缓存原理&设计

本章学习目标：

- 理解缓存的使用场景
- 理解缓存原理
- 了解常见缓存及分类
- 理解服务器端缓存
- 了解缓存的优势和代价
- 理解缓存的读写模式
- 掌握缓存的常见设计思路并能够运用到项目中

生产中遇到的缓存问题（大厂常见面试题）

- 系统在某个时刻访问量剧增（热点新闻），造成数据库压力剧增甚至崩溃，怎么办？
- 什么是缓存雪崩、缓存穿透和缓存击穿，会造成什么问题，如何解决？
- 什么是大Key和热Key，会造成什么问题，如何解决？
- 如何保证 Redis 中的数据都是热点数据？
- 缓存和数据库数据是不一致时，会造成什么问题，如何解决？
- 什么是数据并发竞争，会造成什么问题，如何解决？
- 单线程的Redis为什么这么快？
- Redis哨兵和集群的原理及选择？
- 在多机Redis使用时，如何保证主从服务器的数据一致性？

缓存基本思想

缓存的使用场景

DB缓存，减轻DB服务器压力

一般情况下数据存在数据库中，应用程序直接操作数据库。

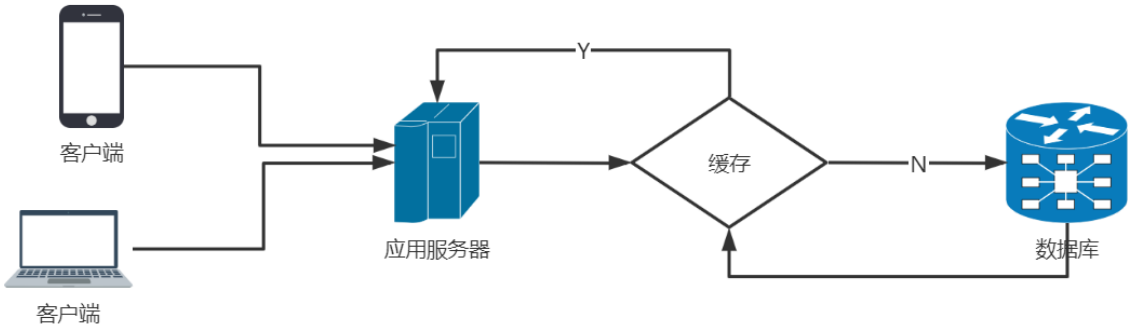
当访问量上万，数据库压力增大，可以采取的方案有：

读写分离，分库分表

当访问量达到10万、百万，需要引入缓存。

将已经访问过的内容或数据存储起来，当再次访问时先找缓存，缓存命中返回数据。

不命中再找数据库，并回填缓存。



提高系统响应

数据库的数据是存在文件里，也就是硬盘。与内存做交换（swap）

在大量瞬间访问时（高并发）MySQL单机会因为频繁IO而造成无法响应。MySQL的InnoDB是有行锁

将数据缓存在Redis中，也就是存在了内存中。

内存天然支持高并发访问。可以瞬间处理大量请求。

qps到达11万/S读请求 8万写/S

做Session分离

传统的session是由tomcat自己进行维护和管理。

集群或分布式环境，不同的tomcat管理各自的session。

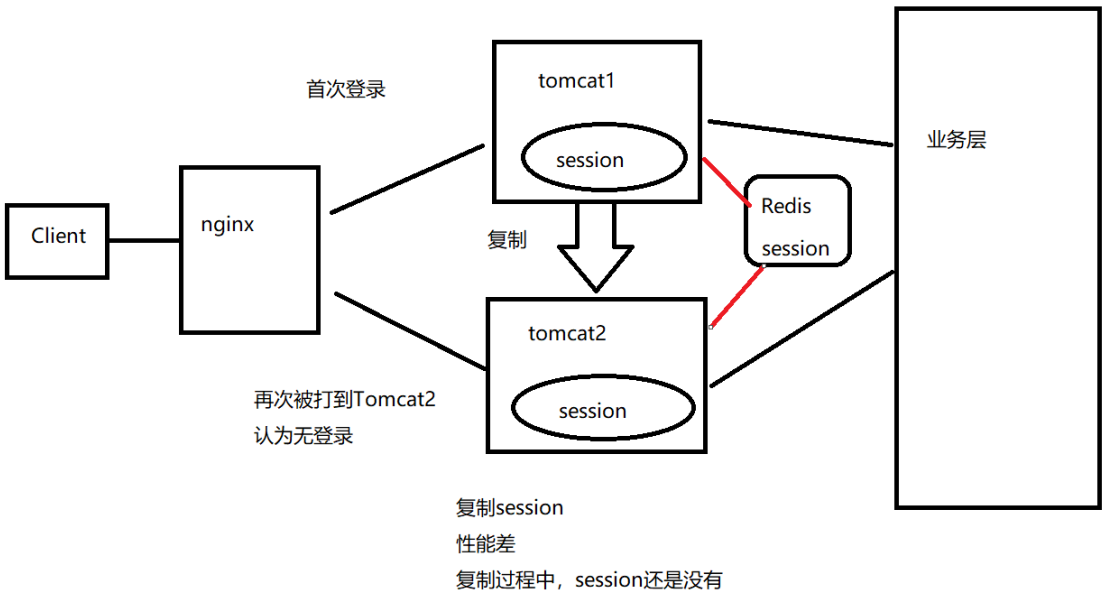
只能在各个tomcat之间，通过网络和Io进行session的复制，极大的影响了系统的性能。

1、各个Tomcat间复制session，性能损耗

2、不能保证各个Tomcat的Session数据同步

将登录成功后的Session信息，存放在Redis中，这样多个服务器(Tomcat)可以共享Session信息。

Redis的作用是数据的临时存储



做分布式锁（Redis）

一般讲锁是多线程的锁，是在一个进程中的

多个进程（JVM）在并发时也会产生问题，也要控制时序性

可以采用分布式锁。使用Redis实现 setNX

做乐观锁（Redis）

同步锁和数据库中的行锁、表锁都是悲观锁

悲观锁的性能是比较低的，响应性比较差

高性能、高响应（秒杀）采用乐观锁

Redis可以实现乐观锁 watch + incr

什么是缓存?

缓存原指CPU上的一种高速存储器，它先于内存与CPU交换数据，速度很快

现在泛指存储在计算机上的原始数据的复制集，便于快速访问。

在互联网技术中，缓存是系统快速响应的关键技术之一

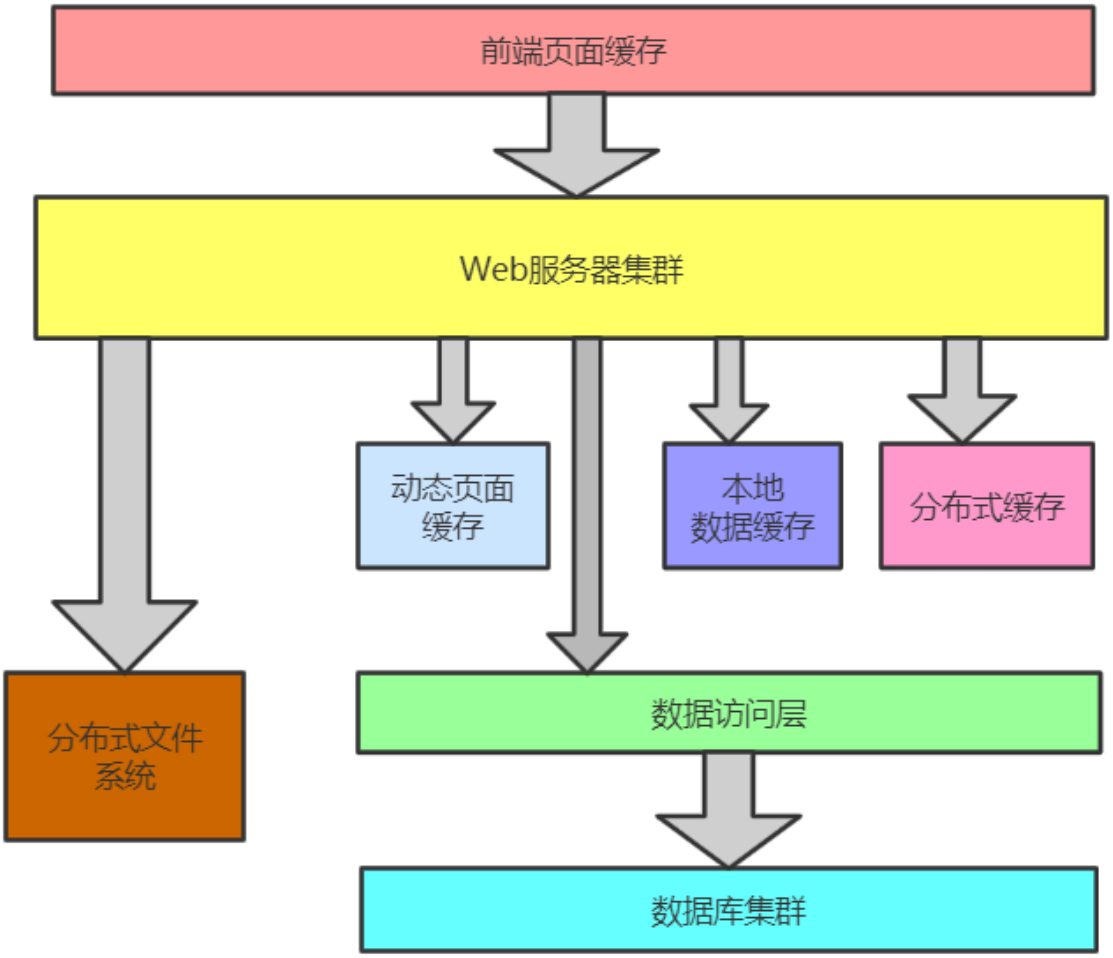
以空间换时间的一种技术（艺术）

大型网站中缓存的使用

单机架构LAMP（Linux+apache+MySQL+PHP）、JavaEE（SSM）

访问量越大，响应力越差，用户体验越差

引入缓存、示意图如下：



在大型网站中从浏览器到网络，再到应用服务器，再到数据库，通过在各个层面应用缓存技术，大大提升了系统性能和用户体验。

常见缓存的分类

客户端缓存

传统互联网：页面缓存和浏览器缓存

移动互联网：APP缓存

页面缓存

页面缓存：页面自身对某些元素或全部元素进行存储，并保存成文件。

html5：Cookie、WebStorage（SessionStorage和LocalStorage）、WebSql、indexDB、Application Cache等

开启步骤：

1、设置manifest描述文件

```
CACHE MANIFEST
#comment

js/index.js
img/bg.png
```

2、html关联manifest属性

```
<html lang="en" manifest="demo.appcache">
```

使用LocalStorage进行本地的数据存储，示例代码：

```
localStorage.setItem("Name","张飞")
localStorage.getItem("Name")
localStorage.removeItem("Name")
localStorage.clear()
```

浏览器缓存

当客户端向服务器请求资源时，会先抵达浏览器缓存，如果浏览器有“要请求资源”的副本，就可以直接从浏览器缓存中提取而不是从原始服务器中提取这个资源。

浏览器缓存可分为强制缓存和协商缓存。

强制缓存：直接使用浏览器的缓存数据

条件：Cache-Control的max-age没有过期或者Expires的缓存时间没有过期

```
<meta http-equiv="Cache-Control" content="max-age=7200" />
<meta http-equiv="Expires" content="Mon, 20 Aug 2010 23:00:00 GMT" />
```

协商缓存：服务器资源未修改，使用浏览器的缓存（304）；反之，使用服务器资源（200）。

```
<meta http-equiv="cache-control" content="no-cache">
```

APP缓存

原生APP中把数据缓存在内存、文件或本地数据库（SQLite）中。比如图片文件。

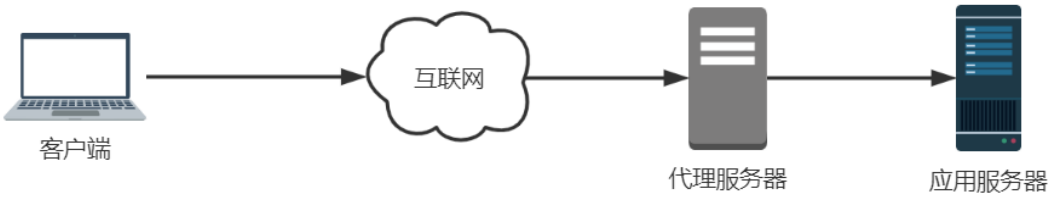
网络端缓存

通过代理的方式响应客户端请求，对重复的请求返回缓存中的数据资源。

Web代理缓存

可以缓存原生服务器的静态资源，比如样式、图片等。

常见的反向代理服务器比如大名鼎鼎的Nginx。



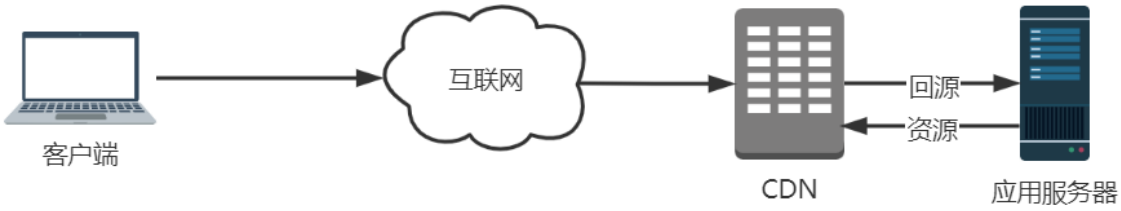
边缘缓存

边缘缓存中典型的商业化服务就是CDN了。

CDN的全称是Content Delivery Network，即内容分发网络。

CDN通过部署在各地的边缘服务器，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。

CDN的关键技术主要有内容存储和分发技术。现在一般的公有云服务商都提供CDN服务。



服务端缓存

服务器端缓存是整个缓存体系的核心。包括数据库级缓存、平台级缓存和应用级缓存。

数据库级缓存

数据库是用来存储和管理数据的。

MySQL在Server层使用查询缓存机制。将查询后的数据缓存起来。

K-V结构，Key：select语句的hash值，Value：查询结果

InnoDB存储引擎中的buffer-pool用于缓存InnoDB索引及数据块。

平台级缓存

平台级缓存指的是带有缓存特性的应用框架。

比如：GuavaCache 、EhCache（二级缓存，硬盘）、OSCache（页面缓存）等。

部署在应用服务器上，也称为服务器本地缓存。

应用级缓存（重点）

具有缓存功能的中间件：Redis、Memcached、EVCache（AWS）、Tair（阿里 、美团）等。

采用K-V形式存储。

利用集群支持高可用、高性能、高并发、高扩展。

分布式缓存

缓存的优势、代价

使用缓存的优势

提升用户体验

用户体验（User Experience）：用户在使用产品过程中建立起来的一种纯主观感受。

缓存的使用可以提升系统的响应能力，大大提升了用户体验。

减轻服务器压力

客户端缓存、网络端缓存减轻应用服务器压力。

服务端缓存减轻数据库服务器的压力。

提升系统性能

系统性能指标：响应时间、延迟时间、吞吐量、并发用户数和资源利用率等。

缓存技术可以：

缩短系统的响应时间

减少网络传输时间和应用延迟时间

提高系统的吞吐量

增加系统的并发用户数

提高了数据库资源的利用率

使用缓存的代价

额外的硬件支出

缓存是一种软件系统中以空间换时间的技术

需要额外的磁盘空间和内存空间来存储数据

搭建缓存服务器集群需要额外的服务器

采用云服务器的缓存服务就不用额外的服务器了

阿里云（Tair、Redis），百度云（Redis），提供缓存服务

AWS亚马逊云服务：EVCache

高并发缓存失效

在高并发场景下会出现缓存失效（缓存穿透、缓存雪崩、缓存击穿）

造成瞬间数据库访问量增大，甚至崩溃

缓存与数据库数据同步

缓存与数据库无法做到数据的时时同步

Redis无法做到主从时时数据同步

缓存并发竞争

多个redis的客户端同时对一个key进行set值得时候由于执行顺序引起的并发问题

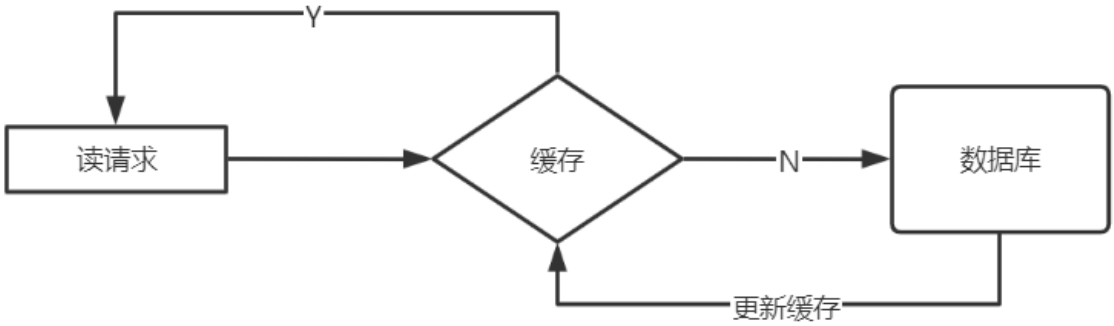
缓存的读写模式

缓存有三种读写模式

Cache Aside Pattern（常用）

Cache Aside Pattern（旁路缓存），是最经典的缓存+数据库读写模式。

读的时候，先读缓存，缓存没有的话，就读数据库，然后取出数据后放入缓存，同时返回响应。



更新的时候，先更新数据库，然后再删除缓存。

为什么是删除缓存，而不是更新缓存呢?

1、缓存的值是一个结构：hash、list，更新数据需要遍历

先遍历（耗时）后修改

2、懒加载，使用的时候才更新缓存

使用的时候才从DB中加载

也可以采用异步的方式填充缓存

开启一个线程 定时将DB的数据刷到缓存中

高并发**脏读**的三种情况

1、先更新数据库，再更新缓存

update与commit之间，更新缓存，commit失败

则DB与缓存数据不一致

2、先删除缓存，再更新数据库

update与commit之间，有新的读，缓存空，读DB数据到缓存 数据是旧的数据

commit后 DB为新数据

则DB与缓存数据不一致

3、先更新数据库，再删除缓存（推荐）

update与commit之间，有新的读，缓存空，读DB数据到缓存 数据是旧的数据

commit后 DB为新数据

则DB与缓存数据不一致

采用延时双删策略

Read/Write Through Pattern

应用程序只操作缓存，缓存操作数据库。

Read-Through（穿透读模式/直读模式）：应用程序读缓存，缓存没有，由缓存回源到数据库，并写入缓存。（guavacache）

Write-Through（穿透写模式/直写模式）：应用程序写缓存，缓存写数据库。

该种模式需要提供数据库的handler，开发较为复杂。

Write Behind Caching Pattern

应用程序只更新缓存。

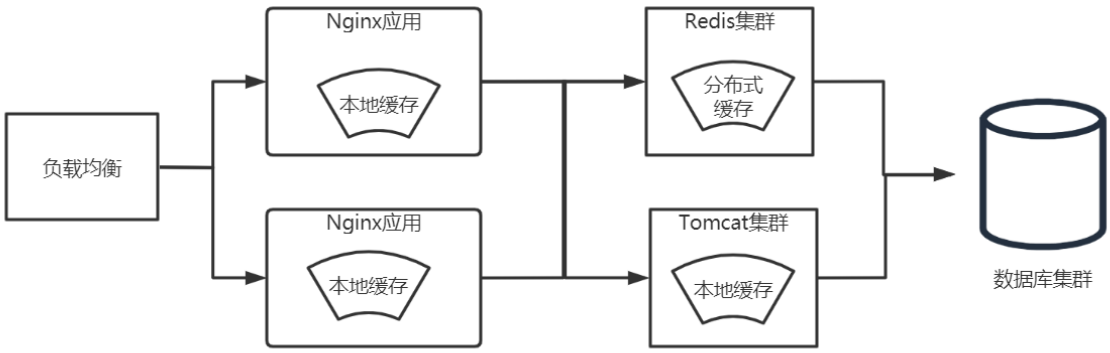
缓存通过异步的方式将数据批量或合并后更新到DB中

不能时时同步，甚至会丢数据

缓存架构的设计思路

缓存的整体设计思路包括：

1、多层次



(多级缓存的系统架构)

分布式缓存宕机，本地缓存还可以使用

2、数据类型

简单数据类型

Value是字符串或整数或二进制

Value的值比较大（大于100K）

只进行setter和getter

可采用Memcached

Memcached纯内存缓存，多线程 K-V

复杂数据类型

Value是hash、set、list、zset

需要存储关系，聚合，计算

可采用Redis

3、要做集群

分布式缓存集群方案（Redis）

codis

哨兵+主从

RedisCluster

4、缓存的数据结构设计

1、与数据库表一致

数据库表和缓存是一一对应的

缓存的字段会比数据库表少一些

缓存的数据是经常访问的

用户表，商品表

2、与数据库表不一致

需要存储关系，聚合，计算等

比如某个用户的帖子、用户的评论。

以用户评论为例，DB结构如下：

ID	UID	PostTime	Content
1	1000	1547342000	xxxxxxxxxx
2	1000	1547342000	xxxxxxxxxx
3	1001	1547341030	xxxxxxxxxx

如果要取出UID为1000的用户的评论，原始的表的数据结构显然是不行的。

我们应做如下设计：

key：UID+时间戳(精确到天) 评论一般以天为计算单位

value：Redis的Hash类型。field为 id和content

expire：设置为一天

案例：设计拉勾首页缓存职位列表、热门职位

拉勾网(www.lagou.com)，是国内的招聘门户网站，亿万级PV，单机响应性能QPS万级。

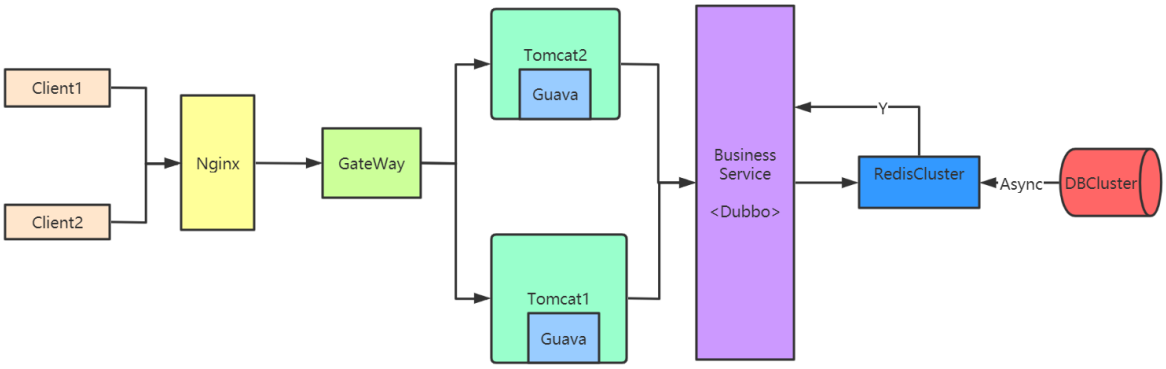
首页分析：

职位时时变化，不能使用静态html

采用模板技术，数据在服务端拿出，不能为空

数据不一定时时

架构图如下：



1、静态文件

在nginx中，放置静态文件，比如css，js， 图片等

```
server {
        listen       80 default_server;
        server_name  localhost;
        root /mnt/blog/;

        location / {

        }

        #要缓存文件的后缀，可以在以下设置。
        location ~ .*\.(gif|jpg|png|css|js)(.*) {
                proxy_pass http://ip地址:90;
                proxy_redirect off;
                proxy_set_header Host $host;
                proxy_cache cache_one;
                proxy_cache_valid 200 302 24h;
                proxy_cache_valid 301 30d;
                proxy_cache_valid any 5m;
                expires 90d;
                add_header wall  "hello lagou.";
        }
    }
```

2、职位列表

数据特点：

固定数据，一次性读取

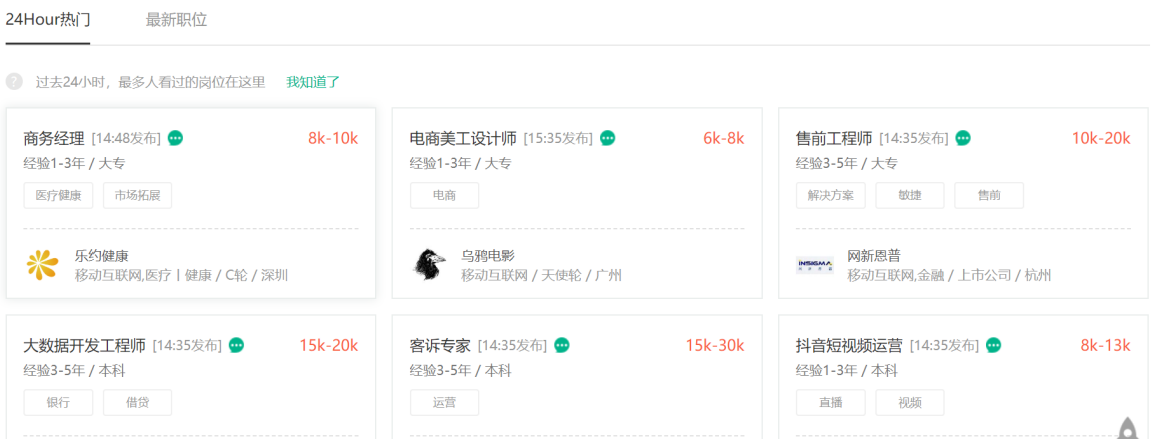
方案：

在服务器开启时一次性初始化（从xml）到服务器本地缓存

采用Guava Cache，Guava Cache用于存储频繁使用的少量数据，支持高并发访问

也可以使用JDK的CurrentHashMap，需要自行实现

3、热门职位



数据特点：

频繁变化，不必时时同步

但一定要有数据，不能为空

方案：

数据从服务层读取（dubbo），然后放到本地缓存中（Guava），如果出现超时或读取为空，则返回原来本地缓存的数据。

注意：不同的客户端看到的数据有可能不一样。

4、数据回填

从Dubbo中读取数据时，先读取Redis集群的缓存，如果缓存命中则直接返回。

如果缓存不命中则返回本地缓存，不能直接读取数据库。

采用异步的形式从数据库刷入到缓存中。

5、热点策略

对于热点数据我们采用本地缓存策略，而不采用服务熔断策略，因为首页数据可以不准确，但不能不响应。