1. java stream

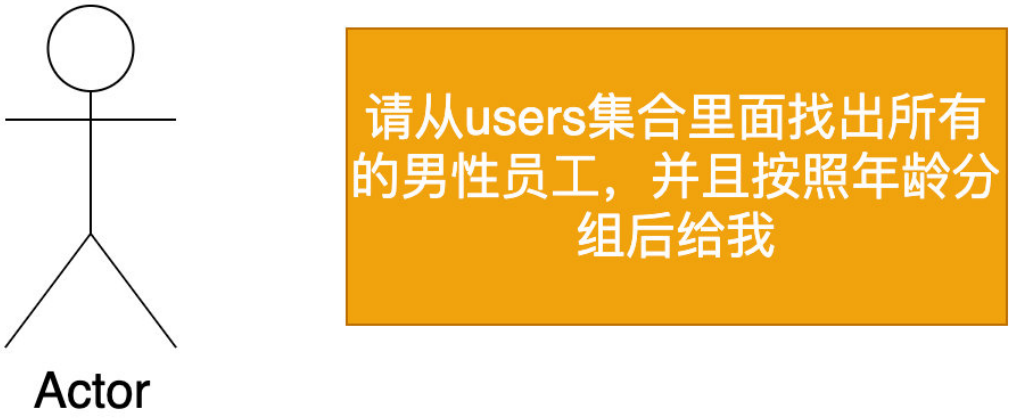
1.1. Stream是什么

第一个不要误会理解为Java里面的IO流。

java8出现的Stream是一种新的抽象，该抽象允许您以声明方式处理数据。此外，流可以利用多核体系结构，而无需编写一行多线程代码。一听就很是舒服嘛。

1.2. 为什么要引入流

Java当中集合处理还不够完美，比如我们要从某个集合里面获取某些数据，就如同我们能够编写SQL一般来得到我们想要的数据，在stream之前，你只能自己写一堆的逻辑去完成。



SQL伪代码

```
select xx from users where gender = 'm'
group by age
```

stream可以轻松帮我们完成类似的事情

伪代码

```
stream.filter(something).collect(groupbysongmething)
```

1.3. 如何创建流

方式很多，我们可以使用一个数组转换得到Stream对象。

通过使用of方法，它有一个重载的方法，为不定长参数。

1.3.1. of

```
package com.github.qiudaozhang;

import java.util.stream.Stream;

/**
```

```
 * 邱道长
 * 2020/2/25
 */
public class CreationStream {

    public static void main(String[] args) {
        String[] names =
{"james","wade","bosh"};
        Stream<String> stream =
Stream.of(names);
    }
}
```

如果你已经有一个集合，可以调用stream方法轻松得到其对象。

1.3.2. 集合.stream

```
List<String> l =
Arrays.asList("james","wade","bosh");
Stream<String> stream = l.stream();
```

1.3.3. builder and build

```
Stream.Builder<String> builder =
Stream.builder();
builder.accept("james");
builder.accept("wade");
builder.accept("bosh");
Stream<String> stream = builder.build();
```

1.4. 流遍历

foreach

```
List<String> l =
Arrays.asList("james","wade","bosh");
Stream<String> stream = l.stream();
stream.forEach(System.out::println);
```

注意它是一个terminal操作，使用一次后就不能再用了，再次使用。会出现错误

```
Exception in thread "main"
java.lang.IllegalStateException: stream has
already been operated upon or closed
```

1.5. 流映射

可以将旧流转新流。

```
package com.github.qiudaozhang;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
/**
 * 邱道长
 * 2020/2/25
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {

    private int id;
    private String name;
}
```

```java
package com.github.qiudaozhang;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Stream;
/**
 * 邱道长
 * 2020/2/25
 */
public class UserRepository {
    public static List<User> users = new
ArrayList<>();
    static  {
        users.add(new User(1,"jame"));
        users.add(new User(2,"bosh"));
        users.add(new User(3,"wade"));
    }

    public User findById(int id) {

        for(User u:users) {
            if(id == u.getId())
                return u;
        }
        return null;
    }
}
```

```
package com.github.qiudaozhang;

import java.util.stream.Stream;
/**
 * 邱道长
 * 2020/2/25
 */
public class MapDemo {

    public static void main(String[] args) {
        Integer[] ids = {1,3,5};
        UserRepository userRepository = new
UserRepository();
        Stream<User> userStream =
Stream.of(ids).map(userRepository::findById);

 userStream.forEach(System.out::println);
    }
}
```

1.6. 收集

collect

```
package com.github.qiudaozhang;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
public class CollectDemo {

    public static void main(String[] args) {
        Integer[] ids = {1,3,5};
```

```
        UserRepository userRepository = new
UserRepository();
        List<User> collect =
Stream.of(ids).map(userRepository::findById).co
llect(Collectors.toList());
        System.out.println(collect);
    }
}
```

其实就是它帮我们变为了一个List。如果你希望转换为set调用toSet即可。

1.7. 过滤

filter，过滤出你想要的数据。

```
package com.github.qiudaozhang;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;
/**
 * 邱道长
 * 2020/2/25
```

```
 */
public class FilterDemo {

    public static void main(String[] args) {
        Integer[] ids = {1,3,4,5,6};
        List<Integer> collect =
Stream.of(ids).filter(i -> i % 2 ==
0).collect(Collectors.toList());
        System.out.println(collect);

    }
}
```

注意这是满足的就留下。

可以添加多次过滤规则。

```
 Integer[] ids = {1,3,4,5,6};
        List<Integer> collect = Stream.of(ids).
          filter(i -> i % 2 == 0).
          filter(i -> i > 4).
          collect(Collectors.toList());
```

1.8. 抓住第一个

findFirst

```
package com.github.qiudaozhang;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
public class FindFirst {

    public static void main(String[] args) {
        Integer[] ids = {1, 3, 4, 5, 6};
        Optional<Integer> first =
Stream.of(ids).
          filter(i -> i % 2 == 0)
                .findFirst();
        System.out.println(first.get());

    }
}
```

1.9. 复杂数据扁平化

```
package com.github.qiudaozhang;

import java.util.Arrays;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

/**
 * 邱道长
 * 2020/2/25
 */
public class FlatMapDemo {


    public static void main(String[] args) {
        List<List<String>> lists =
Arrays.asList(
                Arrays.asList("深圳", "广州", "佛
山"),
                Arrays.asList("株洲", "岳阳", "湘
潭")

        );
```

```
        List<String> collect =
lists.stream().flatMap(Collection::stream).coll
ect(Collectors.toList());
        System.out.println(collect);
    }
}
```

其意思就是扁平化到一个list里面。

1.10. 中间操作一下

某些数据我们中途可能要处理一下。

```
package com.github.qiudaozhang;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

/**
 * 邱道长
 * 2020/2/25
 */
public class PeekDemo {

    public static void main(String[] args) {
        List<String> l =
Arrays.asList("james","wade","bosh");
        List<String> collect = l.stream()
                .peek(System.out::println)
                .collect(Collectors.toList());
    }
}
```

or

```
List<User> users = UserRepository.findAll();
        List<User> collect = users.stream()
                .peek(it ->
it.setName(it.getName().toUpperCase()))
                .collect(Collectors.toList());
        System.out.println(collect);
```

1.11. 统计

计算过滤后的数据有多少，only care this。

```
 package com.github.qiudaozhang;

import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
public class CountDemo {

    public static void main(String[] args) {
```

```
        Integer[] ids = {1, 3, 4, 5, 6};

        long count = Stream.of(ids).filter(it -
> it % 2 == 0).count();
        System.out.println(count);
    }
}
```

1.12. 跳跃

假设我和明确，我对前面3个数据不感兴趣。

```
package com.github.qiudaozhang;

import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
public class SkipDemo {
```

```
    public static void main(String[] args) {
        Integer[] ids = {12, 34, 4, 5,
6,32,446,667};
        long count =
Stream.of(ids).skip(3).filter(it -> it % 2 ==
0).count();
        System.out.println(count);
    }
}
```

1.13. 限制

我一次就吃得下三个，我得限制一下。

```
package com.github.qiudaozhang;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
```

```
public class LimitDemo {

    public static void main(String[] args) {

        Integer[] ids = {12, 5, 4, 9,
6,32,446,667};
        List<Integer> collect =
Stream.of(ids).limit(3).filter(it -> it % 2 ==
0).collect(Collectors.toList());
        System.out.println(collect);
    }
}
```

1.14. 排序

我们有一堆大小随意的数字，限制要排序得到。

```
package com.github.qiudaozhang;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
```

```
 * 邱道长
 * 2020/2/25
 */
public class SortDemo {

    public static void main(String[] args) {

        Integer[] ids = {12, 5, 4, 9,
6,32,23,667};
        List<Integer> collect =
Stream.of(ids).sorted().collect(Collectors.toLi
st());
        System.out.println(collect);

    }
}
```

默认是升序的，你可以自定义规则，加入比较器即可。

降序

```
package com.github.qiudaozhang;

import java.util.List;
import java.util.stream.Collectors;
```

```
import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
public class SortDescDemo {

    public static void main(String[] args) {

        Integer[] ids = {12, 5, 4, 9,
6,32,23,667};
        List<Integer> collect =
Stream.of(ids).sorted((x, y) -> y -
x).collect(Collectors.toList());
        System.out.println(collect);

    }
}
```

1.15. 寻找最大与最小

min max

```java
package com.github.qiudaozhang;

import java.util.Optional;
import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
public class MinMaxDemo {

    public static void main(String[] args) {

        Integer[] ids = {12, 5, 4, 9,
6,32,23,667};
        Optional<Integer> max =
Stream.of(ids).max((x, y) -> x - y);
        System.out.println(max.get());
        Optional<Integer> min =
Stream.of(ids).min((x, y) -> x - y);
        System.out.println(min.get());
    }
}
```

1.16. 去重复

distinct

```
package com.github.qiudaozhang;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
public class DistinctDemo {

    public static void main(String[] args) {

        Integer[] ids = {12, 5, 5, 9,
12,32,23,32};

        List<Integer> collect =
Stream.of(ids).distinct().collect(Collectors.to
List());
        System.out.println(collect);
```

```
    }
}
```

1.17. 匹配三兄弟

- allMatch
- anyMatch
- noneMatch

```
package com.github.qiudaozhang;

import java.util.Arrays;
import java.util.List;

/**
 * 邱道长
 * 2020/2/25
 */
public class MatchDemo {

    public static void main(String[] args) {
```

```
        List<String> l =
Arrays.asList("james","wade","bosh");
        boolean b = l.stream().allMatch(it ->
it.length() == 4);
        System.out.println(b);
    }
}
```

显然james不是4长度，是5，所有无法完全匹配。

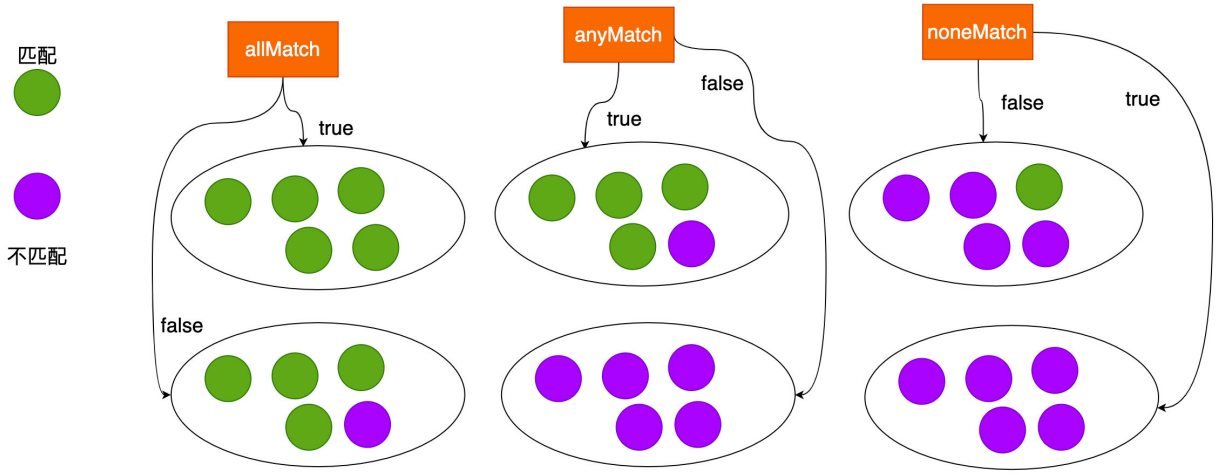
```
package com.github.qiudaozhang;

import java.util.Arrays;
import java.util.List;

/**
 * 邱道长
 * 2020/2/25
 */
public class MatchDemo {

    public static void main(String[] args) {
        List<String> l =
Arrays.asList("james","wade","bosh");
```

```
        boolean b = l.stream().anyMatch(it ->
it.length() == 4);
        System.out.println(b);
    }
}
```

wade和bosh都是4，有满足的了就OK了，所以true。

```
package com.github.qiudaozhang;

import java.util.Arrays;
import java.util.List;

/**
 * 邱道长
 * 2020/2/25
 */
public class MatchDemo {

    public static void main(String[] args) {
        List<String> l =
Arrays.asList("james","wade","bosh");
        boolean b = l.stream().noneMatch(it ->
it.length() == 4);
        System.out.println(b);
    }
}
```

wade bosh都是满足，并不是都不满足，所以必然是false。



1.18. 数字转换求平均

```
package com.github.qiudaozhang;

import java.util.OptionalDouble;
import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
```

```
public class AvgDemo {

    public static void main(String[] args) {
        Integer[] ids = {1,33,5};
        OptionalDouble average =
Stream.of(ids).mapToDouble(it ->
Double.parseDouble(it+"")).average();

 System.out.println(average.getAsDouble());
    }
}
```

1.19. 并流单值

reduce，可以根据指定计算模型生产出你想要的东西。

```
public class ReduceDemo {

    public static void main(String[] args) {
      Integer[] ids = {1,3,5};
        Optional<Integer> reduce =
Stream.of(ids).reduce(Integer::sum);
        System.out.println(reduce.get());
    }
}
```

or

```
List<String> l =
Arrays.asList("james","wade","bosh");
Optional<String> reduce = l.stream().reduce((x,
y) -> x + "-" + y);
System.out.println(reduce.get());
```

可以合并名称为james-wade-bosh。

1.20. 单纯的合并

join

```
package com.github.qiudaozhang;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
 * 邱道长
 * 2020/2/25
 */
public class JoinDemo {

    public static void main(String[] args) {
        List<String> l =
Arrays.asList("james","wade","bosh");
        String collect =
l.stream().collect(Collectors.joining("~ my dog
~"));
        System.out.println(collect);
    }
}
```

1.21. 分组

```
package com.github.qiudaozhang;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * 邱道长
 * 2020/2/25
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Player {

    private String team;
    private String name;
}
```

```java
package com.github.qiudaozhang;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

/**
 * 邱道长
 * 2020/2/25
 */
public class GroupByDemo {

    public static void main(String[] args) {
        List<Player> players = Arrays.asList(
                new Player("湖人","james"),
                new Player("湖人","ad"),
                new Player("热火","jb"),
                new Player("热火","ig"),
                new Player("湖人","ab")
        );

        Map<String, List<Player>> collect =
players.stream().collect(Collectors.groupingBy(
it -> it.getTeam()));
        System.out.println(collect);
```

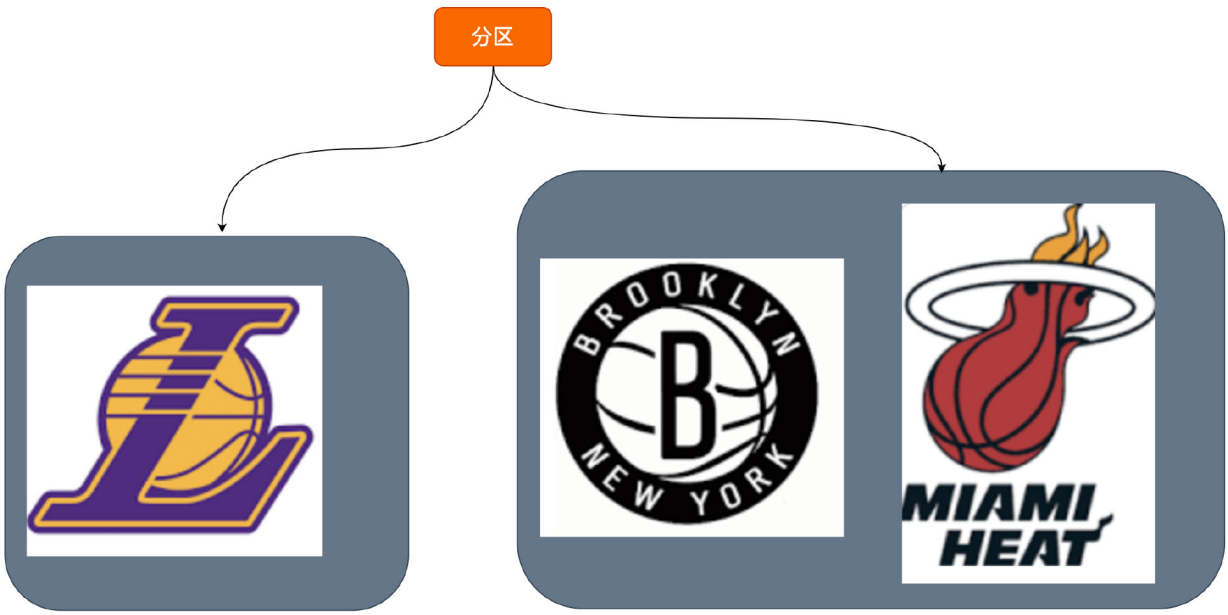
```
    }
}
```

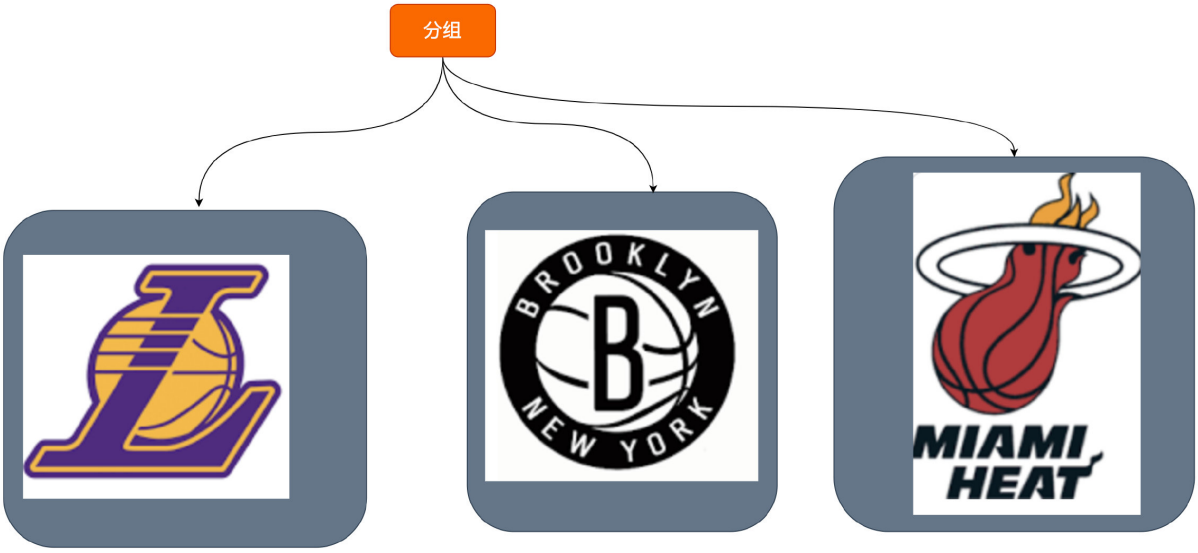
1.22. 分区

partition

```
package com.github.qiudaozhang;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

/**
 * 邱道长
 * 2020/2/25
 */
public class PartitionDemo {

    public static void main(String[] args) {
        List<Player> players = Arrays.asList(
                new Player("湖人","james"),
                new Player("湖人","ad"),
```

```
                new Player("热火","jb"),
                new Player("热火","ig"),
                new Player("篮网","kd"),
                new Player("湖人","ab")
        );

        Map<Boolean, List<Player>> lakers =
players.stream().collect(Collectors.partitionin
gBy(it -> it.getTeam().equals("湖人")));
        System.out.println(lakers);


    }
}
```




分组
BROOKLYN
B
NEW YORK
MIAMI
HEAT