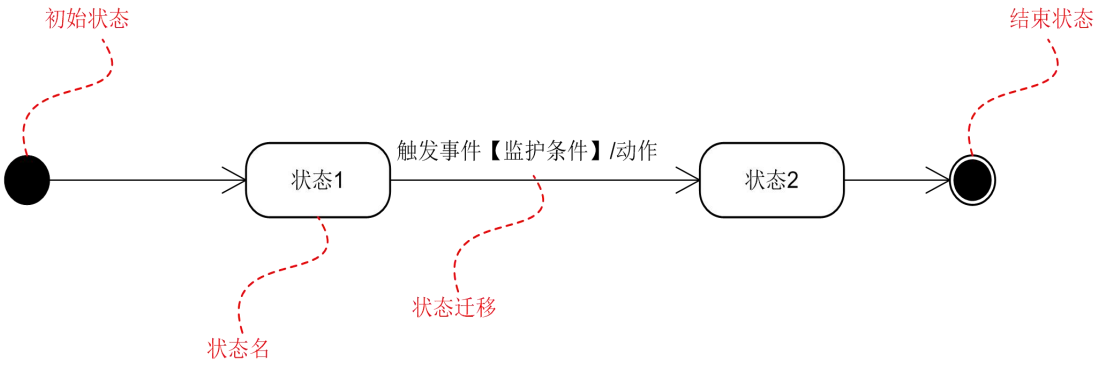
第13章 状态机图
(Statemachine diagram)

学习目标

- 学习完本章节，要求达到以下状态：
  - 能够说明状态机图的表示方法和使用方法
  - 能够读懂状态机图并理解其中的含义
  - 能够用状态机图来表示一个对象在生命周期内的状态变化

状态机图的概要

- 对象除了具有属性和操作之外，还具有状态。
- 状态机图用来描述一个对象从生成到消失整个生命周期内所经历的状态变化。
  - 表示一个对象对于来自外部的事件如何做出反应的情况。
  - 不是所有的对象都需要画状态机图，只有当生命周期内有复杂状态变化的对象，或者需要把握其状态迁移变化的对象才需要画状态机图。
- 状态机图的模型元素：
  - 状态
  - 状态迁移



Statechart Diagram(状态图)

- 定义：A **statechart diagram** shows a state machine, emphasizing the flow of control from state to state.
- **state machine(状态机)**：A state machine models the lifetime of a single object, whether it is an instance of a class, a use case, or even an entire system.

状态图的用途

- UML的状态图主要用于建立类的一个对象在其生存期间的动态行为，表现一个对象所经历的状态序列，引起状态转移的**事件(event)**，以及因状态转移而伴随的**动作(action)**。

- 状态图的目的是为具有以下两个特性的属性建模：
  - 属性拥有很少的值；
  - 属性在这些值之间的转移有一定的限制。

例：类SellableItem有两个属性salePrice:Money和status:Inspection。

其中salePrice的取值范围为正实数，status的取值为received, inInspection, accepted, rejected等，则应根据属性status建立状态图。

说明：

- **状态图**是UML 1.x中对系统的动态行为建模的五个图之一。
- 状态图描述了一个特定对象的所有可能状态以及由于各种事件的发生而引起的状态之间的转移。
- 状态图是显示状态机的一种方式，另一种显示状态机的方式是**活动图**。
- Rose中，状态图不生成代码，但状态图在检查，调试和描述类的动态行为时非常有用。

- **状态图**适合于描述跨越多个用例的单个对象的行为，而不适合描述多个对象之间的行为协作，因此，常常将状态图与其它技术组合使用。例如，**交互图**适合于描述单个用例中的多个对象的行为，**活动图**适合于描述多个对象和多个用例的活动的总次序。

状态图中的基本概念

- State (状态)
- Transition (转移)
- Event (事件)
- Action (动作)

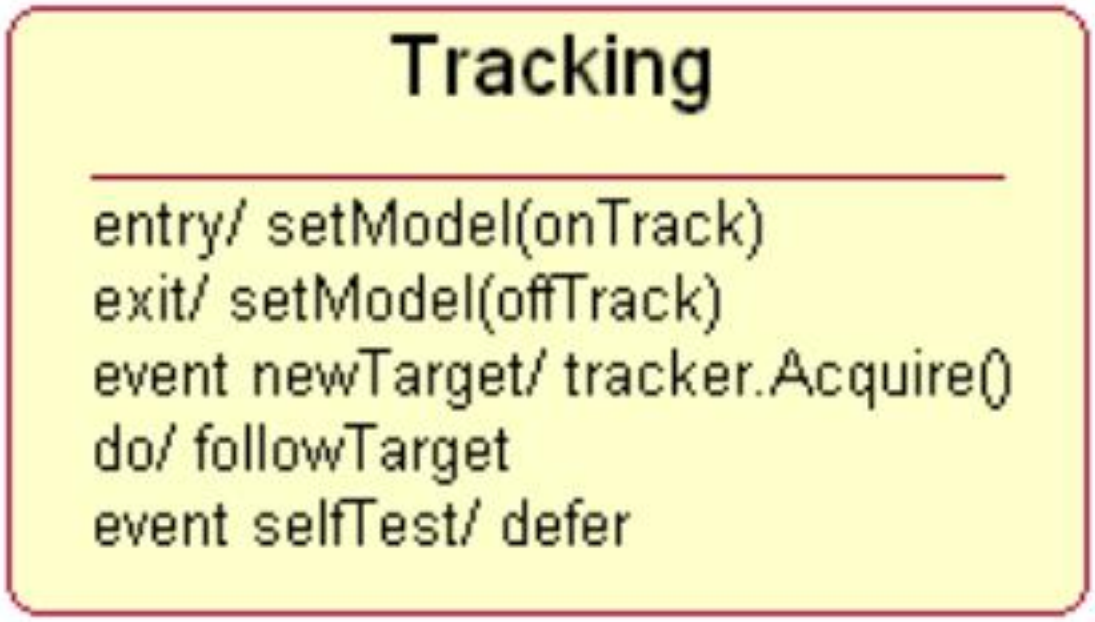
状态(State)

- **定义：**A **state** is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.
- 一个状态是指在对象的生命期中的一个条件或状况，在此期间对象将满足某些条件、执行某些活动或等待某些事件。

说明：

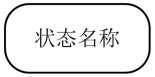
- 所有对象都具有状态，状态是对象执行了一系列活动的结果。当某个事件发生后，对象的状态将发生变化。
- 状态图中定义的状态有：**初态**、**终态**、**中间状态**、**组合状态**、**历史状态**等。
- 一个状态图只能有一个初态，而终态可以有多个，也可以没有终态。

- 一个状态有以下几个部分：
  - **name**
  - **entry/exit action**
  - **internal transition**
  - **substate**
  - **deferred event**

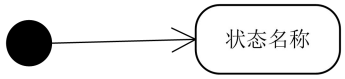


状态

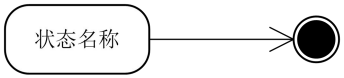
- 在状态机图里，对象的状态有三种：
  - 状态
    - 表示一个对象从生成开始，到消亡结束的过程中，在某一时刻所处的状态。用圆角矩形表示：



  - 初始状态
    - 表示一个对象生命周期的最开始状态，或者一个组合状态的子状态的开始。用实心圆表示：



  - 终了状态
    - 表示一个对象生命周期的终了状态，或者组合状态中某一个子状态的结束。用实心圆外加一空心圆表示：



Transition(转移)

- A **transition** is a relationship between two states indicating that an object in the first state will perform specified **action**s and enter the second state when a specified **event** occurs and specified **guard condition**s are satisfied.
- 一个**转移**是两个状态之间的一种关系，表示对象将在第一个状态中执行一定的**动作**，并在某个特定**事件**发生而且某个特定的**警戒条件**满足时进入第二个状态。

状态迁移

◆ 表示对象在某事件的触发下从一个状态转移到另外一个状态的过程。

■ 用带箭头的直线来连接迁移变化的两个状态，箭头指向迁移后的新状态。

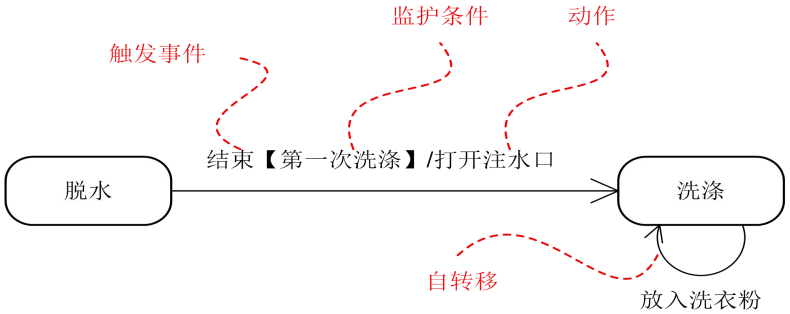
■ 在状态迁移线上可以按照如下格式指明导致状态迁移的触发事件，状态迁移的监护条件，状态迁移时要执行的动作等等要素。

触发事件【监护条件】/ 动作

- “触发事件”表示的是引起该状态迁移的原因。
- “监护条件”表示的是接收到触发事件后，要进行状态迁移前必须要满足的条件。
  表示方法:【监护条件】
- “动作”表示的是，接收到触发事件，并且满足指定的监护条件，进行状态迁移时所执行的动作。
  表示方法: /动作

■ 重新返回到自身状态的状态迁移称为自转移。

例:看一个关于洗衣机的“洗涤”和“脱水”之间的状态迁移:

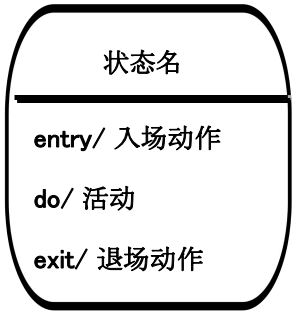


洗衣机在“洗涤”状态时，在“放入洗衣粉”事件的触发下，还是回到自身“洗涤”状态。

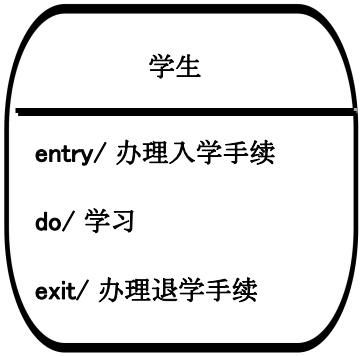
洗衣机在“脱水”状态时，在“脱水结束”事件的触发下，当条件--现在进行的是“第一次洗涤”满足时，洗衣机执行“打开注水口”动作，然后状态转移到“洗涤”状态。

状态区域的划分

- 状态机图中的状态可以划分为两个区域：
  - 状态名区域
    - 指定该状态的名称。
  - 内部活动区域
    - 入场动作
      指定进入该状态时要执行的动作。表示方法为： entry/ 动作名
    - 退场动作
      指定退出该状态时要执行的动作。表示方法为：exit/ 动作名
    - 活动
      指定在进入该状态之后，退出该状态之前，在一段时间内要持续的动作。表示方法为：do/ 活动名

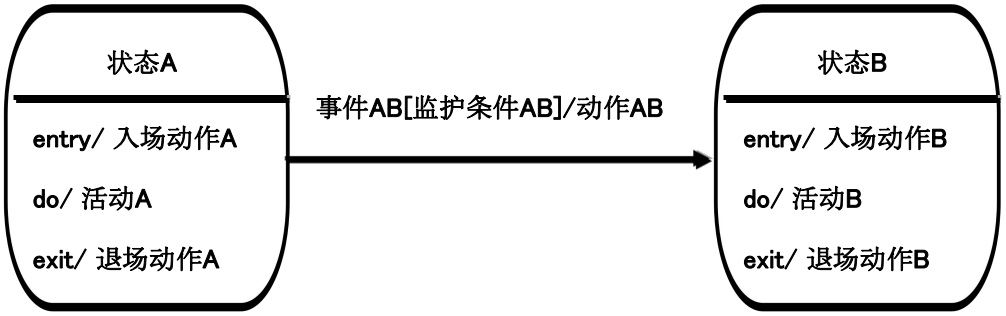


以“学生”状态为例，来看看入场动作，出场动作，以及活动的定义。



事件，动作，活动的发生顺序

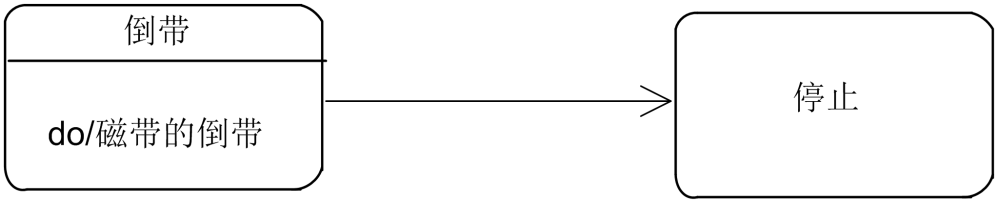
- 状态迁移变过过程中，各个事件，动作，活动等发生的先后顺序如下：



入场动作A -> 活动A -> 事件AB -> 监护条件AB -> 退场动作A -> 动作AB -> 入场动作B -> 活动B

结束迁移(无触发迁移)

◆ 是指没有事件的触发，随着某个活动或者动作的结束而进行的状态迁移：



在“倒带”状态启动了“磁带的倒带”活动，并且在该活动结束后进行状态迁移，迁移到“停止”状态。

composite state(组合状态)和substate(子状态)

- A **composite state** is a state that consists of either concurrent substates or sequential substates.
- A **substate** is a state that is nested inside another one.
- **子状态**是嵌套在另一个状态中的状态。
- 一个含有子状态的状态被称作**组合状态**。

说明：

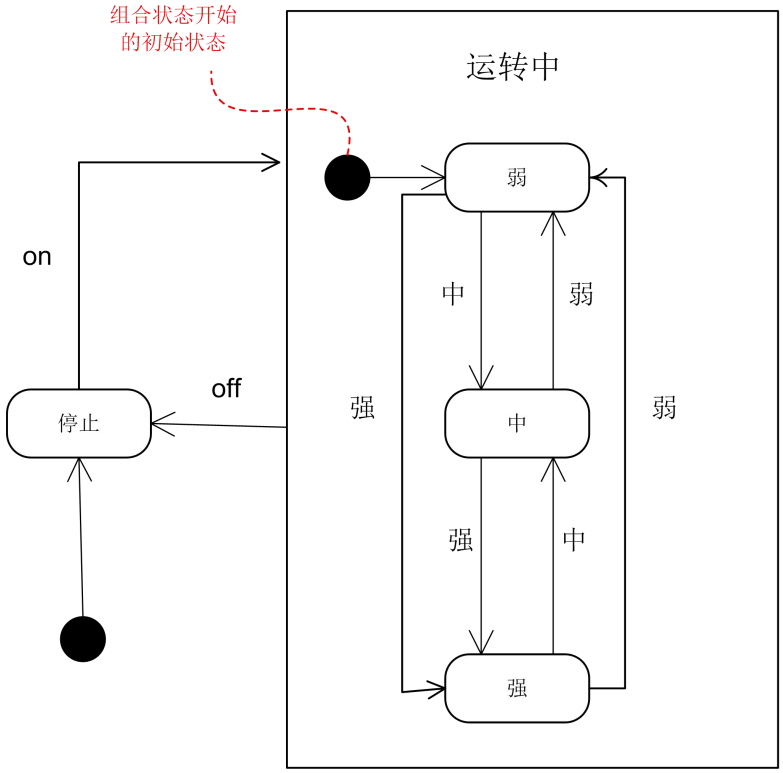
- 组合状态中也有初始状态和终结状态。
- 从源状态可以转移到组合状态本身，也可以直接转移到组合状态中的子状态。
- 子状态之间可分为**“or”关系**和**“and”关系**两种。“or”关系说明在某一时刻仅可到达一个子状态，“and”关系说明组合状态中在某一时刻可同时到达多个子状态。

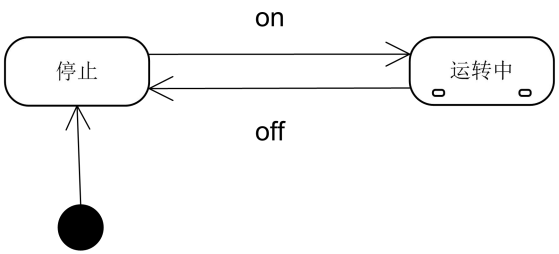
组合状态

◆ 内部还拥有子状态的状态称为组合状态。而且子状态可以分为若干层。

例:电风扇的的“运转中”状态里，还包含有“弱档运转”“中档运转”和“强档运转”三个状态。所以 “运转中”为组合状态。

完整的组合状态图如左下图所示;组合状态图也可省略表示子状态，如右下图所示。



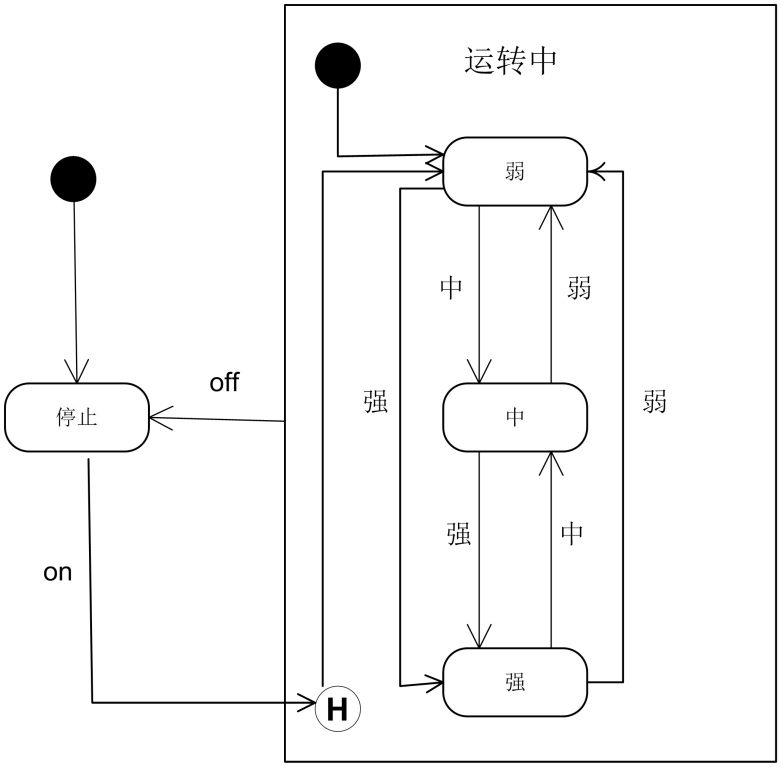


History State(历史状态)

- History State: A pseudostate whose activation restores the previously active state within a composite state.
- 使用历史状态，可以记住从组合状态中退出时所处的子状态，当再次进入组合状态时，可直接进入到这个子状态，而不是再次从组合状态的初态开始。

历史状态符

- 历史状态符为H符号外加一圆圈来表示。
  - 历史状态符所连接的子状态还代表第一次进入组合状态时所处的子状态。
  - 在组合状态中如果不使用历史状态符，当离开组合状态后再次回到组合状态时，则由组合状态的初始状态开始执行。



- 当系统处于“运转中”的“中”状态时，按下off后再按on时：

  当没有历史状态符时的状态迁移过程为：

  “中” -> “停止” -> “弱”

  当有历史状态符时的状态迁移过程为：

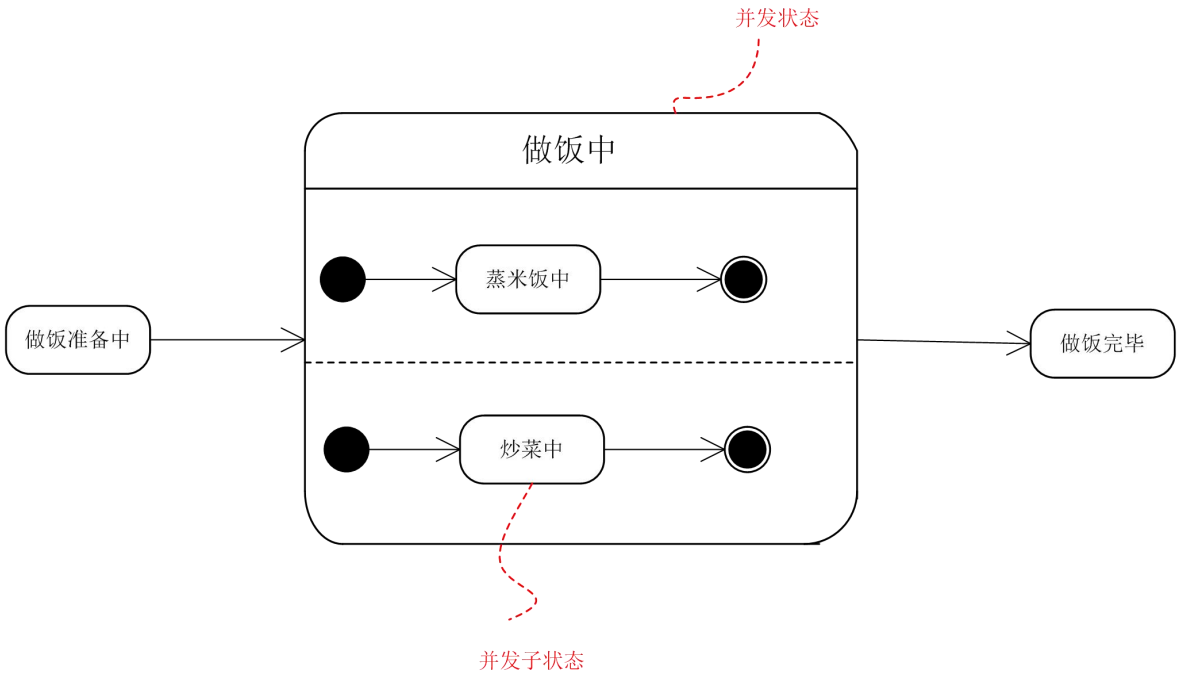
  “中” -> “停止” -> “中”

- 当系统第一次进入“运转中”状态时，是从“弱”状态开始的。

并发状态

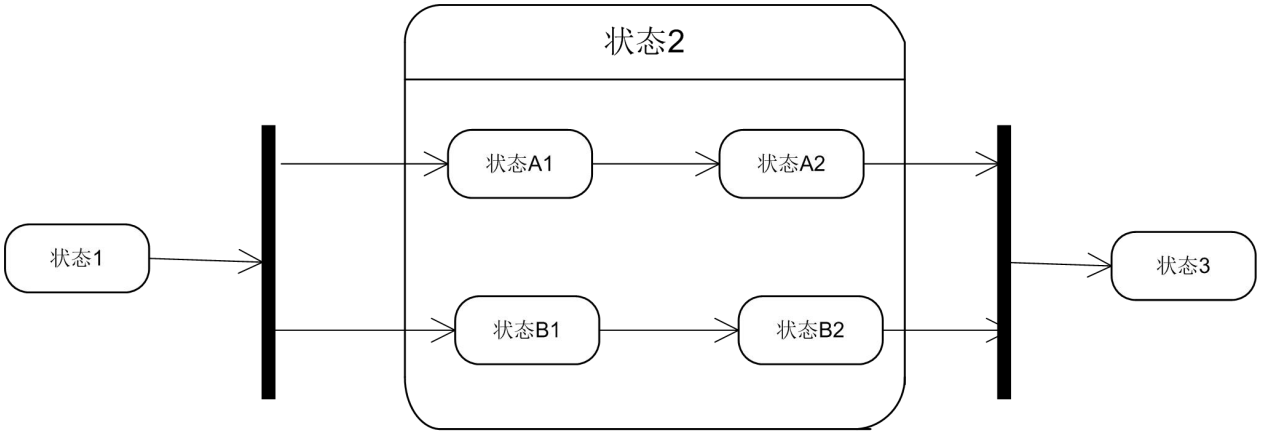
◆ 在一个组合状态中，同时有多个子状态存在的时候，该组合状态称为并发状态。

■并发状态中的子状态为并发子状态。



同期迁移

- 表示在组合状态中多个子状态的迁移同期并发进行。
  - 用同步条来表示。



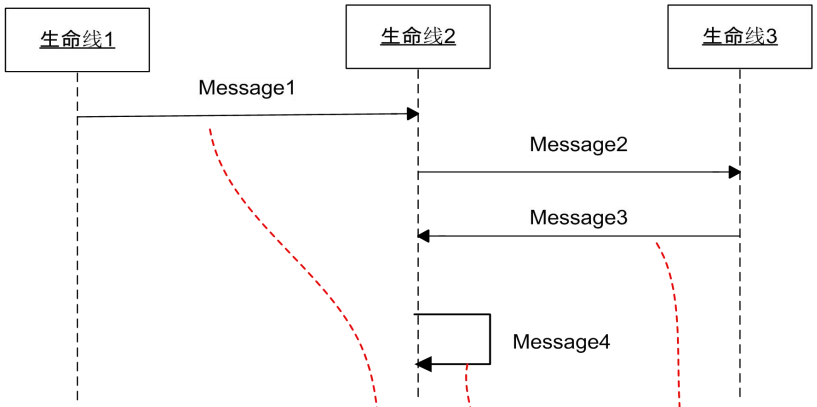
- 状态A1->状态A2的迁移，和状态B1->状态B2的迁移是同时进行的。

- 只有当状态A2和状态B2都结束后，才能进行到状态3的迁移。状态A2和状态B2的任意一方先结束的情况下，都要等待另外一方也结束之后，才能进行到状态3的迁移。

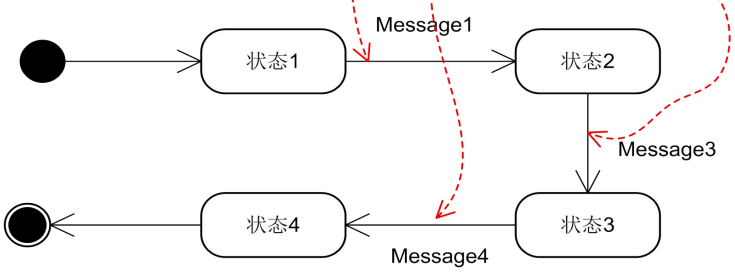
状态机图和交互图的一致性

- 状态机图和交互图之间具有一致性：
  - 交互图中参与交互的生命线实体对象和状态机图中的对象具有对应关系。
  - 交互图中调用生命线对象操作的消息，对应于该对象状态机图中的引起状态迁移的触发事件。

交互图：



生命线2对应的对象的状态机图：



例题:画状态图

◆ 考虑一个培训课程。

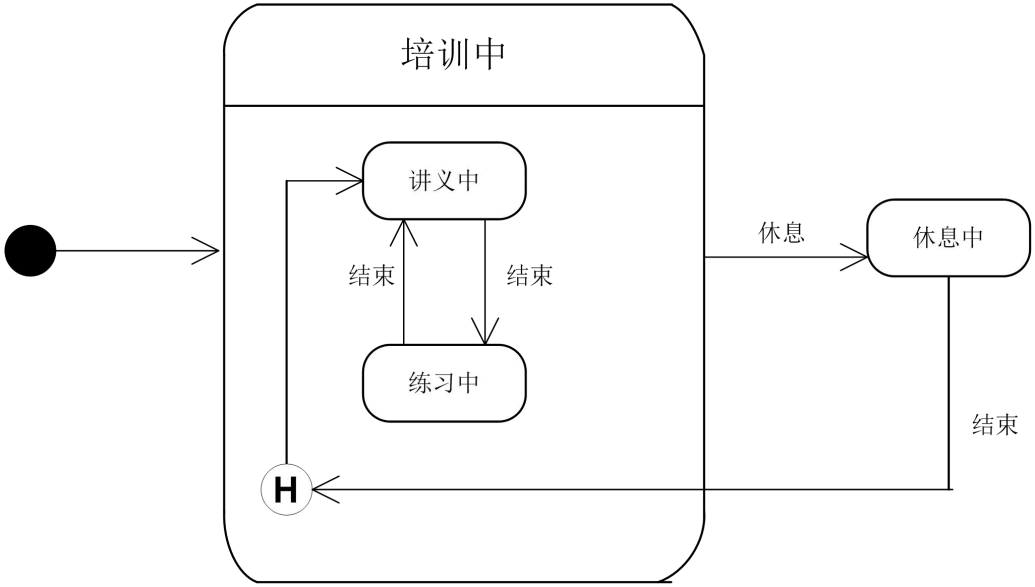
该培训课程由讲义和练习两部分组成，最初由讲义开始。

在讲义或者练习的过程中会插入休息。如果是在讲义过程中插入休息，那么休息完毕后再继续讲义。如果在练习过程中插入休息，那么休息之后再继续练习。

试用状态图来描述一下该培训课程的状态变化过程。

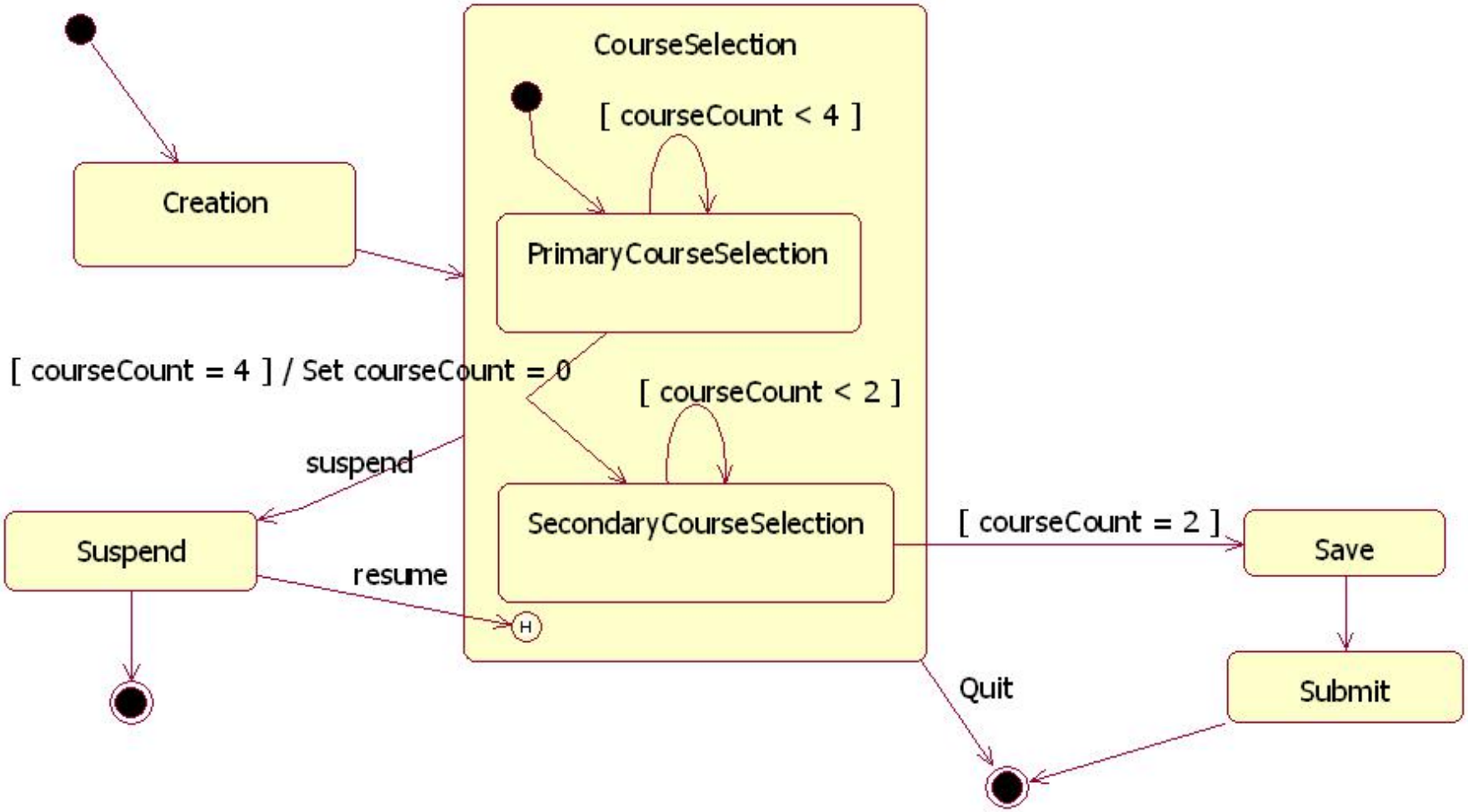
提示:首先抽取出来该培训课程会经历的几种状态(包括组合状态)。然后再考虑状态之间的迁移和触发事件之间的关系。

题解(参考):



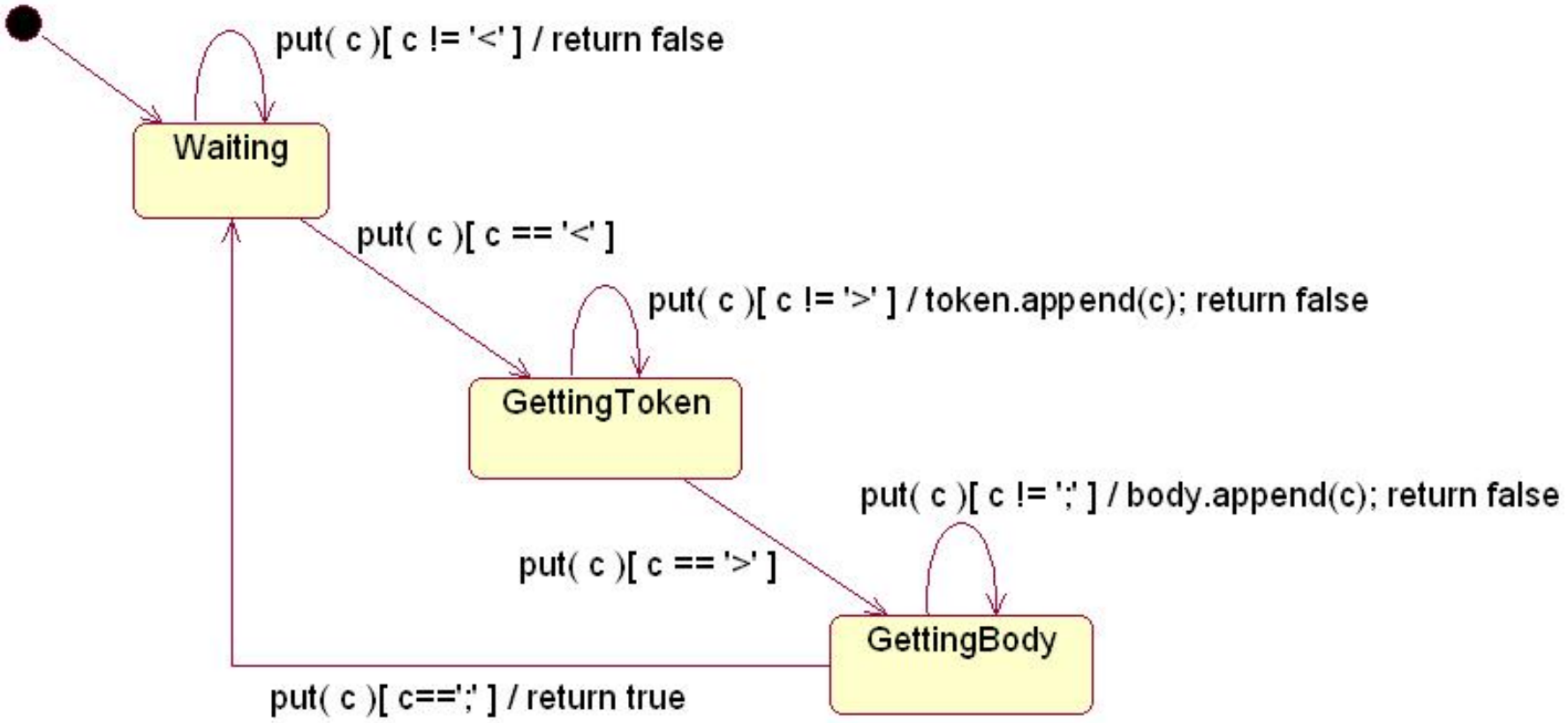
课程注册系统中的状态图例子

类型为RegistrationController的对象的状态图：



状态图的工具支持

- 正向工程：根据状态图生成代码。例：



- 所生成的代码示例：

```
class MessageParser {
public boolean put(char c) {
    switch (state) {
        case Waiting:
            if (c == '<') {
                state = GettingToken;
                token = new StringBuffer();
                body = new StringBuffer();
            }
            break;
        case GettingToken :
            if (c == '>')
                state = GettingBody;
            else
                token.append(c);
            break;
        case GettingBody :
            if (c == ';') {
                state = Waiting;
                return true;
            }
            else
                body.append(c);
        }
        return false;
    }
public StringBuffer getToken() {
    return token;
}
public StringBuffer getBody() {
    return body;
}
private final static int Waiting = 0;
private final static int GettingToken = 1;
private final static int GettingBody = 2;
private int state = Waiting;
private StringBuffer token, body;
}
```

状态图建模风格

- 建模风格1：把初态放置在左上角；把终态放置在右下角。
- 建模风格2：用过去式命名转移事件。
  - 这样做是为了反映转移是事件的结果这个事实。也就是说，事件出现在转移之前，因此引用事件时要用过去式。
  - 例如：用cancelled，closed，scheduled等命名事件。

- 建模风格3：警戒条件不要重叠。
  - 从一个状态出来的相似转移上的警戒条件相互之间必须一致。
  - 例如，像x < 0，x = 0和x > 0这样的警戒条件是一致的，而像x <= 0和 x >= 0这样的警戒条件是不一致的。
- 建模风格4：不要把警戒条件置于初始转移上。
  - 当警戒条件为假的时候，这个对象做什么？

总结

◆ 状态图描述的是一个对象在其生命周期内的状态变化过程。

■具有复杂生命周期的对象，或者需要把握其状态变化的对象，才需要画状态图。

◆ 状态图中表征触发状态转移的消息标签除了能指定事件名称外，还能指定状态迁移所必须满足的监护条件，以及状态迁移时系统要做的动作。

◆ 状态图中的状态可以划分为两个区域---状态名区域和内部活动区域。

■在内部活动区域，可以指定

进入该状态要执行的动作-入场动作；

退出该状态要执行的动作-出场动作；

从进入到退出一直要进行的动作-活动。

◆ 内部还含有状态的状态成为组合状态。组合状态内部含有的状态称为子状态。

■用历史符可以记忆上次离开组合状态时的子状态，以备之后再次返回组合状态时，能够从上次退出时的状态开始继续执行。

◆ 同时有多个子状态存在的组合状态成为并发状态。

◆ 当多个状态迁移具有同期同步关系时，称为同期转移。可用同步条来表示这种关系。

◆ 状态图和交互图之间具有一致性。交互图中的消息对应的是状态图中引起状态迁移的触发事件。

练习题

- 请参考练习题。