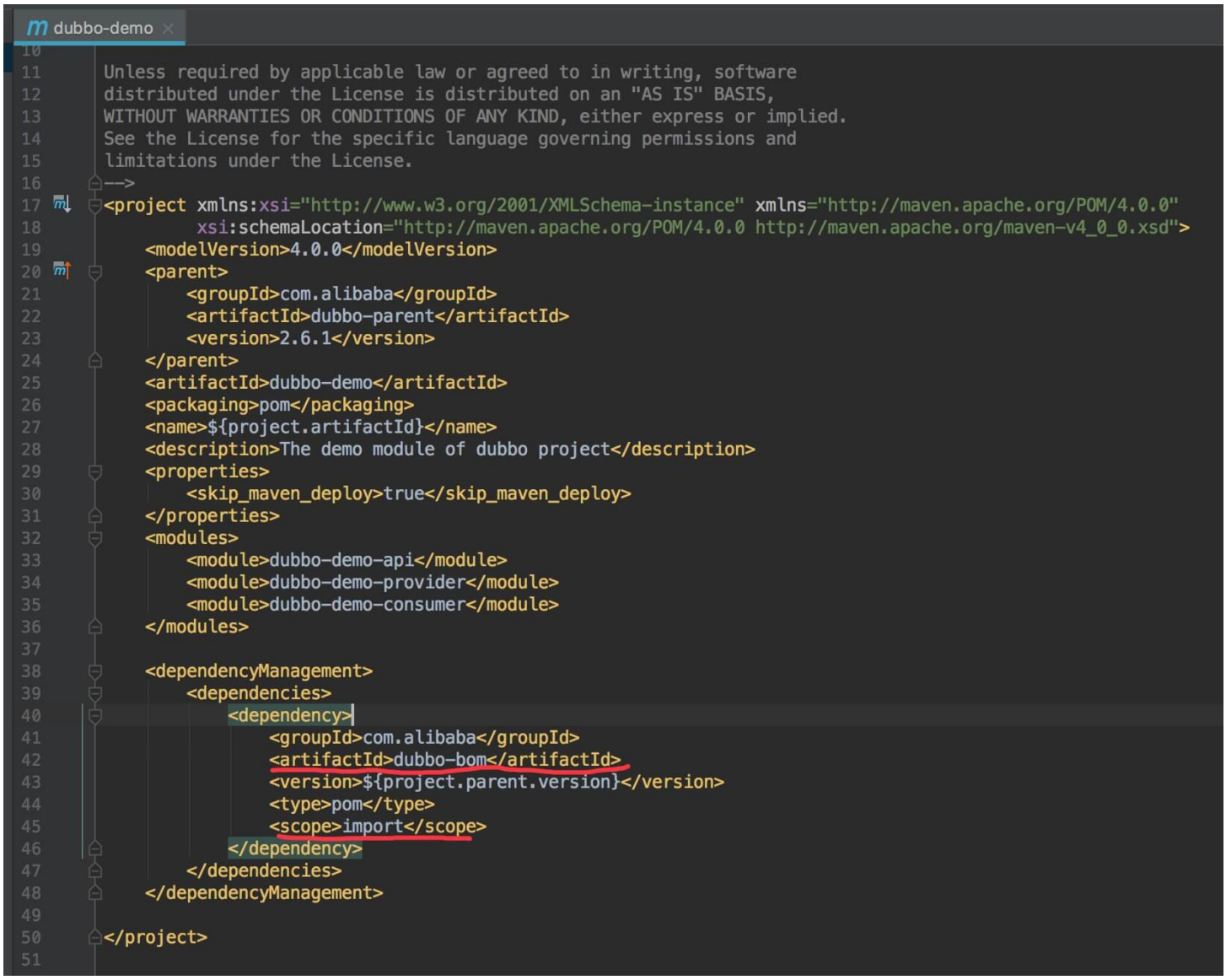
dubbo-bom

```xml
79      <dependencyManagement>
80          <dependencies>
81              <dependency>
82                  <groupId>com.alibaba</groupId>
83                  <artifactId>dubbo-cluster</artifactId>
84                  <version>${project.version}</version>
85              </dependency>
86              <dependency>
87                  <groupId>com.alibaba</groupId>
88                  <artifactId>dubbo-common</artifactId>
89                  <version>${project.version}</version>
90              </dependency>
91              <dependency>
92                  <groupId>com.alibaba</groupId>
93                  <artifactId>dubbo-config-api</artifactId>
94                  <version>${project.version}</version>
95              </dependency>
96              <dependency>
97                  <groupId>com.alibaba</groupId>
98                  <artifactId>dubbo-config-spring</artifactId>
99                  <version>${project.version}</version>
00              </dependency>
01              <dependency>
02                  <groupId>com.alibaba</groupId>
03                  <artifactId>dubbo-filter-cache</artifactId>
04                  <version>${project.version}</version>
```

```xml
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo-filter-validation</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo-remoting-api</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo-remoting-netty</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo-remoting-netty4</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo-remoting-mina</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo-remoting-grizzly</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo-remoting-p2p</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo-remoting-http</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
```

project

`dubbo-demo` 和 `dubbo-test` 会引入该 BOM 。以 `dubbo-demo` 举例子：



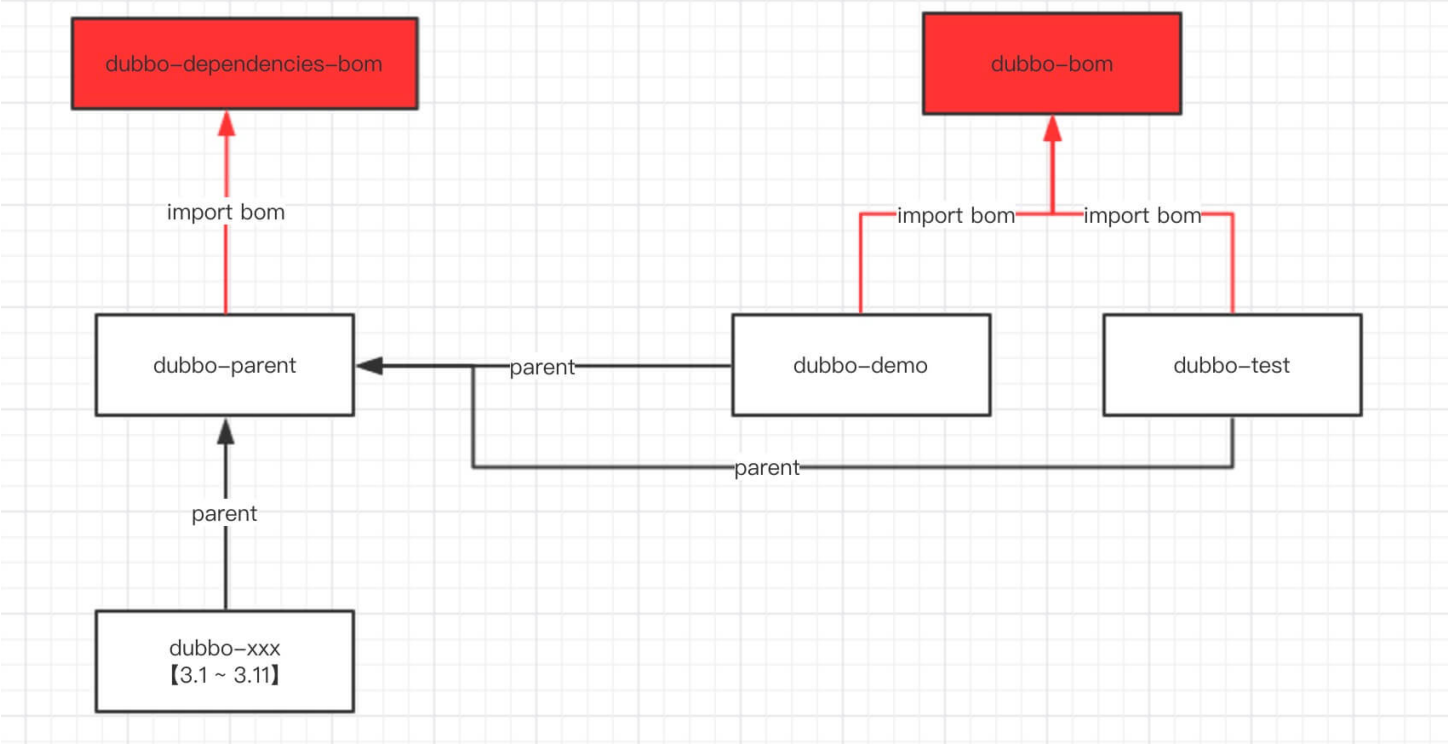
3.14.3 dubbo-parent

`dubbo/pom.xml` ，Dubbo Parent Pom 。

Dubbo 的 Maven 模块，都会引入该 pom 文件。以 `dubbo-cluster` 举例子：

```xml
<!--...-->
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://mave
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo-parent</artifactId>
        <version>2.6.1</version>
    </parent>
    <artifactId>dubbo-cluster</artifactId>
    <packaging>jar</packaging>
    <name>${project.artifactId}</name>
    <description>The cluster module of dubbo project</description>
    <properties...>
    <dependencies...>
</project>
```

我们整理下上面的 pom 文件：



3.14.4 dubbo-all

`dubbo/all/pom.xml` ，Dubbo All Pom ， 定义了 Dubbo 的**打包脚本**。

我们在使用 Dubbo 库时，引入该 pom 文件。

666. 彩蛋

看完本文，是不是对 Dubbo 的整体项目结构，有了一丢丢的整理理解。
笔者推荐，再详细阅读下 《Dubbo 开发指南 —— 框架设计》 ，重新梳理下。

参考文章：

- 《Dubbo 用户指南》
- 《Dubbo 开发指南》