Java 8都出那么久了，Stream API了解下?

原创 梦想de星空 macrozheng 2019-09-02 08:33

收录于合集
#mall学习教程（技术要点篇） 17个

> Java 8 引入了全新的 Stream API，可以使用声明的方式来处理数据，极大地方便了集合操作，让我们可以使用更少的代码来实现更为复杂的逻辑，本文主要对一些常用的Stream API进行介绍。

什么是Stream?

Stream（流）是一个来自数据源的元素队列，它可以支持聚合操作。

- 数据源：流的数据来源，构造Stream对象的数据源，比如通过一个List来构造Stream对象，这个List就是数据源；
- 聚合操作：对Stream对象进行处理后使得Stream对象返回指定规则数据的操作称之为聚合操作，比如filter、map、limit、sorted等都是聚合操作。

Stream 聚合操作

背景介绍

本文将以mall中的UmsPermission对象为例来介绍Stream API的常用操作。UmsPermission是一个权限对象，主要分为三种权限，目录、菜单以及按钮，对象定义如下。

```
publicclass UmsPermission implements Serializable {
    private Long id;

    @ApiModelProperty(value = "父级权限id")
    private Long pid;

    @ApiModelProperty(value = "名称")
    private String name;
```

```
    @ApiModelProperty(value = "权限值")
    private String value;

    @ApiModelProperty(value = "图标")
    private String icon;

    @ApiModelProperty(value = "权限类型：0->目录；1->菜单；2->按钮（接口绑定权限）")
    private Integer type;

    @ApiModelProperty(value = "前端资源路径")
    private String uri;

    @ApiModelProperty(value = "启用状态；0->禁用；1->启用")
    private Integer status;

    @ApiModelProperty(value = "创建时间")
    private Date createTime;

    @ApiModelProperty(value = "排序")
    private Integer sort;

    privatestaticfinallong serialVersionUID = 1L;

    //省略所有getter及setter方法
}
```

Stream对象的创建

Stream对象分为两种，一种串行的流对象，一种并行的流对象。

```
// permissionList指所有权限列表
// 为集合创建串行流对象
Stream<UmsPermission> stream = permissionList.stream();
// 为集合创建并行流对象
tream<UmsPermission> parallelStream = permissionList.parallelStream();
```

filter

对Stream中的元素进行过滤操作，当设置条件返回true时返回相应元素。

```
// 获取权限类型为目录的权限
List<UmsPermission> dirList = permissionList.stream()
    .filter(permission -> permission.getType() == 0)
    .collect(Collectors.toList());
```

map

对Stream中的元素进行转换处理后获取。比如可以将UmsPermission对象转换成Long对象。我们经常会有这样的需求：需要把某些对象的id提取出来，然后根据这些id去查询其他对象，这时可以使用此方法。

```
// 获取所有权限的id组成的集合
List<Long> idList = permissionList.stream()
    .map(permission -> permission.getId())
    .collect(Collectors.toList());
```

limit

从Stream中获取指定数量的元素。

```
// 获取前5个权限对象组成的集合
List<UmsPermission> firstFiveList = permissionList.stream()
    .limit(5)
    .collect(Collectors.toList());
```

count

仅获取Stream中元素的个数。

```
// count操作：获取所有目录权限的个数
long dirPermissionCount = permissionList.stream()
    .filter(permission -> permission.getType() == 0)
    .count();
```

sorted

对Stream中元素按指定规则进行排序。

```
// 将所有权限按先目录后菜单再按钮的顺序排序
List<UmsPermission> sortedList = permissionList.stream()
    .sorted((permission1,permission2)->{return permission1.getType().compareTo(permission2.getTyp
    .collect(Collectors.toList());
```

skip

跳过指定个数的Stream中元素，获取后面的元素。

```
// 跳过前5个元素，返回后面的
List<UmsPermission> skipList = permissionList.stream()
    .skip(5)
    .collect(Collectors.toList());
```

用collect方法将List转成map

有时候我们需要反复对List中的对象根据id进行查询，我们可以先把该List转换为以id为key的map结构，然后再通过map.get(id)来获取对象，这样比较方便。

```
// 将权限列表以id为key，以权限对象为值转换成map
Map<Long, UmsPermission> permissionMap = permissionList.stream()
    .collect(Collectors.toMap(permission -> permission.getId(), permission -> permission));
```

应用

> 我们经常会有返回树形结构数据的需求。比如这里的权限，第一层是目录权限，目录权限之下有菜单权限，菜单权限之下有按钮权限。如果我们要返回一个集合，包含目录权限，目录权限下面嵌套菜单权限，菜单权限下嵌套按钮权限。使用Stream API可以很方便的解决这个问题。

注意：这里我们的权限上下级之间以**pid**来关联，**pid**是指上一级权限的**id**，顶级权限的**id**为**0**。

定义包含下级权限的对象

继承自UmsPermission对象，之增加了一个children属性，用于存储下级权限。

```
/**
 * Created by macro on 2018/9/30.
 */
publicclass UmsPermissionNode extends UmsPermission {
    private List<UmsPermissionNode> children;

    public List<UmsPermissionNode> getChildren() {
        return children;
    }

    public void setChildren(List<UmsPermissionNode> children) {
        this.children = children;
    }
}
```

定义获取树形结构的方法

我们先过滤出pid为0的顶级权限，然后给每个顶级权限设置其子级权限，covert方法的主要用途就是从所有权限中找出相应权限的子级权限。

```
@Override
public List<UmsPermissionNode> treeList() {
    List<UmsPermission> permissionList = permissionMapper.selectByExample(new UmsPermissionExampl
    List<UmsPermissionNode> result = permissionList.stream()
            .filter(permission -> permission.getPid().equals(0L))
            .map(permission -> covert(permission, permissionList)).collect(Collectors.toList());
    return result;
}
```

为每个权限设置子级权限

这里我们使用filter操作来过滤出每个权限的子级权限，由于子级权限下面可能还会有子级权限，这里我们使用递归来解决。但是递归操作什么时候停止，这里把递归调用方法放到了map操作中去，当没有子级权限时filter下的map操作便不会再执行，从而停止递归。

```
/**
* 将权限转换为带有子级的权限对象
* 当找不到子级权限的时候map操作不会再递归调用covert
*/
private UmsPermissionNode covert(UmsPermission permission, List<UmsPermission> permissionList) {
    UmsPermissionNode node = new UmsPermissionNode();
    BeanUtils.copyProperties(permission, node);
    List<UmsPermissionNode> children = permissionList.stream()
            .filter(subPermission -> subPermission.getPid().equals(permission.getId()))
            .map(subPermission -> covert(subPermission, permissionList)).collect(Collectors.toList
    node.setChildren(children);
    return node;
}
```

项目源码地址

https://github.com/macrozheng/mall-learning/tree/master/mall-tiny-stream

推荐阅读

- **SpringBoot Admin 2.0** 详解
- **IDEA**中的**Git**操作，看这一篇就够了！
- **10**分钟搭建自己的**Git**仓库
- 那些年，我们见过的 **Java** 服务端乱象
- 我的**Github**开源项目，从**0**到**20000 Star**！
- **Postman**：**API**接口调试利器

欢迎关注，点个在看

收录于合集 #mall学习教程（技术要点篇） 17

上一篇
前后端分离项目，如何解决跨域问题

下一篇
仅需四步，整合SpringSecurity+JWT实现登录认证！

阅读原文

喜欢此内容的人还喜欢

项目中到底该不该用Lombok?
macrozheng