状态模式

2020年2月24日 9:38

1. 概念

对有状态的对象，把复杂的“判断逻辑”提取到不同的状态对象中，允许状态对象在其内部状态发生改变时改变其行为。

2. UML结构图

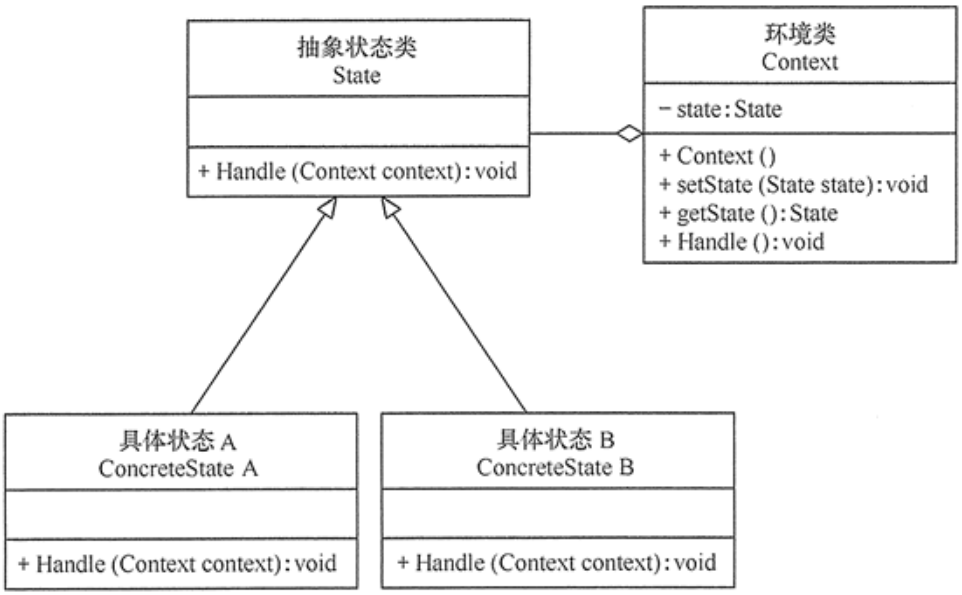

图2.1 状态模式UML结构图

1. 环境（Context）角色：也称为上下文，它定义了客户感兴趣的接口，维护一个当前状态，并将与状态相关的操作委托给当前状态对象来处理。
2. 抽象状态（State）角色：定义一个接口，用以封装环境对象中的特定状态所对应的行为。
3. 具体状态（Concrete State）角色：实现抽象状态所对应的行为。

3. 状态模式示例

在观看视频时，存在播放，暂停，加速和停止状态，利用状态模式把上述四个状态和对应的行为进行抽取到特定状态类。

1. 抽象状态类

```
public abstract class CourseVideoState {
    protected CourseVideoContext courseVideoContext;

    public void setCourseVideoContext(CourseVideoContext courseVideoContext) {
        this.courseVideoContext = courseVideoContext;
    }

    public abstract void play();
    public abstract void speed();
    public abstract void pause();
    public abstract void stop();
}
```

2. 具体状态类

```
public class PauseState extends CourseVideoState{
    @Override
    public void play() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.PLAY_STATE);
    }

    @Override
```

```
    public void speed() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.SPEED_STATE);
    }

    @Override
    public void pause() {
        System.out.println("播放暂停中");
    }

    @Override
    public void stop() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.STOP_STATE);
    }
}
```

```
public class PlayState extends CourseVideoState {
    @Override
    public void play() {
        System.out.println("播放视频中");
    }

    @Override
    public void speed() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.SPEED_STATE);
    }

    @Override
    public void pause() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.PAUSE_STATE);
    }

    @Override
    public void stop() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.STOP_STATE);
    }
}
```

```
public class SpeedState extends CourseVideoState{
    @Override
    public void play() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.PLAY_STATE);
    }

    @Override
    public void speed() {
        System.out.println("播放加速中");
    }

    @Override
    public void pause() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.PAUSE_STATE);
    }

    @Override
    public void stop() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.STOP_STATE);
    }
}
```

```
public class StopState extends CourseVideoState{
    @Override
    public void play() {
        super.courseVideoContext.setCourseVideoState(CourseVideoContext.PLAY_STATE);
    }

    @Override
    public void speed() {
        System.out.println("ERROR 停止状态不能快进");
    }

    @Override
    public void pause() {
        System.out.println("ERROR 停止状态不能暂停");
    }

    @Override
    public void stop() {
        System.out.println("播放停止中");
    }
}
```

3. 状态上下文

```
public class CourseVideoContext {
    private CourseVideoState courseVideoState;
    public final static PlayState PLAY_STATE = new PlayState();
    public final static StopState STOP_STATE = new StopState();
    public final static PauseState PAUSE_STATE = new PauseState();
    public final static SpeedState SPEED_STATE = new SpeedState();

    public CourseVideoState getCourseVideoState() {
```

```
        return courseVideoState;
    }

    public void setCourseVideoState(CourseVideoState courseVideoState) {
        this.courseVideoState = courseVideoState;
        // 设置视频状态上下文
        this.courseVideoState.setCourseVideoContext(this);
    }

    public void play(){
        this.courseVideoState.play();
    }
    public void stop(){
        this.courseVideoState.stop();
    }
    public void pause(){
        this.courseVideoState.pause();
    }
    public void speed(){
        this.courseVideoState.speed();
    }
}
```

4. 客户端

```
@Test
    public void stateTest(){
        //创建一个上下文对象
        CourseVideoContext courseVideoContext = new CourseVideoContext();
        courseVideoContext.setCourseVideoState(new PlayState());
        System.out.println("当前状态: " +
courseVideoContext.getCourseVideoState().getClass().getSimpleName());

        courseVideoContext.pause();
        System.out.println("当前状态: " +
courseVideoContext.getCourseVideoState().getClass().getSimpleName());

        courseVideoContext.speed();
        System.out.println("当前状态: " +
courseVideoContext.getCourseVideoState().getClass().getSimpleName());

        courseVideoContext.stop();
        System.out.println("当前状态: " +
courseVideoContext.getCourseVideoState().getClass().getSimpleName());
    }
```

5. 结果

```
当前状态: PlayState
当前状态: PauseState
当前状态: SpeedState
当前状态: StopState
```

4. 优/缺点

1. 优点

- 状态模式将与特定状态相关的行为局部化到一个状态中，并且将不同状态的行为分割开来，满足“单一职责原则”。
- 减少对象间的相互依赖。将不同的状态引入独立的对象中会使得状态转换变得更加明确，且减少对象间的相互依赖。
- 有利于程序的扩展。通过定义新的子类很容易地增加新的状态和转换。

2. 缺点

- 状态多的业务场景导致类数目增加，系统复杂。

5. 适用场景

一个对象存在多个状态（不同状态下行为不同），且状态可以相互转换。

6. 扩展

在有些情况下，可能有多个环境对象需要共享一组状态，这时需要引入享元模式，将这些具体

状态对象放在集合中供程序共享，其结构图如图所示。

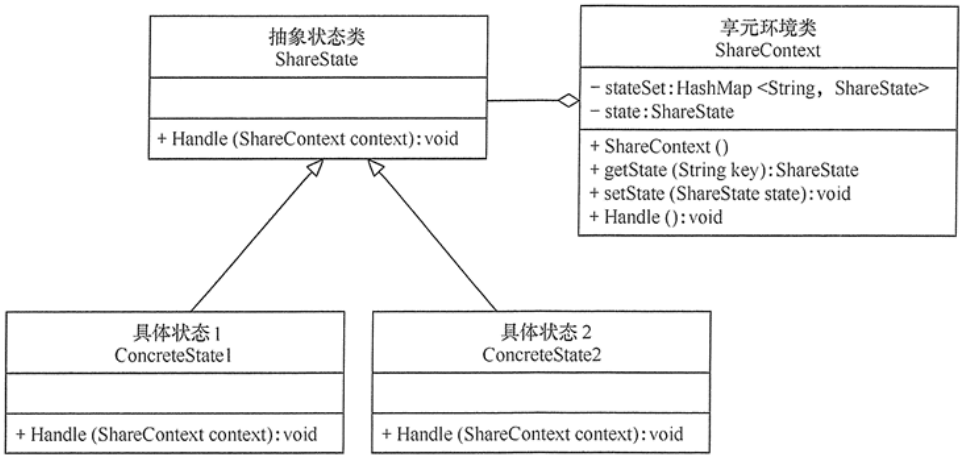

图6.1 状态模式与享元模式组合

7. 参考资料

[1] http://c.biancheng.net/view/1388.html