4-5、结构型模式之---门面模式

（1）目的

- **为了让客户端不再需要关注应该使用service接口的哪个实现类，而是直接调用门面类提供的方法**，而门面类中通过code去获取service接口中的某个实现类（Map<code,实现类对象>），进而走某个实现类的方法。当不同实现类中的存在通用逻辑，则可以将这些通用逻辑抽取出来做成方法放到一个抽象类中，不同实现类存在个性化逻辑，则各自实现类去重写service接口中的方法（多个实现类继承抽象类，抽象类实现service接口），这时可以引入门面模式。

（2）应用

- JCL和slf4j 都可以理解为是门面模式的应用。所有的其它日志实现JUL（java util logging）、logback、log4j、log4j2都可以通过JCL（Jakarta Commons Logging）或者slf4j（Simple Logging Facade for Java）来做门面适配。

（3）使用门面模式前的案例

- 需要画圆就要先实例化圆，画长方形就需要先考虑实例化一个长方形对象，然后再调用相应的 draw() 方法。

定义接口

```
public interface Shape {
   void draw();
}
```

定义几个实现类：

```
public class Circle implements Shape {
    @Override
    public void draw() {
       System.out.println("Circle::draw()");
    }
}
public class Rectangle implements Shape {
    @Override
    public void draw() {
       System.out.println("Rectangle::draw()");
    }
}
```

客户端调用：

```
public static void main(String[] args) {
    // 画一个圆形
     Shape circle = new Circle();
     circle.draw();
     // 画一个长方形
     Shape rectangle = new Rectangle();
     rectangle.draw();
}
```

（4）使用门面模式的案例

- 基于上面的接口和实现类，再增加一个门面类

```
//定义门面类
public class ShapeMaker {
  private Shape circle;
  private Shape rectangle;

  public ShapeMaker() {
    circle = new Circle();
    rectangle = new Rectangle();
  }

  //下面定义一堆方法，具体应该调用什么方法，由这个门面类来决定即可。（是不是很像适配器模式，这个门面类像是一个适配类，适配圆和长方形，调用适配类的画圆或长方形方法，方法内部实际是调用圆或长方形的实例对象的方法，只是客户端调用时直接调门面类的画圆或画长方形的方法即可，不用再考虑先实例化哪种对象）
  public void drawCircle(){
    circle.draw();
  }
  public void drawRectangle(){
    rectangle.draw();
  }
}
```

```
//客户端调用：
public static void main(String[] args) {
        ShapeMaker shapeMaker = new ShapeMaker();
        // 客户端调用现在更加清晰了
        shapeMaker.drawCircle();
        shapeMaker.drawRectangle();
}
```

（5）项目实战

- service接口

```
interface HealthManageService {
    /**
     * 保存通用健康方案
     */
    fun saveHealthManage(xxx): healthManage:
HsHealthSchemeManagementInfo

    /**
     * 保存通用计划
     */
    fun saveCommentHealthPlan(healthManage:
HsHealthSchemeManagementInfo)
    /**
     * 保存个性化健康计划
     */
    fun saveHealthPlan(healthManage:
HsHealthSchemeManagementInfo): List<HsHsmHealthPlan>
}
```

- service抽象类实现service接口，只重写通用的方法

```
abstract class AbstractHealthManageService(
    private val healthSchemeManageService:
HealthSchemeManageService,
    private val medicationRemindClient: MedicationRemindClient
) : HealthManageService {
        //保存通用健康方案
    override fun saveHealthManage(xxx):
    HsHealthSchemeManagementInfo{
        //编写逻辑
     }

    //保存通用计划
    override fun saveCommentHealthPlan(healthManage:
    HsHealthSchemeManagementInfo) {
        //编写逻辑
    }
}
```

- service实现类，继承了AbstractHealthManageService抽象类

```
@Service
class HealthManageHypertensionServiceImpl :
AbstractHealthManageService{
        //保存个性化健康计划
        override fun saveHealthPlan(
                healthManage: HsHealthSchemeManagementInfo
            ): List<HsHsmHealthPlan> {
              //编写个性化逻辑
            }
}

@Service
class HealthManageDiabetesServiceImpl:
AbstractHealthManageService{
   //保存个性化健康计划
   override fun saveHealthPlan(
   healthManage: HsHealthSchemeManagementInfo
   ): List<HsHsmHealthPlan> {
      //编写个性化逻辑
   }
}
```

- 创建门面类

```
@Service
class HealthManageFacadeServiceImpl(
        //1、依赖注入
        val healthManageServiceList: List<HealthManageService>,
) : HealthManageFacadeService{
        //2、静态块：健康方案类型映射实现类
```

```
//2、静态块：健康方案类型映射实现类
companion object{
    val SERVICE_MAP = mapOf(
        HealthManageType.HYPERTENSION to
        HealthManageHypertensionServiceImpl::class,
        HealthManageType.DIABETES to
        HealthManageDiabetesServiceImpl::class
    )
}

//3、整个保存流程的入口（保存通用健康方案，保存通用计划，保存个性化健康计划）
override fun save(saveHealthManageParam:
SaveHealthManageParam):
HsHealthSchemeManagementInfo? {
    //3-1、准备健康方案类型（一般这里是根据save方法的参数计算出来healthManageType是哪种方案类型，进而获取到实现类）
      val healthManageService: HealthManageService
        = healthManageServiceList.firstOrNull { it::class ==
        SERVICE_MAP[healthManageType] }
    //3-2、保存通用健康方案，这里会调用抽象类AbstractHealthManageService中的方法
    val healthManage: HsHealthSchemeManagementInfo =
    healthManageService.saveHealthManage(xxx)
    //3-3、保存通用健康计划，这里会调用抽象类AbstractHealthManageService中的方法
    healthManageService.saveCommentHealthPlan(health
    Manage)
    //3-4、保存个性化健康计划，这里会调用实现类中的方法
     healthManageService.saveHealthPlan(healthManage)
}
```

```
}
```