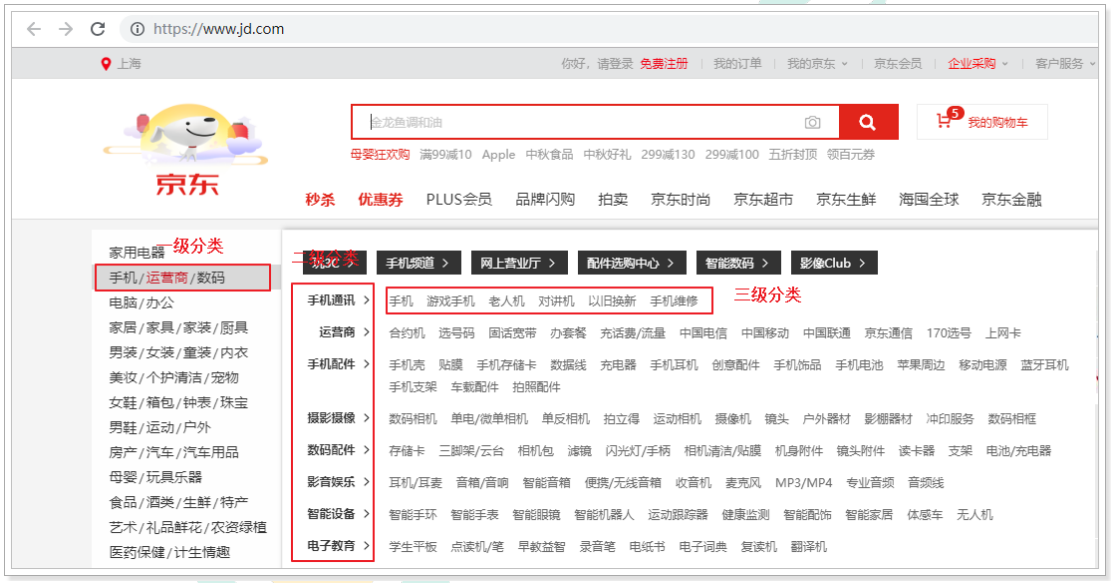
尚品汇商城复习

版本：V 1.0

商城首页

首页商品分类实现

前面做了商品详情，我们现在来做首页分类，我先看看京东的首页分类效果，我们如何实现类似效果：



思路：

1，首页属于并发量比较高的访问页面，我看可以采取页面静态化方式实现，或者把数据放在缓存中实现

2，我们把生产的静态文件可以放在 nginx 访问或者放在 web-index 模块访问

1、封装数据接口

由于商品分类信息在 service-product 模块，我们在该模块封装数据，数据结构为父子层级，商品分类保存在 base_category1、base_category2 和 base_category3 表中，

由于需要静态化页面，我们需要一次性加载所有数据，前面我们使用了一个视图base_category_view，所有我从视图里面获取数据，然后封装为父子层级

数据结构如下：

```
[
  {
    "index": 1,
    "categoryChild": [
      {
        "categoryChild": [
          {
            "categoryName": "电子书",
            "categoryId": 1
          },
          {
            "categoryName": "网络原创",
            "categoryId": 2
          },
          ...
        ],
        "categoryName": "电子书刊",
        "categoryId": 1
      },
     ...
    ],
    "categoryName": "图书、音像、电子书刊",
    "categoryId": 1
  },
  ...
]
```

2、什么是数据库视图

数据库中的视图是一个虚拟表，但它同真实表一样，包含一系列带有名称的行和列数据。行和列数据来自由定义视图查询所引用的表，并且在应用视图时动态生成。另外，视图还可以在已经存在的视图的基础上定义。

视图一经定义变存储在数据库中，与其相对应的数据并没有像表那样在数据库中再存储一份，通过视图看到的数据只是存储在基本表中的数据。对视图的操作与对表的操作一样，可以对其进行查询、修改和删除。当对通过视图看到的数据进行修改时，相应的基本表中的数据也会发生变化；同时，若是基本表的数据发生变化，则这种变化也会自动地反映在视图上。

视图的作用

与直接从真实数据表中进行数据操作相比，视图具有以下的有点：

（1）简单化

视图不仅可以简化用户对数据的理解，也可以简化他们的操作。那些被经常使用的查询可以被定义为视图，从而用户不必为以后的每一次操作指定全部的条件。

（2）安全性

通过视图用户只能查询和修改他们所能看到的数据。数据库中的其他数据则既看不见也取不到。数据库授权命令可以使每个用户对数据库的检索限制到特定的数据库对象上，但不能限制到特定行和特定列上。但通过视图，用户可以被限制到数据库的行列级别的子集上。

（3）逻辑数据独立性

视图可以帮助用户屏蔽真实表结构变化带来的影响。

3、页面静态化：

什么是页面静态化

- 将动态页面转化成静态的 html，降低与数据库的交互次数，提高页面的访问速度
- 就是服务器在请求来之前把已经固定好的东西先编译好了，等请求来了再动态的填数据，不要等请求来了什么都没做忙得半死
- 利用第三方提供的模板引擎,生成对应的 html
- 常用的页面静态化技术有 thymeleaf、freemarker

为什么要使用静态化

- 网页静态化技术和缓存技术的共同点都是为了减轻数据库的访问压力
- 而网页静态化比较适合大规模且相对变化不太频繁的数据。
- 将网页以纯静态化的形式展现，就可以使用 Nginx 这样的高性能的 web 服务器来部署
- Nginx 可以承载 5 万的并发，而 Tomcat 只有几百

实现代码：

```
package com.atguigu.gmall.all.controller;

@Controller
@RequestMapping
public class IndexController {

    @Autowired
    private ProductFeignClient productFeignClient;

    @Autowired
    private SpringTemplateEngine templateEngine;

        /**
     * 生成静态页面
   * @return
     * @throws IOException
     */
    @GetMapping("/createHtml")

@ResponseBody
    public Result createHtml() throws IOException {
        Result result =
productFeignClient.getBaseCategoryList();
        Context context = new Context();
        context.setVariable("list", result.getData());
        FileWriter write = new FileWriter("D:\\index.html");
        templateEngine.process("index/index.html", context,
write);
        return Result.ok();
    }
```