* Float和double使用的大坑

1. 典型现象
2. 条件判断超出预期

System.out.println( 1f == 0.9999999f );   // 打印：false  
System.out.println( 1f == 0.99999999f );  // 打印：true    纳尼？

1. 数据转换超过预期

float f = 1.1f;  
double d = (double) f;  
System.out.println(f);  // 打印：1.1  
System.out.println(d);  // 打印：1.100000023841858  纳尼？

1. 基本运算超过预期

System.out.println( 0.2 + 0.7 );

1. 数据自增超过预期
2. 原因

***浮点数的精度太差***

浮点数的表示方法是尾数部分\*2^（阶码部分），float尾数部分是23位，换算成十进制是8388608，所以十进制精度只有6~7位，double有15~16位

1. 如何解决
2. 使用字符串或者数组表示大数，自己写大数运算
3. 用JAVA里面的大数类（BigInteger和BigDecimal）

* 为什么要面向接口编程

1. 面向接口编程

事先约定好各个功能的接口（约定好接口协议），在Java中就是interface

* 枚举类型（enum）

1. 利用枚举类型进行扩展

**enum** UserRole{

    ROLE\_ROOT\_ADMIN("系统管理员",000000),

    ROLE\_ORDER\_ADMIN("订单管理员",100000),

    ROLE\_NORMAL("普通用户",200000),

    ;

**private** final String roleName;

**private** final Integer roleCode;

    UserRole( String roleName, Integer roleCode ) {

**this**.roleName = roleName;

**this**.roleCode = roleCode;

    }

**public** String getRoleName(){

**return** **this**.roleName;

    }

**public** Integer getRoleCode(){

**return** **this**.roleCode;

    }

}

1. 利用枚举加接口干掉if/else
2. 先写一个接口
   1. interface RoleOperation{
   2. String op();
   3. }
3. 写一个枚举，让每一个成员实现接口方法
   1. enum RoleEnum implements RoleOperation{
   2. ROLE\_ROOT\_ADMIN{
   3. @Override
   4. public String op() {
   5. return "ROLE\_ROOT\_ADMIN"+"HAS AAA PERMISSION";
   6. }
   7. },
   9. ROLE\_ORDER\_ADMIN{
   10. @Override
   11. public String op() {
   12. return "ROLE\_ORDER\_ADMIN"+"HAS BBB PERMISSION";
   13. }
   14. },
   16. ROLE\_NORMAL{
   17. @Override
   18. public String op() {
   19. return "ROLE\_NORMAL"+"HAS CCC PERMISSION";
   20. }
   21. }
   22. ;
   23. }
4. 调用的时候使用enum（写你自己的枚举名）.valueOf( String ).(接口名)
   1. **public** **static** String judge(String roleName){
   2. **return** RoleEnum.valueOf(roleName).op();
   3. }
   5. **public** **static** **void** main(String[] args){
   6. System.out.println(judge("ROLE\_ROOT\_ADMIN"));
   7. System.out.println(judge("ROLE\_ORDER\_ADMIN"));
   8. System.out.println(judge("ROLE\_NORMAL"));
   9. }
5. 专用于枚举的集合类

我们通常使用HashMap和HashSet来存放元素，也可以用EnumSet和EnumMap来存放枚举

* 1. // 定义一个管理员角色的专属集合
  2. EnumSet<UserRole> userRolesForAdmin
  3. = EnumSet.of(
  4. UserRole.ROLE\_ROOT\_ADMIN,
  5. UserRole.ROLE\_ORDER\_ADMIN
  6. );
  8. // 判断某个进来的用户是不是管理员
  9. Boolean isAdmin( User user ) {
  11. if( userRoles.contains( user.getUserRole() ) )
  12. returntrue;
  14. returnfalse;
  15. }

HashMap专门存放枚举类型为key的Map类型，比如系统新增了一批人，我们需要统计这批人不同角色的数量，就可以使用HashMap

* 1. Map<UserRole,Integer> userStatisticMap = new EnumMap<>(UserRole.class);
  3. for ( User user : userList ) {
  4. Integer num = userStatisticMap.get( user.getUserRole() );
  5. if( null != num ) {
  6. userStatisticMap.put( user.getUserRole(), num+1 );
  7. } else {
  8. userStatisticMap.put( user.getUserRole(), 1 );
  9. }
  10. }