作业三 Websocket与线程安全

> 张子谦 520111910121
>
> Tips：其实上一次作业已经用WS实现了相关的功能，所以附一个上一次作业的PDF版本，里面在具体的实现可能会更加详细一些。

- 能够将订单处理的结果通过 WebSocket 方式返回给前端，并且能够在前端正确地呈现（2分）
- 能够正确地对订单用户进行筛选，即对于某个订单来说，只有下该订单的用户才会收到订单处理完成的消息（1分）
- 回答为什么要选择线程安全的集合类型来维护客户端 Session，而你选择的类型为什么是线程安全的（2 分）

一、线程安全相关问题解释

a）理由

问题：为什么要选择线程安全的集合类型来维护客户端 Session?

如果有大量的用户（或者说客户端进入到Websocket连接），那么这个时候要做的动作就是把Session放入一个集合中【我用的集合是ConcurrentHashMap】，也就是在往集合里面写入内容，如果不能保证线程安全，由于在往集合中写入内容这一个操作不是原子操作，甚至涉及到很多复杂的过程，也就是说这些线程直接会相互干扰影响（我们可以举一个最简单的例子，哪怕i++这一行代码从汇编来看都涉及到三个操作，从内存拷贝、自加、然后复制回去内存），最终导致的结果就是：可能同时有100个用户涌入进来，但是由于线程之间的干扰，最终能够维持的Session就只有80或者90多个，会有遗漏。为此，我们需要使用线程安全的集合来维护Sessions集合。

b）原理

为了研究为什么它是线程安全的，我打开了ConcurrentHashMap.Java的源码：总体来说：ConcurrentHashMap大部分的逻辑代码和HashMap是一样的，主要通过synchronized来保证新数据节点插入的线程安全。

1、初始化的线程安全

在 JDK 1.8 中，初始化ConcurrentHashMap 的时候这个 Node[] 数组是还未初始化的，会等到第一次 put() 方法调用时才初始化，sizeCtl 变量注释如下

> Table initialization and resizing control. When negative, the table is being initialized or resized: -1 for initialization, else -(1 + the number of active resizing threads). Otherwise, when table is null, holds the initial table size to use upon creation, or 0 for default. After initialization, holds the next element count value upon which to resize the table.

```
// 表初始化和调整控件大小。如果为负值，则表正在初始化或调整大小：-1用于初始化，否则-（1+活动调整大小线程的数量）
// 否则，当table为null时，将保留创建时使用的初始表大小，默认值为0。初始化后，保存下一个要调整表大小的元素计数值
private transient volatile int sizeCtl;
```

2、put操作的线程安全

- 如下面的代码所示：
- `tabAt(tab, i)`方法使用 Unsafe 类 volatile 的操作查看值，保证每次获取到的值都是最新的
- `putVal()` 方法的核心在于其减小了锁的粒度，若 Hash 完美不冲突的情况下，可同时支持 n 个线程同时 put 操作，n 为 Node 数组大小，在默认大小 16 下，可以支持最大同时 16 个线程无竞争同时操作且线程安全。
- synchronized 同步锁：如果此时拿到的最新的 Node 不为 null，则说明已经有线程在此 Node 位置进行了插入操作，此时就产生了 hash 冲突；此时的synchronized 同步锁就起到了关键作用，防止在多线程的情况下发生数据覆盖（线程不安全），接着在 synchronized 同步锁的管理下按照相应的规则执行操作，
  - 当 hash 值相同并 key 值也相同时，则替换掉原 value
  - 否则，将数据插入链表或红黑树相应的节点

```
static final <K,V> Node<K,V> tabAt(Node<K,V>[] tab, int i) {
  return (Node<K,V>)U.getObjectVolatile(tab, ((long)i << ASHIFT) + ABASE);
}

final V putVal(K key, V value, boolean onlyIfAbsent) {
        // K,V 都不能为空 然后取得 key 的 hash 值
        if (key == null || value == null) throw new NullPointerException();
        int hash = spread(key.hashCode());
        // 用来计算在这个节点总共有多少个元素，用来控制扩容或者转换为树
        int binCount = 0;
        // 数组的遍历，自旋插入结点，直到成功
        for (Node<K,V>[] tab = table;;) {
            Node<K,V> f; int n, i, fh;
            // 当Node[]空时，进行初始化
            if (tab == null || (n = tab.length) == 0)
                tab = initTable();
            // 此时Node位置若为 null，则表示还没有线程在此 Node 位置进行插入操作，说明本次操作是第一次
            else if ((f = tabAt(tab, i = (n - 1) & hash)) == null) {
                if (casTabAt(tab, i, null,
                             new Node<K,V>(hash, key, value, null)))
                    break;                   // no lock when adding to empty bin
            }
            // 如果检测到某个节点的 hash 值是 MOVED，则表示正在进行数组扩容
            // 那么就开始帮助扩容
            else if ((fh = f.hash) == MOVED)
                tab = helpTransfer(tab, f);
            else {
                V oldVal = null;
                synchronized (f) {
                    if (tabAt(tab, i) == f) {
                        if (fh >= 0) {
                            binCount = 1;
                            for (Node<K,V> e = f;; ++binCount) {
```

```
                                K ek;
                                if (e.hash == hash &&
                                    ((ek = e.key) == key ||
                                     (ek != null && key.equals(ek)))) {
                                    oldVal = e.val;
                                    if (!onlyIfAbsent)
                                        e.val = value;
                                    break;
                                }
                                Node<K,V> pred = e;
                                if ((e = e.next) == null) {
                                    pred.next = new Node<K,V>(hash, key,
                                                              value, null);
                                    break;
                                }
                            }
                        }
                        else if (f instanceof TreeBin) {
                            Node<K,V> p;
                            binCount = 2;
                            if ((p = ((TreeBin<K,V>)f).putTreeVal(hash, key,
                                                           value)) != null) {
                                oldVal = p.val;
                                if (!onlyIfAbsent)
                                    p.val = value;
                            }
                        }
                    }
                }
                if (binCount != 0) {
                    if (binCount >= TREEIFY_THRESHOLD)
                        treeifyBin(tab, i);
                    // 如果本次put操作只是替换了旧值，不用更改计数值 之间结束函数
                    if (oldVal != null)
                        return oldVal;
                    break;
                }
            }
        }
        // 计数值加1
        addCount(1L, binCount);
        return null;
    }
```

二、筛选客户端的方式设计

由于我们之前用了一个哈希表维护Session，我在设计Hash表的时候：

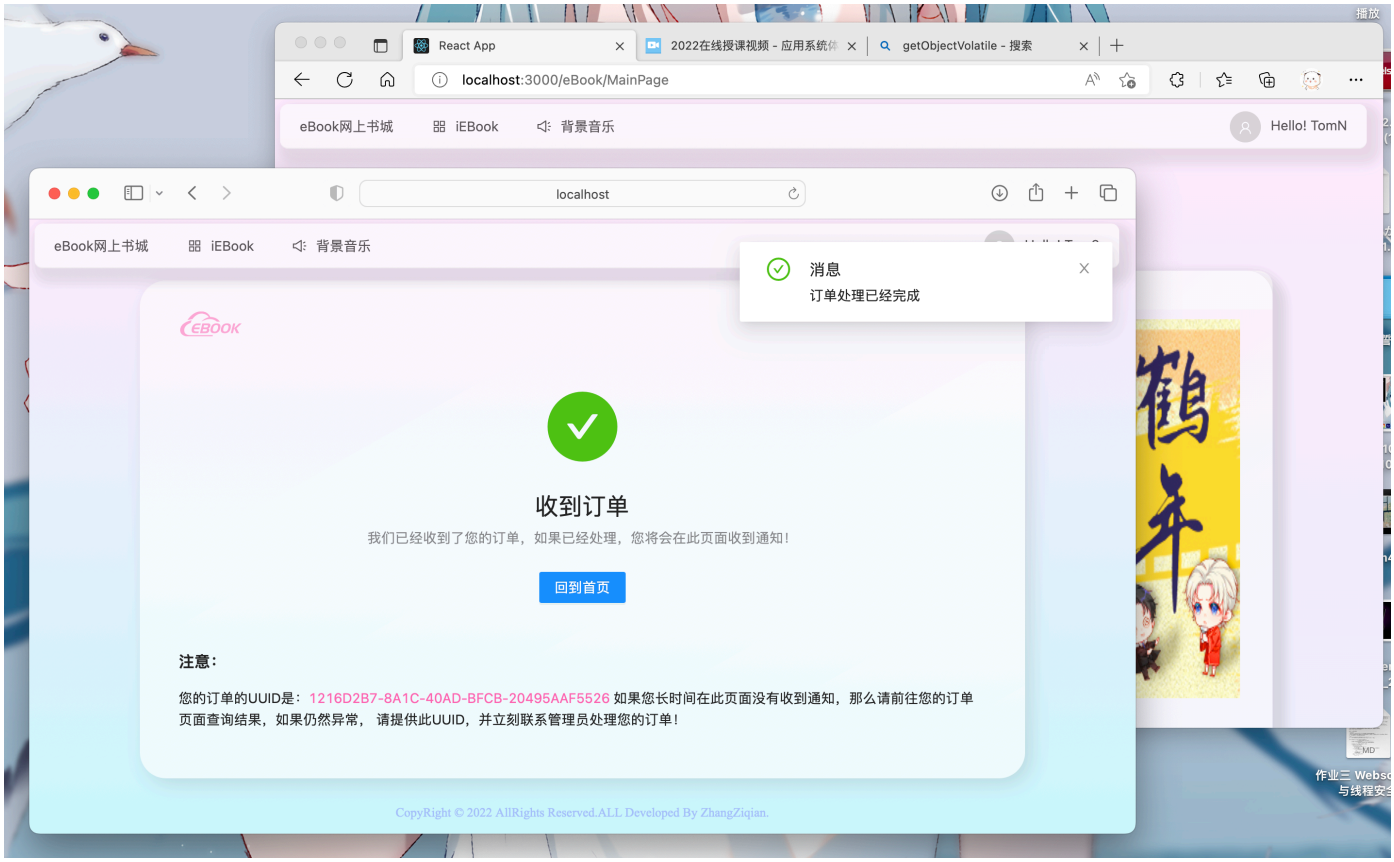
- Key采用的是用户名的Sha-256加密的String值
- Val对应的是这个用户的Session

我这里通过一个函数，指定接收的用户，考虑到怕发送不成功，就多发了几次。

```
@ServerEndpoint("/websocket/transfer/{userId}")
@Component
public class WebSocketServer {
    // ...

    public void sendMessageToUser(String user, String message) throws
InterruptedException {
        System.out.println( "sendMessageToUser output  " + user);
        for (int i = 0; i < 10; i++){
            Session toSession = SESSIONS.get(user);
            if(sendMessage(toSession, message) == 0)
                return;
            Thread.sleep(1000);
        }
    }
    // ...
}
```

效果如下图所示（只有当前用户收到订单，在另外一个窗口登录的另外一个用户不会收到通知）：

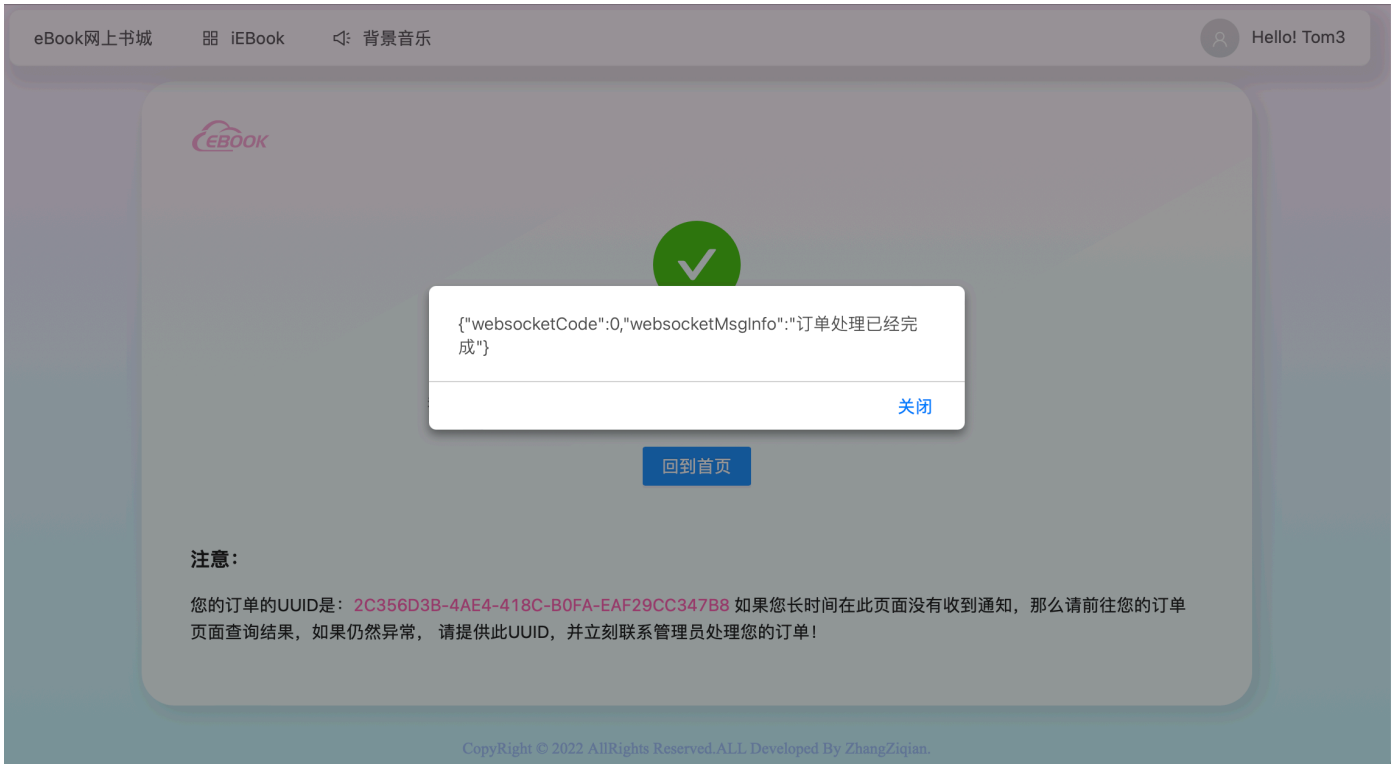


三、包括WebSocket的消息格式

包含两个主要内容：

- 消息代码：区别成功或者失败订单处理结果的数字，0代表成功，1代表失败
- 消息内容：包含消息处理成功的内容，或者处理出错的原因解释。

```
{"websocketCode":0,"websocketMsgInfo":"订单处理已经完成"}
```



四、相关代码实现

订单完成页面

- 当订单完成后，会跳转到订单完成的页面，在这里创建一个WebSocket连接，连接的地址通过SHA256加密用户名以及拼接URL得到。
- 建立连接后，传入相关的处理回掉函数，一旦接受到消息，就调用回调函数。
- 回掉函数的最后关闭Websocket连接，因为从目前的功能来看不再需要连接，释放资源。

```
1
2  class purchaseSuccess extends React.Component{
3      orderUUID = "";
4      socketURL = "";
5      constructor() {
6          super();
7          let url = decodeURI(window.location.search);
8          //获取url中"?"符后的字串 ('?modFlag=business&role=1')
9          let theRequest = urlDecoder(url);
10         console.log(theRequest);
```

```
        if(theRequest["orderUUID"]!= null ){
            this.orderUUID = theRequest["orderUUID"];
            let SHA256 = require("crypto-js/sha256");
            this.socketURL = "ws://localhost:8080/websocket/transfer/" +
SHA256(LoginPassport.getUserName());
            createWebSocket(this.socketURL,
                (info) => {
                    let jsonData = JSON.parse(info.data);

                    if(jsonData.websocketCode === 0){
                        reminderInfoCheck('success', jsonData.websocketMsgInfo);
                    }
                    else if(jsonData.websocketCode === 1){
                        reminderInfoCheck('warning', jsonData.websocketMsgInfo);
                    }
                    closeWebSocket();
                }
            );
        }
    }

    render() {
        return(
            <div className="eBookPageContainer">
                // 此处内容省略
            </div>
        );
    }
}
export default purchaseSuccess;
```

前端组件代码

通过util的组件自实现了一个带有发送心跳包小组件，由于他是全局性质的，所以经过测试，哪怕用户跳转到了别的页面，用户同样可以接收到订单完成的消息。

```
let websocket, lockReconnect = false;
let createWebSocket = (url, handleEvent) => {
    websocket = new WebSocket(url);
    websocket.onopen = function () {
        heartCheck.reset().start();
    }
    websocket.onerror = function () {
        reconnect(url);
    };
    websocket.onclose = function (e) {
        console.log('websocket 断开: ' + e.code + ' ' + e.reason + ' ' +
e.wasClean)
    }
```

```
    websocket.onmessage = function (event) {
        lockReconnect=true;
        handleEvent(event);
        //event 为服务端传输的消息，在这里可以处理
    }
}

let reconnect = (url) => {
    if (lockReconnect) return;
    //没连接上会一直重连，设置延迟避免请求过多
    setTimeout(function () {
        createWebSocket(url);
        lockReconnect = false;
    }, 4000);
}

let heartCheck = {
    timeout: 60000, //60秒
    timeoutObj: null,
    reset: function () {
        clearInterval(this.timeoutObj);
        return this;
    },
    start: function () {
        this.timeoutObj = setInterval(function () {
            websocket.send("HeartBeat");
        }, this.timeout)
    }
}

//关闭连接
let closeWebSocket=()=> {
    websocket && websocket.close();
}
export {
    websocket,
    createWebSocket,
    closeWebSocket
};


```

后端部分代码

```
package com.zzq.ebook.utils.websocket;
import *
// 省略 import
@ServerEndpoint("/websocket/transfer/{userId}")
@Component
```

```java
public class WebSocketServer {
    public WebSocketServer() {
        //每当有一个连接，都会执行一次构造方法
        System.out.println("新的连接已经开启");
    }
    private static final AtomicInteger COUNT = new AtomicInteger();
    private static final ConcurrentHashMap<String, Session> SESSIONS = new ConcurrentHashMap<>();
    public int sendMessage(Session toSession, String message) {
        if (toSession != null) {
            try {
                toSession.getBasicRemote().sendText(message);
                return 0;
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("对方不在线");
            return 1;
        }
        return 1;
    }
    public void sendMessageToUser(String user, String message) throws InterruptedException {
        System.out.println( "sendMessageToUser output  " + user);
        for (int i = 0; i < 10; i++){
            Session toSession = SESSIONS.get(user);
            if(sendMessage(toSession, message) == 0)
                return;
            Thread.sleep(1000);
        }
    }
    @OnMessage
    public void onMessage(String message) {
        System.out.println("服务器收到消息：" + message);
    }
    @OnOpen
    public void onOpen(Session session, @PathParam("userId") String userId) {
        if (SESSIONS.get(userId) != null) {
            return;
        }
        SESSIONS.put(userId, session);
        COUNT.incrementAndGet();
        System.out.println(userId + "加入Websocket连接，当前在线人数：" + COUNT);

    }
    @OnClose
    public void onClose(@PathParam("userId") String userId) {
        SESSIONS.remove(userId);
```

```
        COUNT.decrementAndGet();
        System.out.println(userId + "推出Websocket连接，当前在线人数：" + COUNT);
    }
    @OnError
    public void onError(Session session, Throwable throwable) {
        System.out.println("发生错误");
        throwable.printStackTrace();
    }
}
```

卡夫卡监听器

- 还是保留了两个卡夫卡信箱的特性，尽管课堂上已经说了第二个卡夫卡没有必要hhh

```
package com.zzq.ebook.utils.listener;

import com.zzq.ebook.constant.constant;
import com.zzq.ebook.service.OrderService;
import com.zzq.ebook.utils.tool.ToolFunction;
import com.zzq.ebook.utils.websocket.WebSocketServer;
import net.sf.json.JSONObject;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;
import java.util.Map;
import java.util.Objects;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import static com.zzq.ebook.utils.tool.ToolFunction.getSHA256StrJava;

@Component
public class OrderListener {
    @Autowired
    private OrderService orderService;
    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;
    @Autowired
    private WebSocketServer webSocketServer;

    @KafkaListener(topics = "orderQueue", groupId = "group_topic_order")
    public void orderQueueListener(ConsumerRecord<String, String> record) throws
Exception {
        Map<String,String> params = ToolFunction.mapStringToMap(record.value());
        int itemNum = (params.size() - 6) / 2 ;
        String orderFrom = params.get("orderFrom");
```

```
        String username = params.get(constant.USERNAME);
        String receivename = params.get("receivename");
        String postcode = params.get("postcode");
        String phonenumber = params.get("phonenumber");
        String receiveaddress = params.get("receiveaddress");
        int[] bookIDGroup = new int[itemNum];
        int[] bookNumGroup = new int[itemNum];

        for(int i=1; i<=itemNum; i++){
            bookIDGroup[i-1] = Integer.parseInt(params.get("bookIDGroup" + i));
            bookNumGroup[i-1] = Integer.parseInt(params.get("bookNumGroup" + i));
        }

        JSONObject respData = new JSONObject();
        // 根据购买的来源，把数组交给服务层业务函数
        try {
            int result = -1;
            if(Objects.equals(orderFrom, "ShopCart")) {
                result =
orderService.orderMakeFromShopCart(bookIDGroup,bookNumGroup,username,receivename,
                        postcode, phonenumber, receiveaddress,itemNum);

            }
            else if(Objects.equals(orderFrom, "DirectBuy")){
                result =
orderService.orderMakeFromDirectBuy(bookIDGroup,bookNumGroup,username,receivename,
                        postcode, phonenumber, receiveaddress,itemNum);
            }
            else {

 respData.put(constant.WEBSOCKET_MSG_CODE,constant.WEBSOCKET_MSG_CODE_Info_Error);

 respData.put(constant.WEBSOCKET_MSG_Info,constant.OrderDeal_MSG_ERROR_POST_PARAME
TER);
            }

 respData.put(constant.WEBSOCKET_MSG_CODE,constant.WEBSOCKET_MSG_CODE_Info_Success
);

 respData.put(constant.WEBSOCKET_MSG_Info,constant.OrderDeal_MSG_Success);

        }catch (Exception e){

 respData.put(constant.WEBSOCKET_MSG_CODE,constant.WEBSOCKET_MSG_CODE_Info_Error);

 respData.put(constant.WEBSOCKET_MSG_Info,constant.OrderDeal_MSG_ERROR_TRAINSITION
);
        }

```

```
        kafkaTemplate.send("orderFinished",  getSHA256StrJava(username), respData.toString());
    }

    @KafkaListener(topics = "orderFinished", groupId = "group_topic_order")
    public void orderFinishedListener(ConsumerRecord<String, String> record) throws InterruptedException {
        String key = record.key();
        System.out.println(key);
        webSocketServer.sendMessageToUser(key, record.value());
    }

}
```