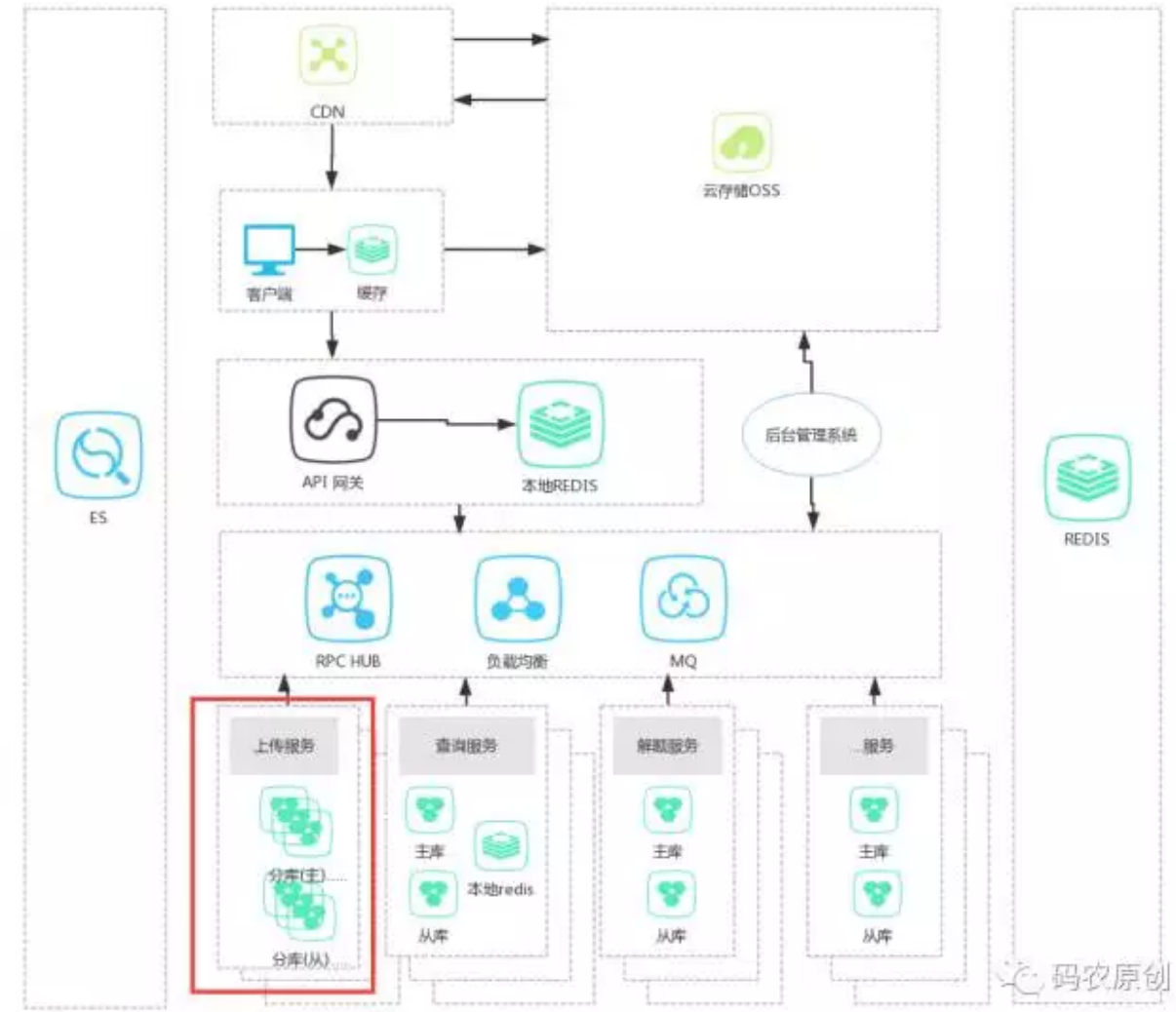
这种模式主要解决单表写入、读取、存储压力过大，从而导致业务缓慢甚至超时，交易失败，容量不够的问题。一般有水平切分和垂直切分两种，这里主要介绍水平切分。这个模式也是技术架构迭代演进过程中的必经之路。

这种模式的一般设计见下图：



如上图所示红色部分，把一张表分到了几个不同的库中，从而分担压力。是不是很笼统？哈哈，那我们接下来就详细的讲解一下。首先澄清几个概念，如下：

主机：硬件，指一台物理机，或者虚拟机，有自己的CPU，内存，硬盘等。

实例：数据库实例，如一个MySQL服务进程。一个主机可以有多个实例，不同的实例有不同的进程，监听不同的端口。

库：指表的集合，如学校库，可能包含教师表、学生表、食堂表等等，这些表在一个库中。一个实例中可以有多个库。库与库之间用库名来区分。

表：库中的表，不必多说，不懂的就不用往下看了，不解释。

那么怎么把单表分散呢？到底怎么个分发呢？分发到哪里呢？以下是几个工作中的实践，分享一下：

主机：这是最主要的也是最重要的点，本质上分库分表是因为计算与存储资源不够导致的，而这种资源主要是由物理机，主机提供的，所以在这里分是最基本的，毕竟没有可用的计算资源，怎么分效果都不是太好的。

实例：实例控制着连接数，同时受OS限制，CPU、内存、硬盘、网络IO也会受间接影响。会出现热实例的现象，即：有些实例特别忙，有些实例非常的空闲。一个典型的现象是：由于单表反应慢，导致连接池被打满，所有其他的业务都受影响了。这时候，把表分到不同的实例是有一些效果的。

库：一般是由于单库中最大单表数量的限制，才采取分库。

表：单表压力过大，索引量大，容量大，单表的锁。据以上，把单表水平切分成不同的表。

大型应用中，都是一台主机上只有一个实例，一个实例中只有一个库，**库==实例==主机，所以才有了分库分表**这个简称。

既然知道了基本理论，那么具体是怎么做的呢？逻辑是怎么跑的呢？接下来以一个例子来讲解一下。

这个需求很简单，用户表(user)，单表数据量1亿，查询、插入、存储都出现了问题，怎么办呢？

首先，分析问题，这个明显是由于数据量太大了而导致的问题。

其次，设计方案，可以分为10个库，这样每个库的数据量就降到了1KW，单表1KW数据量还是有些大，而且不利于以后量的增长，所以每个库再分100个表，这个每个单表数据量就为10W了，对于查询、索引更新、单表文件大小、打开速度，都有一些益处。接下来，给IT部门打电话，要10台物理机，扩展数据库.....

最后，逻辑实现，这里应该是最有学问的地方。首先是写入数据，需要知道写到哪个分库分表中，读也是一样的，所以，需要有个请求路由层，负责把请求分发、转换到不同的库表中，一般有路由规则的概念。

怎么样，简单吧？哈哈，too 那义务。说说这个模式的问题，主要是带来了事务上的问题，因为分库分表，事务完成不了，而**分布式事务**又太笨重，所以这里需要有一定的策略，保证在这种情况下事务能够完成。采取的策略如：最终一致性、复制、特殊设计等。再有就是业务代码的改造，一些关联查询要改造，一些单表orderBy的问题需要特

殊处理，也包括groupBy语句，如何解决这些副作用不是一句两句能说清楚的，以后有时间，我单独讲讲这些。

该总结一下这种模式的优缺点的了，如下：

优点：减少数据库单表的压力。
缺点：事务保证困难、业务逻辑需要做大量改造。