1 状态模式 (State)

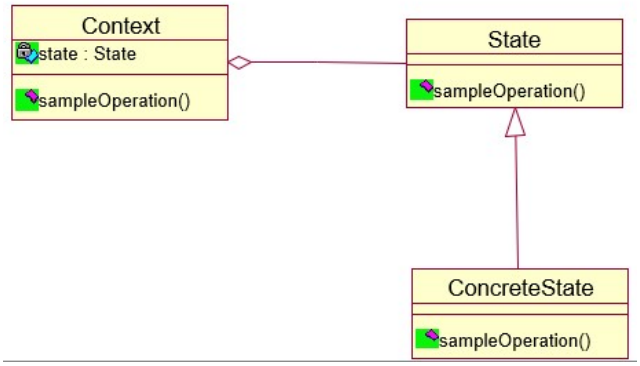
状态模式，又称状态对象模式。状态模式是对象的行为模式 [GOF95] 状态模式允许一个对象在其内部状态改变的时候改变其行为。这个对象看上去就像是改变了它所属的类一样。

很多情况下，一个对象的行为取决于一个或多个动态变化的属性，这样的属性叫做状态，这样的对象叫做有状态的对象。这样的对象状态通常是从一系列预定义的值中取出的。

```
1  class Elevator {
2    int level; //-2表示最底层
3    int state; //1表示正常，2表示故障,3表示检修
4    void onDownButtonPressed() {
5      if(level===-2 || state!=1) {
6        //do nothing
7      } else {
8        //电梯向下
9      }
10   }
11 }
```

1.1 状态模式的结构

状态模式把对象的行为包装在不同的状态对象里，状态对象都属于一个抽象状态类的一个子类。状态模式的意图是让一个对象在其内部状态改变的时候，其行为也随之改变。状态模式的示意性类图如下：



模式所涉及的角色有：

- 抽象状态角色：定义一个接口，用以封装环境（Context）对象的一个特定的状态所对应的行为。
- 具体状态角色：每一个具体状态类都实现了环境（Context）的一个状态所对应的行为。
- 环境角色：定义行为将随状态变化的对象，并持有一个具体状态对象的引用。

下面是环境角色 Context 的源代码，可以看出，环境类持有一个 State 对象，并把所有的行为委派给此对象。

```
public class Context {
   private State state;
   public void sampleOperation() { state.sampleOperation(); }
   public void setState(State state) { this.state=state; }
}
```

```
public interface State { void sampleOperation(); }
```

```
public class ConcreteState implements State {
  public void sampleOperation() { }
}
```

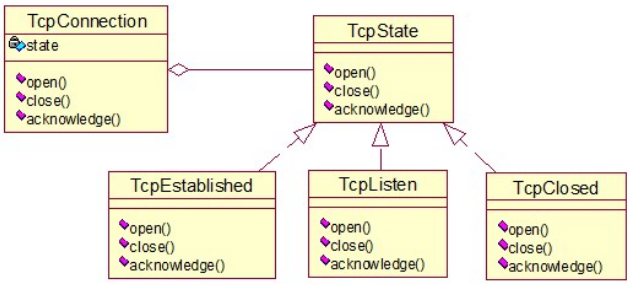
1.2 使用场景

在以下各种情况下可以使用状态模式：

1. 一个对象的行为依赖于它所处的状态，对象的行为必须随着其状态的改变而改变。
2. 对象在某个方法里依赖于一重或多重的条件转移语句，其中有大量的代码。状态模式把条件转移语句的每一个分支都包装到一个单独的类里。这使得这些条件转移分支能够以类的方式独立存在和演化。

1.3 例:TCP

考虑由 TcpConnection 代表一个 TCP/IP 网络的连接。一个 TcpConnection 对象可以具有以下状态之一：Listening（监听）、Established（已建立连接）、Closed（关闭）。当 TcpConnection 对象接到其它对象的请求时，会根据其状态不同而给出不同的回应。TCP 系统的 UML 类图如下所示：



在此，使用状态模式的关键是引进了抽象类或接口 TcpState 来代表网络的连接状态。它的子类实现了由状态决定的行为。TcpConnection 类持有一个状态对象，TcpState 子类的实例，来表示 TCP 连接的现在状态。TcpConnection 类把所有与状态有关的请求都委派给它的状态

对象。TcpConnection 使用它的 TcpState 的子类实例来执行特定的连接状态所对应的操作。当连接的状态改变时，TcpConnection 对象就改变它所用的状态对象。例如，当网络连接从“已建立”改为“关闭”时，TcpConnection 对象会把它的状态对象从 TcpEstablished 的实例改为 TcpClosed 的实例。

系统有如下的角色：

- 环境角色：TcpConnection。此类定义了客户端感兴趣的接口，并持有有关具体状态类的实例，以定义它当前所处的状态。
- 状态角色：TcpState。此类定义了把依赖于一个特定的状态所对应的行为包装起来所需要的接口。
- 具体状态角色：TcpEstablished、TcpListen、TcpClosed。实现了环境类的状态所对应的行为。

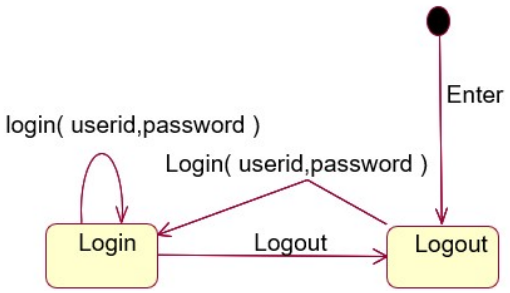
```
1 public class TcpConnecion {
2   private TcpState state;
3   public void open() { state.open(); }
4   public void setState(TcpState state) { this.state=state; }
5   public void close() { state.close(); }
6   public void acknowledge() { state.acknowledge(); }
7 }
```

```
1 public interface TcpState {
2    void open();
3    void close();
4    void acknowledge();
5 }
```

```
1 public class TcpEstablished implements TcpState {
2   public void open() { }
3   public void close() { }
4   public void acknowledge() { }
5 }
```

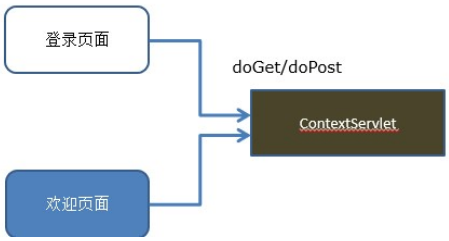
1.4 例: 用户登录子系统

用户进入系统时首先遇到登录页面，输入用户名和密码，单击〔log on〕键，系统会向数据库查询。登录成功后，就会见到欢迎页面。登录失败，仍然要面对登录页面。下图描述了用户的状态变化情况。

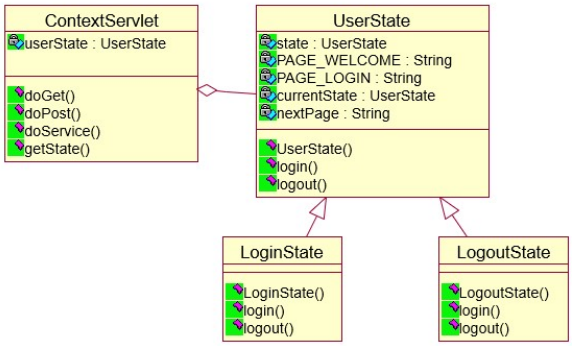


从上面的状态图可以看出，用户进入系统时，处于 Logout 状态，用户输入正确的用户名和密码可以过渡到 Login 状态。当用户处于 Login 状态时，可以发出 Logout 命令，使系统过渡到 Logout 状态。用户在 Login 状态时，重新发出 Login 的命令不会改变状态。

不同的状态会产生不同的网页，对应于 Login 状态的是欢迎页面，对应于 Logout 状态的是登录页面。登录页面提交后的 Servlet 是 ContextServlet。这个 servlet 在 doGet 或 doPost 方法中接收用户名和密码，并且创立对应的状态对象。



登录子系统的静态结构图如下:



登录页面的源代码如下:

```
<HTML>
  <BODY>
     <FORM action= “/Servlet/com.javapatterns.state.login.ContextServlet"
    method=get>
   User ID<INPUT TYPE=text name=userid value= “jeff"><br>
   Password < INPUT TYPE=password name=password value= “pass"><br>
```

```
     <input type=submit name= “btnAction" value= “Log On">
    </FORM>
  </BODY>
</HTML>
```

登录后的页面（欢迎页面）。其源代码如下：

```
<HTML>
  <BODY>
    <FORM action= “/Servlet/com.javapatterns.state.login.ContextServlet"
   method=get>
   Welcome to website!<br>
     <input type=submit name= “btnAction" value= “Log Out">
    </form>
  </body>
</html>
```

环境类 ContextServlet 的源代码如下：

```
package com.javapatterns.state.login;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.Servlet.ServletRequest;
import javax.Servlet.http.*;
public class ContextServlet extends HttpServlet {
  private UserState userState = new LogoutState();
  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {
    doService(request,response);
  }
  public void doPost(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException,IOException {
    doService(request,response);
  }
  public void doService(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException,IOException {
    String userId=request.getParameter("userid");
    String password=request.getParameter("password");
    String btnAction=request.getParameter("btnAction");
```

```
24      if (btnAction.equalsIgnoreCase("log on")) {
25        this.getState().login(userId,password);
26      } else if (btnAction.equalsIgnoreCase("log out")) {
27        this.getState().logout();
28      }
29      response.sendRedirect(this.getCurrentState().getNextPage());
30    }
31    /*状态的取值方法*/
32    public UserState getState() { return userState.getCurrentState(); }
33  }
```

环境类首先判断客户端的请求是登录还是退出。当接到登录请求时,环境类调用状态类的 login() 方法。当接到退出请求时，环境类调用状态类的 logout() 方法。

抽象类 UserState 代码清单如下:

```
1   public abstract class UserState {
2     private UserState state;
3     private String nextPage;
4     protected static String PAGE_WELCOME=
5     "/javapatterns/state/welcome.html";
6     protected static String PAGE_LOGIN=
7     "/javapatterns/state/login.html";
8     public UserState() { this.nextPage= PAGE_LOGIN; }
9     /*行为方法*/
10    public abstract boolean login(String userId,String password);
11    public abstract void logout();
12    /*设置状态*/
13    public void setCurrentState(UserState state) { this.state=state; }
14    /*获取当前状态*/
15    public UserState getCurrentState() {
16      if (this.state==null) {
17        this.state=new LogoutState();
18      }
19      return this.state;
20    }
21    public String getNextPage() { return nextPage; }
22    public void setNextPage(String nextPage) { this.nextPage=nextPage; }
23  }
```

子系统的具体类 LoginState 包装了系统在登录成功后的 Login 状态。其源代码如下:

```
1   public class LoginState extends UserState {
```

```
  public LoginState() { }
  public boolean login(String userId, String password) {
    setNextPage(UserState.PAGE_WELCOME);
    setCurrentState(new LoginState());
    return true;
  }
  public void logout() {
    setNextPage(UserState.PAGE_LOGIN);
    setCurrentState(new LogoutState());
  }
}
```

子系统的具体类 LogoutState 包装了系统在登录失败后的 Logout 状态。其源代码如下：

```
public class LogoutState extends UserState {
  public LogoutState() { }
  public boolean login(String userId, String password) {
    StringBuffer sql=new StringBuffer(50);
    sql.append("select count(*) from user_info where user_id='");
    sql.append(userId);
    sql.append("' and password='");
    sql.append(password);
    sql.append("'");
    int counting=DBManager.query(sql.toString());
    if (counting > 0) {
      this.setNextPage(UserState.PAGE_WELCOME);
      this.setCurrentState(new LoginState());
      return true;
    } else {
      this. setCurrentState(new LogoutState());
      setNextPage(UserState.PAGE_LOGIN);
      return false;
    }
  }
  public void logout() {
    this.setCurrentState(new LogoutState());
    setNextPage(UserState.PAGE_LOGIN);
  }
}
```