09 | 线上服务：如何在线上提供高并发的推荐服务？

你好，我是王喆。今天开始，我们进入线上服务篇的学习。

很多同学提起推荐系统，首先想到的是那些结构“华丽”，发展迅速的推荐模型。但事实上，在一个实际的工业级推荐系统中，训练和实现推荐模型的工作量往往连一半都没有。大量的工作都发生在搭建并维护推荐服务器、模型服务模块，以及特征和模型参数数据库等线上服务部分。

同时，由于线上服务模块是直接服务用户，产生推荐结果的模块，如果一旦发生延迟增加甚至服务宕机的情况，就会产生公司级别的事故。因此毫不夸张地说，线上服务实际上是推荐系统中最关键的一个模块。

线上服务如果写得不好，不仅杂乱无章，而且难以升级维护。因此，为了让你掌握搭建起一个支持深度学习的、稳定可扩展的推荐服务的方法，在这一模块中，我们会依次来讲线上服务器、特征存储、模型服务等模块的知识。

今天，我们先聚焦线上服务器，一起搭建直接产生推荐结果的服务接口。在这个过程中，我们按照先了解、后思考、再实践的顺序，依次解决这 3 个关键问题：

1. 一个工业级的推荐服务器内部究竟都做了哪些事情？
2. 像阿里、字节、腾讯这样级别的公司，它们的推荐系统是怎么承接住每秒百万甚至上千万的推荐请求的？
3. 我们自己该如何搭建一个工业级推荐服务器的雏形呢？

工业级推荐服务器的功能

首先，我们来解决第一个问题，一个工业级的推荐服务器内部究竟做了哪些事情？要回答这个问题，我们要先回到专栏的出发点，在推荐系统的技术架构图上找到推荐系统线上服务模块的位置。只有我们心中有全局，学习才能有重点。图 1 中红色的部分就是我们要详细来讲的线上服务模块。

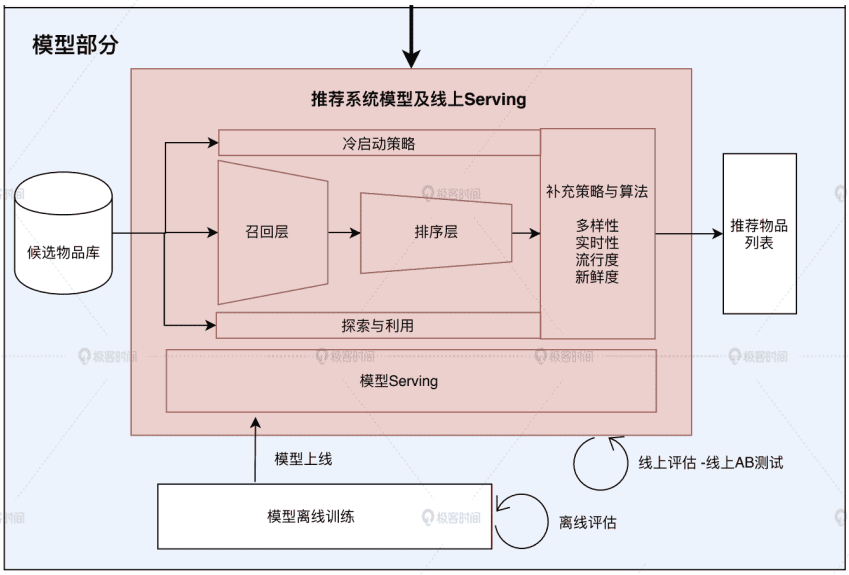

图1 推荐系统技术架构图

可以看到，线上服务模块的功能非常繁杂，它不仅需要跟离线训练好的模型打交道，把离线模型进行上线，在线进行模型服务（Model Serving），还需要跟数据库打交道，把候选物品和离线处理好的特征载入到服务器。

而且线上服务器内部的逻辑也十分地复杂，不仅包括了一些经典的过程，比如召回层和排序层，还包括一些业务逻辑，比如照顾推荐结果

多样性，流行度的一些硬性的混合规则，甚至还包括了一些 AB 测试相关的测试代码。

高并发推荐服务的整体架构

我刚才说的就是线上服务的技术框架了，可以说，想要把线上服务写好难度并不小，更何况在面对高 QPS 的压力下，事情还会变得更复杂。接下来，我们就来看第二个问题，说一说阿里、字节、腾讯这样级别的公司，使用了哪些策略来承接住每秒百万甚至是上千万推荐请求的。

说实话，想彻底讲清楚这个问题并不容易，因为大厂关于甚高并发具体的解决方案是集整个集团的技术精英打造的，而且维护一个高可用的服务集群的工作也不是一个算法工程师的主要工作方向。但这里，我还是希望你能够从宏观的角度了解高并发的主要解决方案，因为它是一个工业级推荐系统的重要组成部分，也是我们在与架构组配合工作时应有的知识储备。

宏观来讲，高并发推荐服务的整体架构主要由三个重要机制支撑，它们分别是**负载均衡、缓存、推荐服务降级机制。**下面，我们一一来看。

首先是负载均衡。它是整个推荐服务能够实现高可用、可扩展的基础。当推荐服务支持的业务量达到一定规模的时候，单独依靠一台服务器是不可行的，无论这台服务器的性能有多强大，都不可能独立支撑起高 QPS（Queries Per Second，每秒查询次数）的需求。这时候，我们就需要增加服务器来分担独立节点的压力。既然有多个劳动力在干活，那我们还需要一个“工头”来分配任务，以达到按能力分配和高效率分配的目的，这个“工头”就是所谓的“负载均衡服务器”。

下图就很好地展示了负载均衡的原理。我们可以看到，负载均衡服务器（Load Balancer）处在一个非常重要的位置。因此在实际工程中，负载均衡服务器也经常采用非常高效的 nginx 技术选型，甚至采用专门的硬件级负载均衡设备作为解决方案。

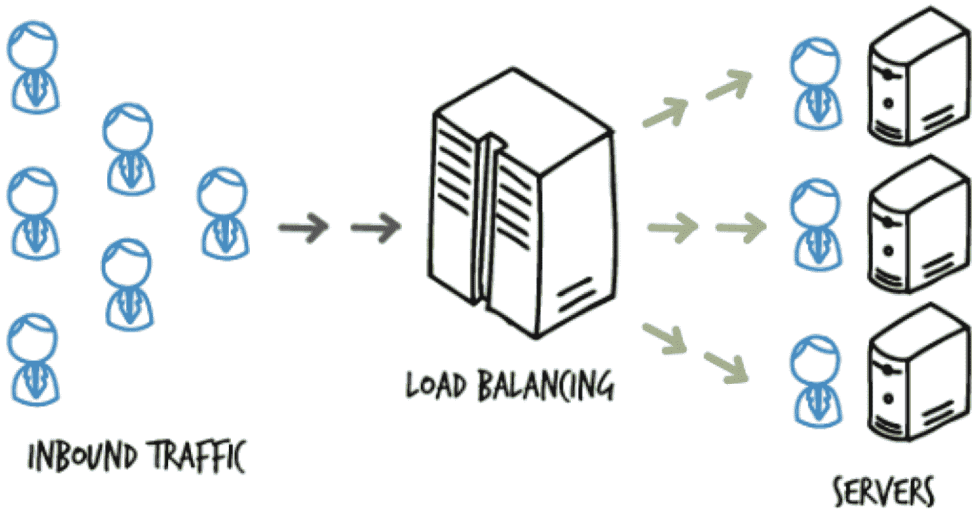

图2 高并发情况下的负载均衡服务器（来源：GitHub）

这个时候，有的同学可能会问，“负载均衡”解决高并发的思路是“增加劳动力”，那我们能否从“减少劳动量”的角度来解决高并发带来的负载压力呢？这是一个非常好的角度。要知道，推荐过程特别是基于深度学习的推荐过程往往是比较复杂的，进一步来说当候选物品规模比较大的时候，产生推荐列表的过程其实非常消耗计算资源，服务器的“劳动量”非常大。这个时候，我们就可以通过减少“硬算”推荐结果的次数来给推荐服务器减负，那具体怎么做呢？

比如说，当同一个用户多次请求同样的推荐服务时，我们就可以在第一次请求时把 TA 的推荐结果缓存起来，在后续请求时直接返回缓存

中的结果就可以了，不用再通过复杂的推荐逻辑重新算一遍。再比如说，对于新用户来说，因为他们几乎没有行为历史的记录，所以我们可以先按照一些规则预先缓存好几类新用户的推荐列表，等遇到新用户的时候就直接返回。

因此，在一个成熟的工业级推荐系统中，合理的缓存策略甚至能够阻挡掉 90% 以上的推荐请求，大大减小推荐服务器的计算压力。

但不管是再强大的服务集群，还是再有效的缓存方案，也都有可能遭遇特殊时刻的流量洪峰或者软硬件故障。在这种特殊情况下，为了防止推荐服务彻底熔断崩溃，甚至造成相关微服务依次崩溃的“雪崩效应”，我们就要在第一时间将问题控制在推荐服务内部，而应对的最好机制就是“服务降级”。

所谓“服务降级”就是抛弃原本的复杂逻辑，采用最保险、最简单、最不消耗资源的降级服务来度过特殊时期。比如对于推荐服务来说，我们可以抛弃原本的复杂推荐模型，采用基于规则的推荐方法来生成推荐列表，甚至直接在缓存或者内存中提前准备好应对故障时的默认推荐列表，做到“0”计算产出服务结果，这些都是服务降级的可行策略。

总之，“负载均衡”提升服务能力，“缓存”降低服务压力，“服务降级”机制保证故障时刻的服务不崩溃，压力不传导，这三点可以看成是一个成熟稳定的高并发推荐服务的基石。

搭建一个工业级推荐服务器的雏形

那说了这么多，这对我们搭建一个工业级推荐服务器有什么实际帮助呢？

相信你肯定听说过一句话，算法工程师是“面试造火箭，工作拧螺丝”。说实话，这确实反映了算法岗面试的一些不合理之处，但也不是说造火箭的知识不应该掌握。要给一个火箭拧螺丝，真不是说会拧螺丝就可以了，还真是得清楚火箭的构造是什么样的，否则螺丝你是拧上了，但地方拧错了，照样会让火箭出事故。

我们刚才讲的大厂处理高并发服务的方法就是“造火箭”，理解了这些方法，我们再来看实际工作中“拧螺丝”的技巧，就能做到有的放矢。下面，我们就一起在 Sparrow Recsys 里面实践一下搭建推荐服务器的过程，看看如何一步步拧螺丝，搭建起一个可用的推荐服务器。当然，它肯定无法直接具备负载均衡这些企业级服务的能力，但我可以保证，它可以作为一个工业级推荐服务器的雏形。让你以此为起点，逐渐把它扩展成为一个成熟的推荐服务。

首先，我们要做的就是选择服务器框架。这里，我们选择的服务器框架是 Java 嵌入式服务器 Jetty。为什么我们不选择其他的服务器呢？原因有三个。

第一，相比于 Python 服务器的效率问题，以及 C++ 服务器的开发维护难度，Java 服务器在效率和开发难度上做到了一个权衡。而且互联网上有大量开源 Java 项目可以供我们直接融合调用，所以 Java 服务器开发的扩展性比较好。第二，相比 Tomcat 等其他 Java 服务器，Jetty 是嵌入式的，它更轻量级，没有过多 J2EE 的冗余功能，可以专注于建立高效的 api 推荐服务。而 Tomcat 更适用于搭建一整套的 J2EE 项目。第三，相比于基于 Nodejs、Go 这样的服务器，Java 社区更成熟和主流一些，应用范围更广。

当然，每一种技术选择都有它的优势，C++ 的效率更高，Python 更便捷，Go 的上升势头也愈发明显，我们只要清楚 Jetty 是企业级服务

的选择之一就够了，我们接下来的服务器端实践也是基于 Jetty 开展的。

作为一款嵌入式服务器框架，Jetty 的最大优势是除了 Java 环境外，你不用配置任何其他环境，也不用安装额外的软件依赖，你可以直接在 Java 程序中创建对外服务的 http api，之后在 IDE 中运行或者打 Jar 包运行就可以了。下面就是我们 Sparrow Recsys 中创建推荐服务器的代码，我已经在所有关键的地方添加了注释，你可以逐句解读一下。

```
public class RecSysServer {
    //主函数，创建推荐服务器并运行
    public static void main(String[] args) throws
        new RecSysServer().run();
    }
    //推荐服务器的默认服务端口6010
    private static final int DEFAULT_PORT = 6010;


    //运行推荐服务器的函数
    public void run() throws Exception{

        int port = DEFAULT_PORT;
        //绑定IP地址和端口，0.0.0.0代表本地运行
        InetSocketAddress inetAddress = new InetSo
        //创建Jetty服务器
        Server server = new Server(inetAddress);
        //创建Jetty服务器的环境handler
        ServletContextHandler context = new Servle
        context.setContextPath("/");
        context.setWelcomeFiles(new String[] { "in


        //添加api，getmovie，获取电影相关数据
```

```
        context.addServlet(new ServletHolder(new M
        //添加api，getuser，获取用户相关数据
        context.addServlet(new ServletHolder(new U
        //添加api，getsimilarmovie，获取相似电影推荐
        context.addServlet(new ServletHolder(new S
        //添加api，getrecommendation，获取各类电影推荐
        context.addServlet(new ServletHolder(new R
        //设置Jetty的环境handler
        server.setHandler(context);


        //启动Jetty服务器
        server.start();
        server.join();
    }
```

你可以看到，创建 Jetty 服务的过程非常简单直观，十几行代码就可以搭建起一套推荐服务。当然，推荐服务的主要业务逻辑并不在这里，而是在每个注册到 Jetty Context 中的 Servlet 服务中。这里我们用其中最简单的 Servlet 服务 MovieService，来看一看 Jetty 中的 Servlet 服务是怎么写的。

```
//MovieService需要继承Jetty的HttpServlet
public class MovieService extends HttpServlet {
    //实现servlet中的get method
    protected void doGet(HttpServletRequest reques
                         HttpServletResponse respo
        try {
            //该接口返回json对象，所以设置json类型
            response.setContentType("application/j
            response.setStatus(HttpServletResponse
            response.setCharacterEncoding("UTF-8")
            response.setHeader("Access-Control-All
```

```
            //获得请求中的id参数，转换为movie id
            String movieId = request.getParameter(
            //从数据库中获取该movie的数据对象
            Movie movie = DataManager.getInstance(


            if (null != movie) {
                //使用fasterxml.jackson库把movie对象
                ObjectMapper mapper = new ObjectMa
                String jsonMovie = mapper.writeVal
                //返回json对象

                response.getWriter().println(jsonM
            }else {
                response.getWriter().println("");
            }


        } catch (Exception e) {
            e.printStackTrace();
            response.getWriter().println("");
        }
    }
```

熟悉了这个 Servlet 服务，其他服务就依葫芦画瓢就可以啦。唯一的不同就是其中的业务逻辑。如果你已经从 github 上下载了 sparrow recsys 项目把它运行起来，并且在浏览器中输入 http://localhost:6010/getmovie?id=1，就可以看到 getmovie 接口的返回对象了。

小结

这节课我们既学习了怎么“造火箭”，又实践了怎么“拧螺丝”。对于一个合格的算法工程师来说，这两方面缺一不可。

“造火箭”的知识包括工业级推荐服务器的具体功能，以及实现工业级高并发推荐服务的主要机制。其中，推荐服务器的具体功能主要有：模型服务、数据库接口、推荐模块逻辑、补充业务逻辑等等，而工业级高并发推荐服务的主要机制有负载均衡、缓存和服务降级。

“拧螺丝”的技能我们也掌握了不少，我们利用 Jetty 实践并搭建起了我们 SparrowRecSys 的推荐服务接口。这个过程中，我们需要重点关注的是，每个注册到 Jetty Context 的 Servlet 服务中的主要业务逻辑，只要掌握了一个，在实际工作中我们就能举一反三了。老规矩，我今天继续用表格的形式帮你整理了这节课的主要知识点，你可以看看。

知识点	关键描述
工业级推荐服务器的功能	主要包括模型服务、数据库接口、 推荐模块逻辑、补充业务逻辑等等
工业级高并发推荐服务的主要机制	负载均衡、缓存、降级机制
Jetty服务器的优势	轻量级嵌入式，Java开源社区完善， 服务器开发效率和运行效率兼顾
Jetty实战	创建Jetty服务器和实现Jetty Servlet服务

好了，推荐服务器的相关内容我就先讲到这里，下节课我会继续讲解线上服务的另一个主要的组成部分，存储模块。

课后思考

在一个高并发的推荐服务集群中，负载均衡服务器的作用至关重要，如果你是负载均衡服务器的策略设计师的话，你会怎么实现这个“工头”的调度策略，让它能够公平又高效的完成调度任务呢？（比如是按每个节点的能力分配？还是按照请求本身的什么特点来分配？如何知道什么时候应该扩展节点，什么时候应该关闭节点？）

欢迎把你的思考和答案写在留言区，也欢迎你把这节课分享给你的朋友，我们下节课见！