当客户端多个线程并发请求时，框架内部会调用 DefaultFuture 对象的 get 方法进行等待。在请求发起时，框架内部会创建 Request 对象，这个时候会被分配一个唯一 id，DefaultFuture 可以从 Request 对象中获取 id，并将关联关系存储到静态 HashMap 中，就是图 6-3 中的 Futures 集合。当客户端收到响应时，会根据 Response 对象中的 id，从 Futures 集合中查找对应 DefaultFuture 对象，最终会唤醒对应的线程并通知结果。客户端也会启动一个定时扫描线程去探测超时没有返回的请求。

6.3 编解码器原理

6.2 节主要给出了 Dubbo 目前的协议格式，有了标准协议约束，我们需要再探讨 Dubbo 是怎么实现编解码的。在讲解编解码实现前，先熟悉一下编解码设计关系，如图 6-4 所示。

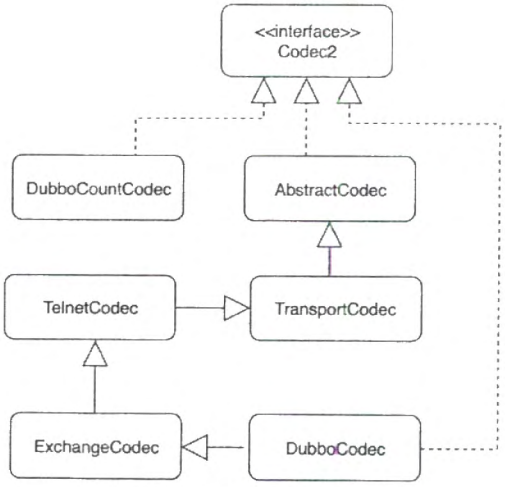

图 6-4 Dubbo 编解码关系

在图 6-4 中，AbstractCodec 主要提供基础能力，比如校验报文长度和查找具体编解码器等。TransportCodec 主要抽象编解码实现，自动帮我们去调用序列化、反序列实现和自动 cleanup 流。我们通过 Dubbo 编解码继承结构可以清晰看到，DubboCodec 继承自 ExchangeCodec，它又再次继承了 TelnetCodec 实现。我们前面说过 Telnet 实现复用了 Dubbo 协议端口，其实就是在这层编解码做了通用处理。因为流中可能包含多个 RPC 请求，Dubbo 框架尝试一次性读取更多完整报文编解码生成对象，也就是图中的 DubboCountCodec，它的实现思想比较简单，依次调用 DubboCodec 去解码，如果能解码成完整报文，则加入消息列表，然后触发下一个 Handler 方法调用。

6.3.1 节和 6.3.2 节会详细介绍编码器和解码器的实现。