**cache服务器的分析与设计**

**一、分析**

游戏服务器使用memcached来对一些数据进行加速访问，我们需要使用libmemcached库来与memcached服务器进行交互，由于libmemcached库与memcached服务器的交互都是同步操作，如果在游戏逻辑中直接使用libmemcached进行操作，由于同步问题，会阻塞服务器的其它逻辑，造成服务器性能大幅度下降，所以服务器如果需要支持memcached，就需要解决同步问题，让libmemcached的同步操作不影响原本服务器的正常逻辑，也就是让逻辑与memcached服务器的交互是异步的、非阻塞的。

三种实现方法：

1. 使用memcached协议自己开发一个非阻塞的memcached客户端，然后就可以在逻辑代码中直接使用。

这种方法开发成本最高，所以暂不考虑。

1. 在游戏服务器中开一个或若干个memcached线程，在这些线程内部使用libmemcached库来和memcached服务器进行阻塞式地交互，而逻辑线程和memcached线程之间是非阻塞式的交互，从而达到在逻辑上和memcached服务器进行非阻塞式异步交互的目的。

这种方法是轻量级的方式，需要在底层加入一组线程，开发成本应该是相对最低的，扩展性会差一点，如果对memcached的请求量比较大的时候，可能对效率会有所影响。

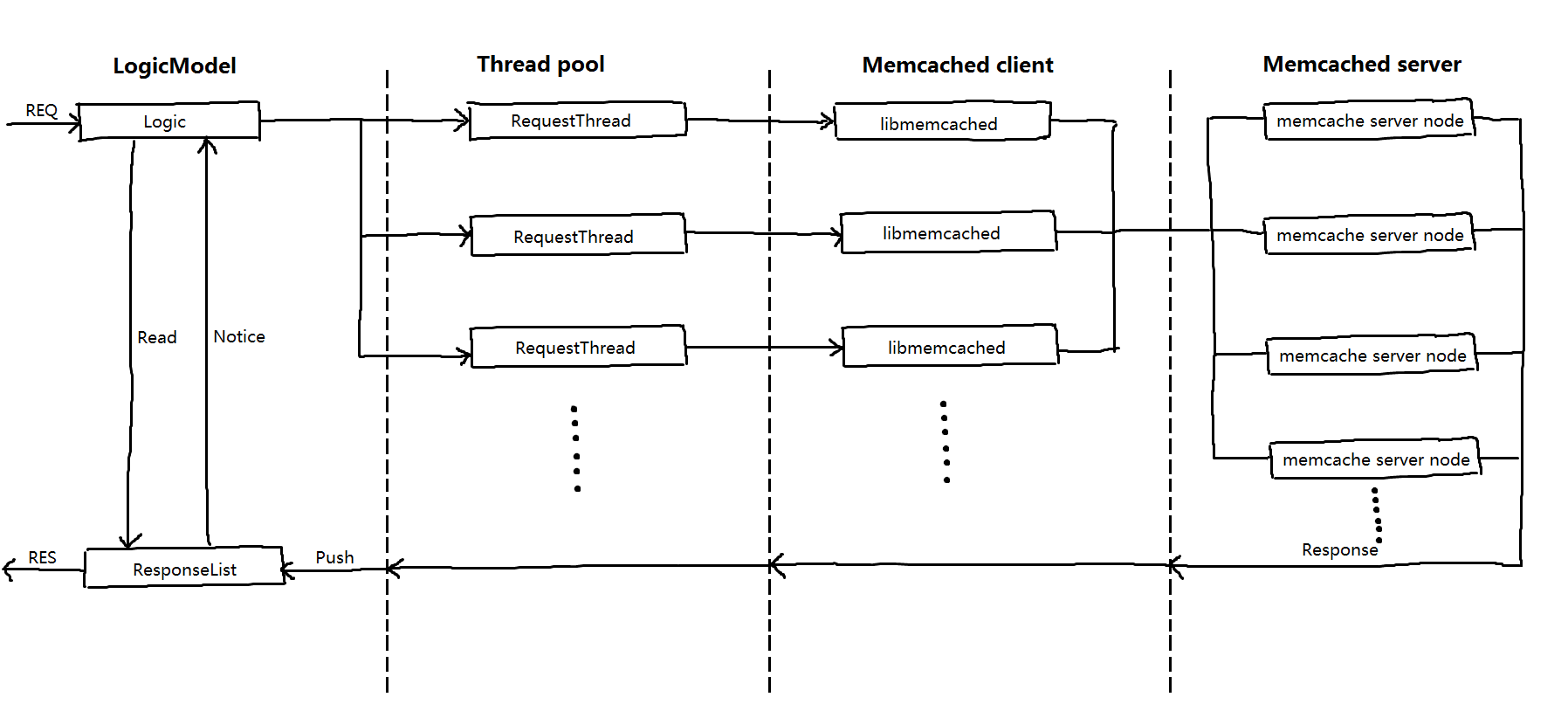
1. 在服务器架构中加入memcached代理服务器，在这台服务器内部使用libmemcached库来和memcached服务器进行阻塞式地交互，其它服务器（主要是游戏服务器）与这台代理服务器之间进行非阻塞式异步交互（类似于现在架构中的数据库前端服务器）。

这种方法是重量级的方式，需要新加入代理服务器，也需要对其它服务器逻辑进行修改，开发成本相比2要高一些，但安全性较高、对现有代码的影响很小，扩展性、灵活性与可控性较好，对游戏服务器的效率没有什么影响。

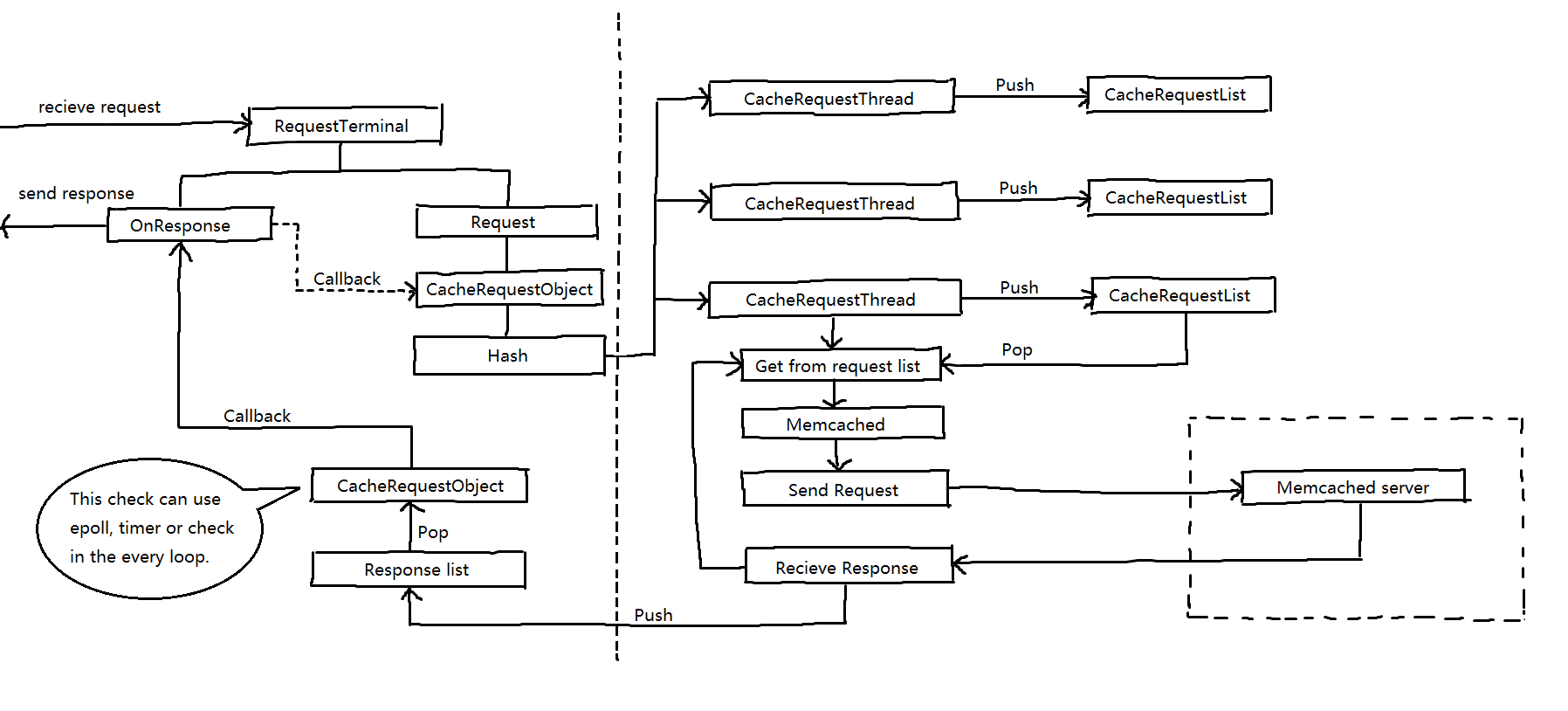
权衡各方面，我们选择第三种方式来实现。

**二、设计与实现**

权衡各方面，我们选择第三种方式来实现，使用以下两图来说明具体的设计：



此图是cache代理服务器的时序图，从一个请求发来到这个请求的回应过程。因为我们需要使用libmemcached这个客户端库，而这个库与memcached服务器的交互是阻塞的，所以我们必须使用多线程来提高并发量，所以我们设计了一组用于cache请求的客户端线程池，逻辑线程非阻塞，cache请求客户端线程都是阻塞的，没有请求不占用CPU。



此图是从请求到回应的流程图，是根据时序图的具体化，用以说明整个请求的流程与设计。