- NETTY

精尽 Netty 源码分析 —— 调试环境搭建

1. 依赖工具

- Maven
- Git
- JDK
- IntelliJ IDEA

2. 源码拉取

从官方仓库 https://github.com/netty/netty `Fork` 出属于自己的仓库。为什么要 `Fork` ？既然开始阅读、调试源码，我们可能会写一些注释，有了自己的仓库，可以进行自由的提交。😈

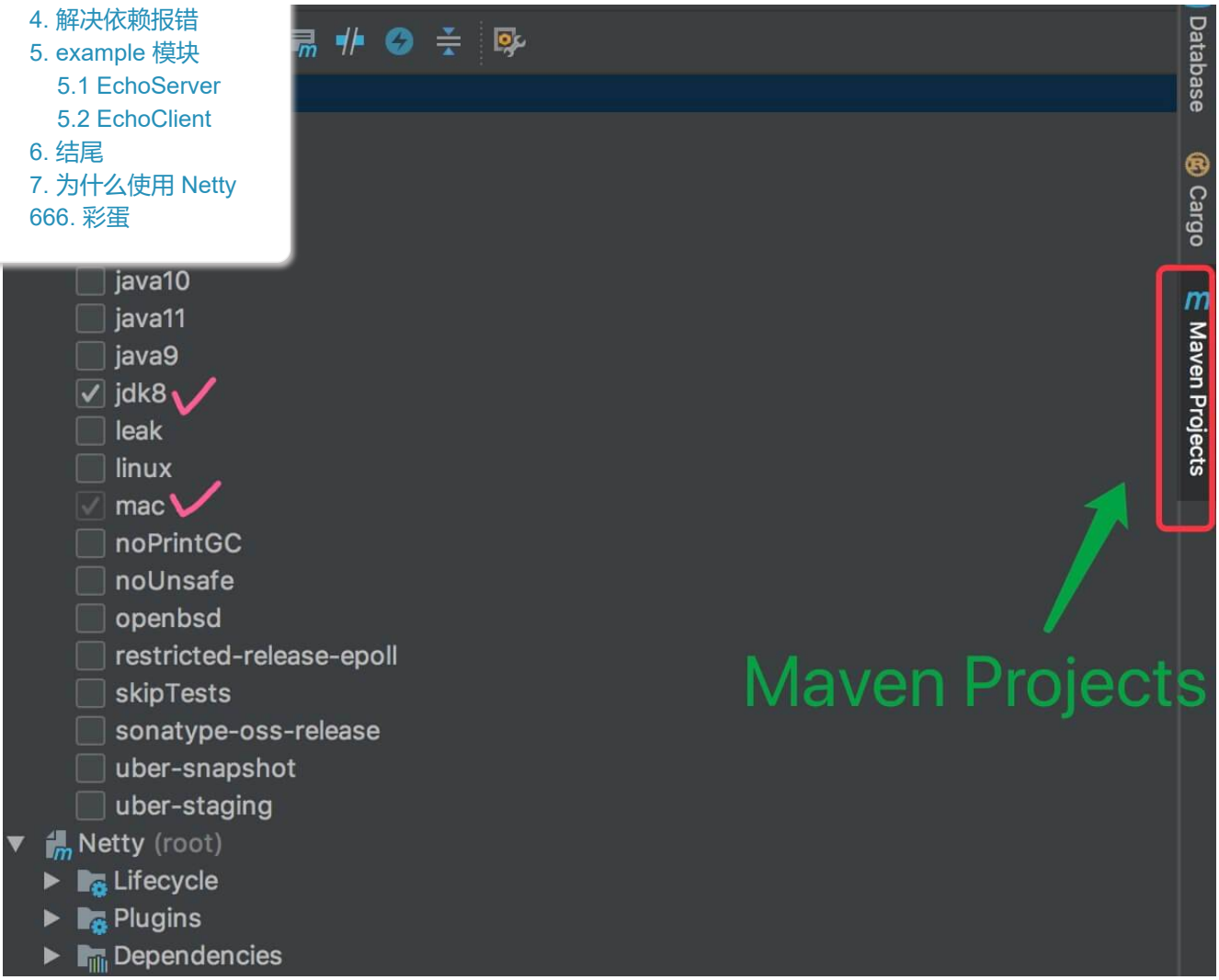
使用 `IntelliJ IDEA` 从 `Fork` 出来的仓库拉取代码。

本文使用的 Netty 版本为 `4.1.26.Final-SNAPSHOT` 。

3. Maven Profile

打开 IDEA 的 **Maven Projects** ，选择对应的 Profiles 。如下图所示：

文章目录



- `jdk8` ：笔者使用的 JDK 版本是 8，所以勾选了 `jdk8` 。如果错误的选择，可能会报如下错误：

```
java.lang.NoSuchMethodError: java.nio.ByteBuffer.clear()Ljava/nio/ByteBuffer
```

- `mac` ：选择对应的系统版本。😈 笔者手头没有 windows 的电脑，所以不知道该怎么选。

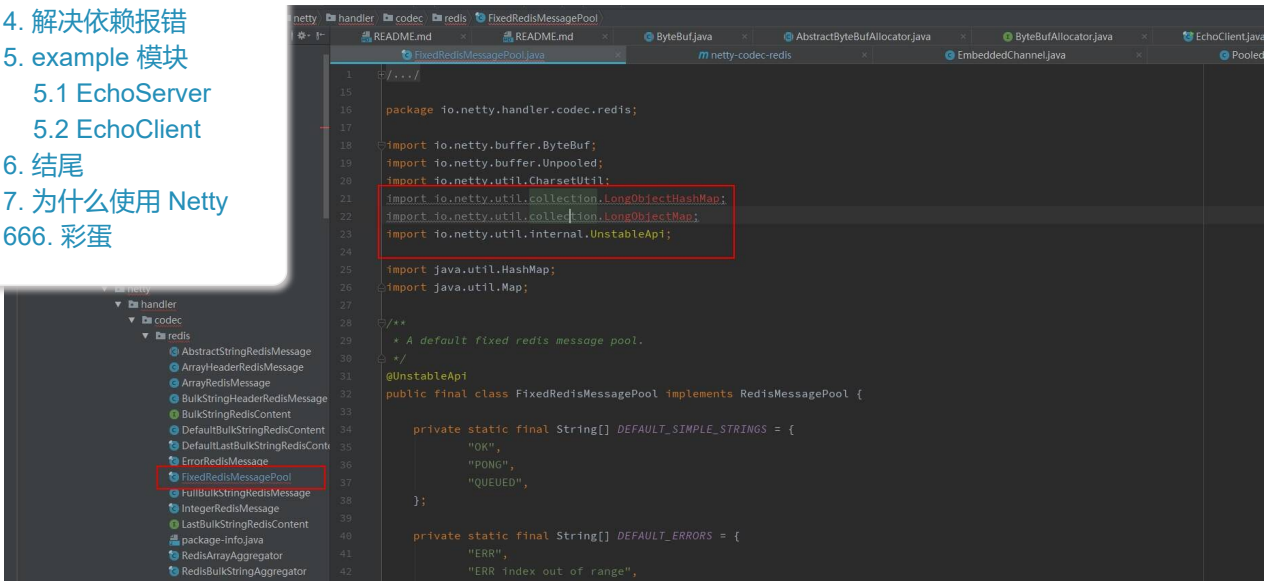
修改完成后，点击左上角的【刷新】按钮，进行依赖下载，耐心等待…

4. 解决依赖报错

在 `codec-redis` 模块中，类 FixedRedisMessagePool 会报如下类不存在的问题：

```
import io.netty.util.collection.LongObjectHashMap;
import io.netty.util.collection.LongObjectMap;
```

- 具体如下图所示：



解决方式如下：

```
cd common;
mvn clean compile;
```

- 跳转到 `common` 模块中，编译生成对应的类。为什么可以通过编译生成对应的类呢，原因参见 `common` 模块的 `src/java/templates/io/netty/util/collection` 目录下的 `.template` 文件。

在 Github 上，也有多个针对这个情况讨论的 issue：

- 《Can not find class io.netty.util.collection.LongObjectHashMap in 4.1.8.final》
- 《io.netty.util.collection.LongObjectHashMap not found at branch 4.1》

5. example 模块

在 `example` 模块里，官网提供了多个 Netty 的使用示例。
本文以 `echo` 包下来作为示例。哈哈哈，因为最简单。

5.1 EchoServer

执行 `io.netty.example.echo.EchoServer` 的 `#main(args)` 方法，启动服务端。输出日志如下：

```
20:41:41.215 [nioEventLoopGroup-2-1] INFO  i.n.handler.logging.LoggingHandler - [id: 0xd0219f1c] REGIS
20:41:41.222 [nioEventLoopGroup-2-1] INFO  i.n.handler.logging.LoggingHandler - [id: 0xd0219f1c] BIND:
20:41:41.228 [nioEventLoopGroup-2-1] INFO  i.n.handler.logging.LoggingHandler - [id: 0xd0219f1c, L:/0:
```

5.2 EchoClient

执行 `io.netty.example.echo.EchoClientr` 的 `#main(args)` 方法，启动客户端。**不输出任何日志。**

下日志：

```
opGroup-2-1] INFO  i.n.handler.logging.LoggingHandler - [id: 0xd0219f1c, L:/0:
opGroup-2-1] INFO  i.n.handler.logging.LoggingHandler - [id: 0xd0219f1c, L:/0:
```

如此，我们就可以愉快的进行 Netty 调试啦。读源码，一定要多多调试源码。非常重要！！！

另外，也推荐基友 zhisheng 的文章：《Netty系列文章（一）：Netty 源码阅读之初始环境搭建》。

7. 为什么使用 Netty

如下内容，引用自我的基友闪电侠的分享

> FROM 《Netty 源码分析之服务端启动全解析》
>
> netty 底层基于 jdk 的 NIO，我们为什么不直接基于 jdk 的 nio 或者其他 nio 框架？下面是我总结出来的原因
>
> 1. 使用 jdk 自带的 nio 需要了解太多的概念，编程复杂
> 2. netty 底层 IO 模型随意切换，而这一切只需要做微小的改动
> 3. netty 自带的拆包解包，异常检测等机制让你从nio的繁重细节中脱离出来，让你只需要关心业务逻辑
> 4. netty 解决了 jdk 的很多包括空轮训在内的 bug
> 5. netty 底层对线程，selector 做了很多细小的优化，精心设计的 reactor 线程做到非常高效的并发处理
> 6. 自带各种协议栈让你处理任何一种通用协议都几乎不用亲自动手
> 7. netty 社区活跃，遇到问题随时邮件列表或者 issue
> 8. netty 已经历各大rpc框架，消息中间件，分布式通信中间件线上的广泛验证，健壮性无比强大

666. 彩蛋

笔者开始更新 Netty 源码解析系列，让我们在 2018 一起**精尽** Netty 。