姓名：Yan Yi, Celine Loh 骆彦伊

学号：520030990040

HW1

Part A

1. 在你设计的负责用户登录和登出的 Controller 中，在登录方法(login)被调用 时，调用这个定时器 Service，在其中初始化计时器，并开始计时。

```
@RequestMapping("/login")
public Message login(@RequestBody Map<String, String> params) {
    System.out.println("--- login ---");
    String username = params.get("username");
    String userPassword = params.get("userPassword");
    UserAuth userAuth = userService.checkAuth(username, userPassword);
    System.out.println("login Auth:"+ userAuth);
    if (userAuth != null) {
        User nowUser=userService.getUserById(userAuth.getUserId());
        if(nowUser.getEnabled()==false)
        {
            return MessageUtil.createMessage(MessageUtil.LOGIN_ERROR_CODE, MessageUtil.LOGIN_BAN_MSG);
        }
        timerService.startTimer();
          watch = new StopWatch();
          watch.start();
```

2. 在登出方法(logout)被调用时，调用这个定时器 Service，在其中停止计时器， 并获取计时器的计时值，返回给 logout，logout 方法会返回给前端显示这个 时间；

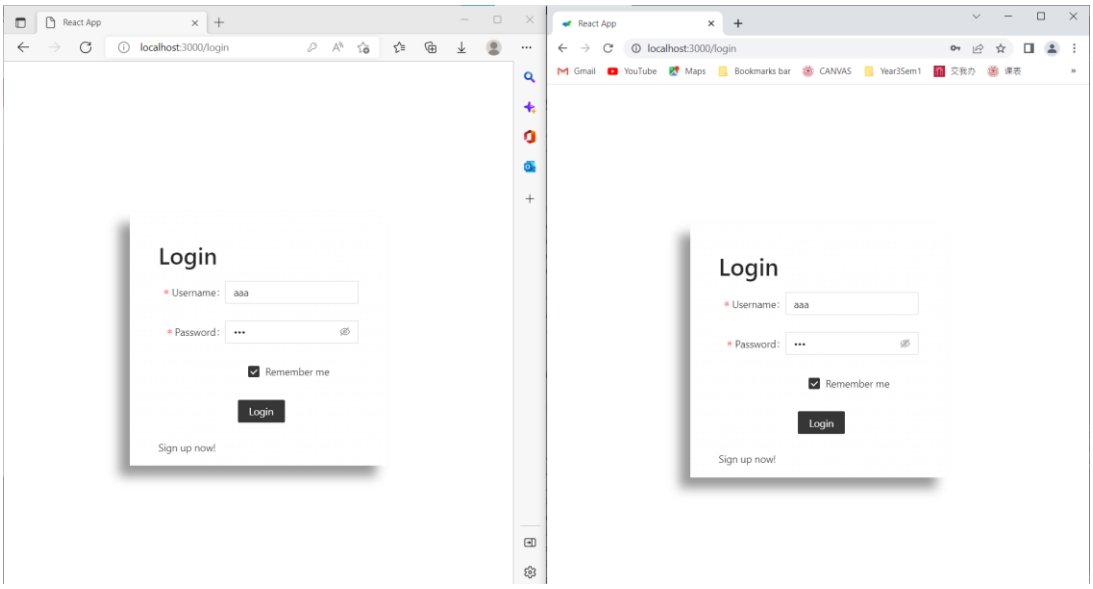
```
@RequestMapping("/logout")
public Message logout() {
    boolean status = SessionUtil.removeSession();
    System.out.println("--- logout ---");
    System.out.println("logout:"+status);

    String timeused = timerService.stopTimer();
      String timeused =watch.getTime()+" ms";
      watch.stop();
      System.out.println("logout! Time used: "+timeused);

    if (!status) {
        return MessageUtil.createMessage(MessageUtil.LOGOUT_ERROR_CODE, MessageUtil.LOGOUT_ERROR_MSG);
    } else
          return MessageUtil.createMessage(MessageUtil.LOGOUT_SUCCESS_CODE, MessageUtil.LOGOUT_SUCCESS_MSG );
        return MessageUtil.createMessage(MessageUtil.LOGOUT_SUCCESS_CODE,  message: MessageUtil.LOGOUT_SUCCESS_MSG +" Timer: "+timeused );
}
```

3. 在编写代码时，请正确配置 Controller 和 Service 的@Scope 属性
Controller 和 Service 的@Scope 属性设置为 session。

```
@RestController
@Scope("session")
public class LoginController {
```

```
@Service
@Scope("session")
public class TimerServiceImpl  implements TimerService {
```

4. 请你使用多个浏览器同时登录你的系统，并且在登出时观察它们获得的计数值。



登出时分别获得的计数值：

从上面的截图可以看到当使用多个 browser 来同时 login，并且当 logout 时，得到的计时值是不一样的，因为在 Controller 和 Service 的@Scope 属性设置为了 session，使得同一个 Http Session 共用一个 Bean，所以在同一个 browser login 和 logout 会同步，而不同 browser 之间不会同步。如果使用了 singleton，那这多个 browser 得到的计时会是一样的值，而如果使用了 prototype，使用同一个 browser login 时，会产生多个 object instance， 那 login 和 logout 将不会同步，不同 browser 也一样。

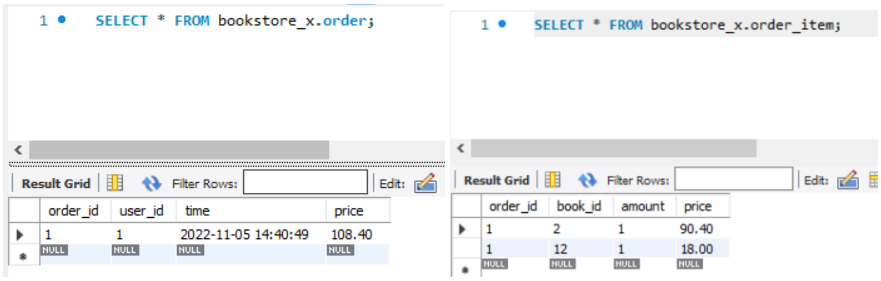
Part B

1. 假设你的代码中在下订单的 Service 中，需要完成两件事情，在数据库的 Order 表中插入一条记录，在 OrderItem 表中插入多条记录。例如，如果一个订单包含 3 种不同的书，那么在 Order 表中会插入一条记录，在 OrderItem 表中会插入 3 条记录；按照上述操作，Service 需要调用 OrderDao 和 OrderItemDao 两个对象的响 应方法来实现下订单，而且必须保证两个表的数据要么都插入成功，要么 都不插入，即这两个表的插入操作必须在一个事务中完成。

   下订单的逻辑如下：

   a. 添加订单：cartDao.addOrder
   b. 添加订单项：cartDao.addOrderItem
   c. 删除购物车：cartDao.deleteCart
   d. 更新图书库存：cartDao.updateBookInventory

   默认情况下，service 层的 submitCart方法使用的是 REQUIRED 属性，Dao 层下订单的四个方法也一样使用的是 REQUIRED 属性。正常情况下，下订单后：



   如果出现错误，可以看到有关 rollback 的报错。

Case	Submit cart	Add Order	Add Order Item	Result
1	正常	正常	正常	正常下订单
2	Result = 10/0	正常	正常	整个事务回滚
3	正常	Result = 10/0	正常	整个事务回滚
4	正常	正常	Result = 10/0	整个事务回滚
5	正常	正常	正常 REQUIRES_NEW	正常下订单
6	Result = 10/0, 在 add order 之前	正常	正常 REQUIRES_NEW	整个事务回滚
7	Result = 10/0, 在 add order 之后	正常	正常 REQUIRES_NEW	Add order item 成功，而其他事务回滚
8	正常	Result = 10/0	正常 REQUIRES_NEW	整个事务回滚
9	正常	正常	Result = 10/0 , REQUIRES_NEW	Add order item 回滚，而其他事务成功

Case 1:
所有 transaction 正常，结果正常。

Case 2 & Case 3 & Case 4:
当 submit cart, add order 和 add order item 的事务属性为默认的 REQUIRED 时，只要当前存在一个 transaction，后面的业务逻辑也会加到当前的 transaction 中，而如果当前没有 active 的 transaction，将创建一个新的 transaction。所以只要在 submit cart, add order 或 add order item 中的任何一个 transaction 发生异常，整个事务就会回滚。

Case 5:
Add order item 的 transaction 属性为 REQUIRES_NEW 时，将创建一个新的 transaction，也就是与 submit cart 和 add order 的 transaction 不为一个 transaction。这里既然所有 transaction 都没有异常，将可以正常下订单。

Case 6 & Case 8:
当异常发生在 submit cart 里（add order 之前）或 add order 时，由于这两个 transaction 的属性为 REQUIRED，而程序也还未跑到 add order item 这个 transaction，因此整个 transaction 回滚。

Case 7:
当异常发生在 submit cart 里（add order 之后）时，由于 add order item 这个 transaction 的属性为 REQUIRES_NEW，它与 submit cart 和 add order 的 transaction 分开，所以，add order item 可以正常执行而 submit cart 和 add order 回滚。

Case 9:
当异常发生在 add order item，由于这个 transaction 与 submit cart 和 add order 分开，所以只有 add order item 这一个 transaction 回滚，而其他不受影响，正常执行。

```
1 • SELECT * FROM bookstore_x.order;
```

```
1 • SELECT * FROM bookstore_x.order_item;
```

Result Grid | Filter Rows: | Edit:

order_id	user_id	time	price
1	1	2022-11-05 14:56:29	114.40
2	1	2022-11-05 14:57:32	195.80
11	1	2022-11-05 20:38:24	467.00
12	1	2022-11-05 20:40:01	24.00
14	1	2022-11-05 20:44:40	55.80
NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit:

order_id	book_id	amount	price
1	2	1	90.40
1	11	1	24.00
2	1	1	15.00
2	2	2	180.80
11	11	1	24.00
12	11	1	24.00
12	12	1	18.00
NULL	NULL	NULL	NULL