私塾在线《设计模式综合项目实战》

——跟着CC学设计系列精品教程

本阶段课程概览

- 设计并实现核心框架中的具体调用模块

一：具体调用模块的详细功能

二：具体调用模块的功能边界

三：具体调用模块对外的接口

四：具体调用模块的内部实现

包括：状态模式、模板方法模式、工厂方法模式、装饰者模式、观察者模式的综合应用

五：具体调用模块和其他模块的交互实现

主要是使用中介者模式

具体调用模块的详细功能

- 能够控制整个X-gen的调用过程，并能够灵活的扩展这个调用过程
- 调用theme提供的Action来具体实现每一个需要生成的功能
- 能够在每个Action执行前后，动态组合添加一些功能
- 能很灵活的通知多个输出实现，并能实现调用模块和输出模块的解耦

具体调用模块的功能边界

- 具体调用模块的功能边界

1：具体调用模块只是负责具体的generate调用过程

2：具体调用模块不关心generate的数据从何而来

3：具体调用模块不关心实际如何generate

4：具体调用模块不关心按照什么流程顺序来generate

5：具体调用模块也不关心每个步骤都需要完成些什么功能，那都不是固定的，完全可以通过配置或者是开发人员在外部theme中来定

具体调用模块的对外接口

- 对外程序接口设计

具体调用模块的内部实现

- 实现的起点

为了让大家更好的理解调用模块的内部实现架构，因此先以一个最简单的实现结构为起点，采用重构的方式，逐步把相关的设计模式应用进来，从简单到复杂，从而让大家更好的看到如何选择要使用的设计模式、如何实际应用设计模式以及如何让多种设计模式协同工作。

1：针对前面定义的API，提供一个最基本的实现。其实，真正调用的实现就相当于命令模式的receiver

加入状态模式

- 面临的问题

  来思考上面的实现，现在在executeGen方法里面只有两步实现，可是不排除今后有更多需要调用的功能，比如：在内容生成后执行事件处理等等。

  通常情况下，我们还希望保持DefaultGenInvocation的通用性，该怎么办呢？换句话说，executeGen方法里面的调用流程是可能变动的，或者是需要添加新的流程步骤，或者是需要改变调用的顺序等等。

  该怎么实现这样的功能呢？

- 用状态模式来解决

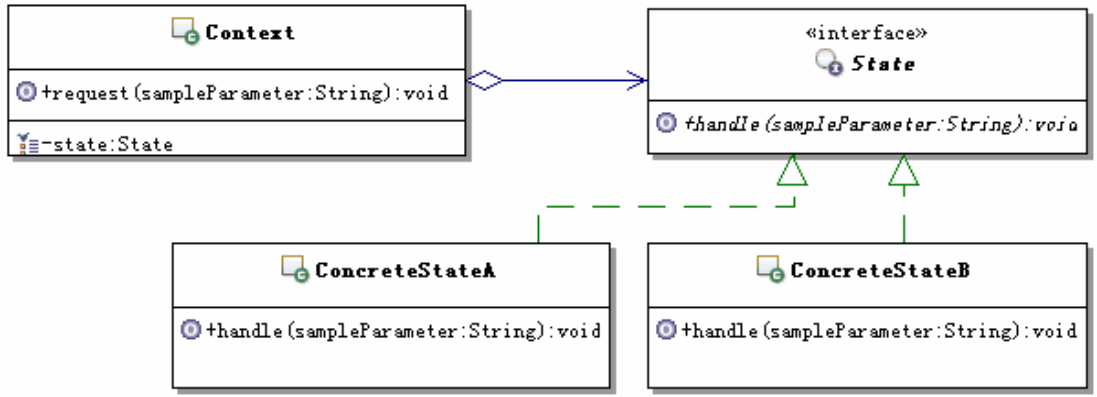
- 状态模式基础回顾

初识状态模式

- 定义

  允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它的类。

- 结构和说明



Context：环境，也称上下文，通常用来定义客户感兴趣的接口，同时维护一个来具体处理当前状态的实例对象。

State：状态接口，用来封装与上下文的一个特定状态所对应的行为。

ConcreteState：具体实现状态处理的类，每个类实现一个状态的具体处理。

状态模式的知识要点

- 状态模式的知识要点

1：状态模式的功能就是分离状态的行为，通过维护状态的变化，来调用不同的状态对应的不同的功能

2：状态和行为是相关联的，它们的关系可以描述为：**状态决定行为。**

3：状态模式的状态对应的行为具有平行性，平行性指的是各个状态的行为相互是独立的、没有关联的，相互之间是不可替换的。而策略模式的各个策略之间是平等性，平等性强调的是可替换性，大家是同一行为的不同描述或实现，因此在同一个行为发生的时候，可以根据条件来挑选任意一个实现来进行相应的处理

4：状态模式中，客户端通常不负责运行期间状态的维护，也不负责决定到底后续使用哪一个具体的状态处理对象

5：在状态模式中，通常有两个地方可以进行状态的维护和转换控制。一个是在上下文中，另外一个地方就是在状态的处理类里面

6：由于状态模式对状态的维护是一个内部行为，因此增加新的状态的时候，就需要修改内部的状态维护的代码，算是一种不完美的OCP

7：由于状态模式要维护状态，也就是数据，因此要考虑何时创建和销毁状态对象

8：可以使用数据库来记录和维护状态

思考状态模式

- 状态模式的本质

  状态模式的本质是：**根据状态来分离和选择行为**

- 何时选用状态模式

1：如果一个对象的行为取决于它的状态，而且它必须在运行时刻根据状态来改变它的行为。可以使用状态模式，来把状态和行为分离开，虽然分离开了，但状态和行为是有对应关系的，可以在运行期间，通过改变状态，就能够调用到该状态对应的状态处理对象上去，从而改变对象的行为

2：如果一个操作中含有庞大的多分支语句，而且这些分支依赖于该对象的状态。可以使用状态模式，把各个分支的处理分散包装到单独的对象处理类里面，这样，这些分支对应的对象就可以不依赖于其它对象而独立变化了

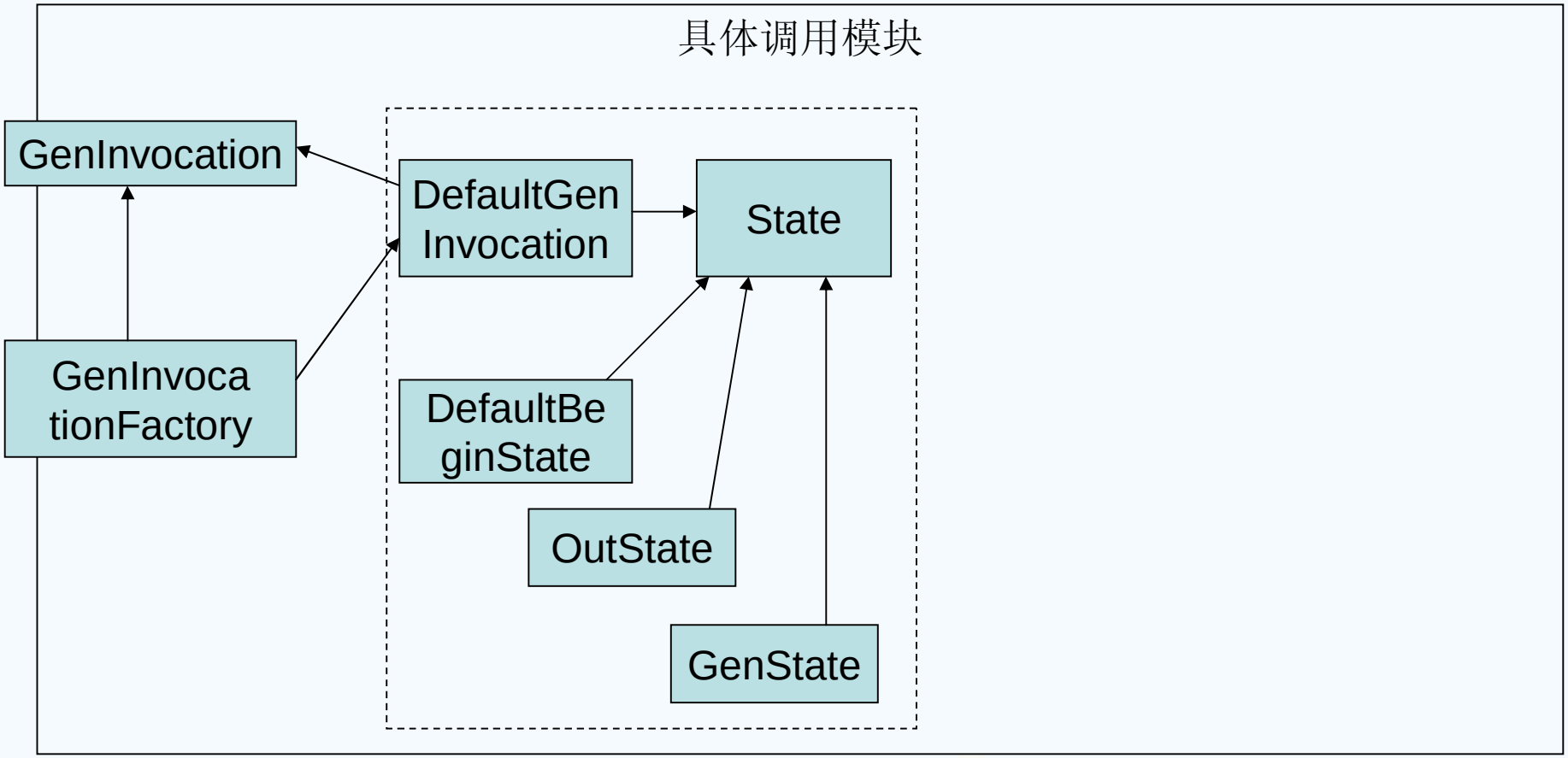
应用状态模式

- 使用状态模式来解决问题的思路

  要解决上面的问题，一个很简单的思路就是：其实就是相当于用状态模式来模拟调用的工作流程，把调用流程的每一步单独包装成为一个状态对象，然后在一个对象执行过后，由这个状态对象来设置下一步状态，使得流程继续流转。

应用状态模式

- 此时具体调用模块的结构示意如图



加入模板方法模式

- 面临的问题

  在调用每个theme的action来generate内容的时候，虽然action是由theme提供的，也就是由开发人员自行扩展的，但是调用action的过程是一样的。

  有些朋友会想，调用过程一样的，那就做成公共的功能吧，但是又发现其中某些步骤执行的功能并不一样，比如具体的action的处理是不一样的，这该如何处理呢？

- 用模板方法模式来解决

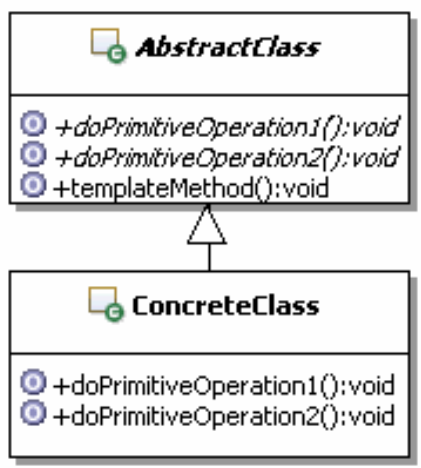
- 模板方法模式基础回顾

初识模板方法模式

- 定义

  定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

- 结构和说明



AbstractClass：抽象类。用来定义算法骨架和原语操作，在这个类里面，还可以提供算法中通用的实现。

ConcreteClass： 具体实现类。用来实现算法骨架中的某些步骤，完成跟特定子类相关的功能。

模板方法模式的知识要点

- 模板方法模式的知识要点

1：模板方法模式的功能在于固定算法骨架，而让具体算法实现可扩展

2：模板方法模式通常实现成为抽象类，而不是接口

3：程序设计的一个很重要的思考点就是“变与不变”，模板类实现的就是不变的方法和算法的骨架，而需要变化的地方，都通过抽象方法，把具体实现延迟到子类去了，而且还通过父类的定义来约束了子类的行为，从而使系统能有更好的复用性和扩展性

4：模板方法模式体现了好莱坞法则，是一种父类来找子类的反向的控制结构

5：可以使用Java回调来实现模板方法，回调机制是通过委托的方式来组合功能，它的耦合强度要比继承低一些，这会给我们更多的灵活性。

模板方法模式的知识要点

6：要重点区分模板方法模式中的一些常见类型：

（1）模板方法：就是定义算法骨架的方法

（2）具体的操作：在模板中直接实现某些步骤的方法，通常这些步骤的实现算法是固定的，而且是不怎么变化的，因此就可以当作公共功能实现在模板里面。

（3）具体的AbstractClass操作：在模板中实现某些公共功能，可以提供给子类使用，一般不是具体的算法步骤的实现，只是一些辅助的公共功能

（4）原语操作：就是在模板中定义的抽象操作，通常是模板方法需要调用的操作，是必需的操作

（5）钩子操作：在模板中定义，并提供默认实现的操作。这些方法通常被视为可扩展的点，但不是必须的，子类可以有选择的覆盖这些方法。

（6）Factory Method：在模板方法中，如果需要得到某些对象实例的话，可以考虑通过工厂方法模式来获取，把具体的构建对象的实现延迟到子类中去

思考模板方法模式

- 模板方法模式的本质

  模板方法模式的本质是：**固定算法骨架**

- 何时选用模板方法模式

1：需要固定定义算法骨架，实现一个算法的不变的部分，并把可变的行为留给子类来实现的情况。

2：各个子类中具有公共行为，应该抽取出来，集中在一个公共类中去实现，从而避免代码重复

3：需要控制子类扩展的情况。模板方法模式会在特定的点来调用子类的方法，这样只允许在这些点进行扩展

应用模板方法模式

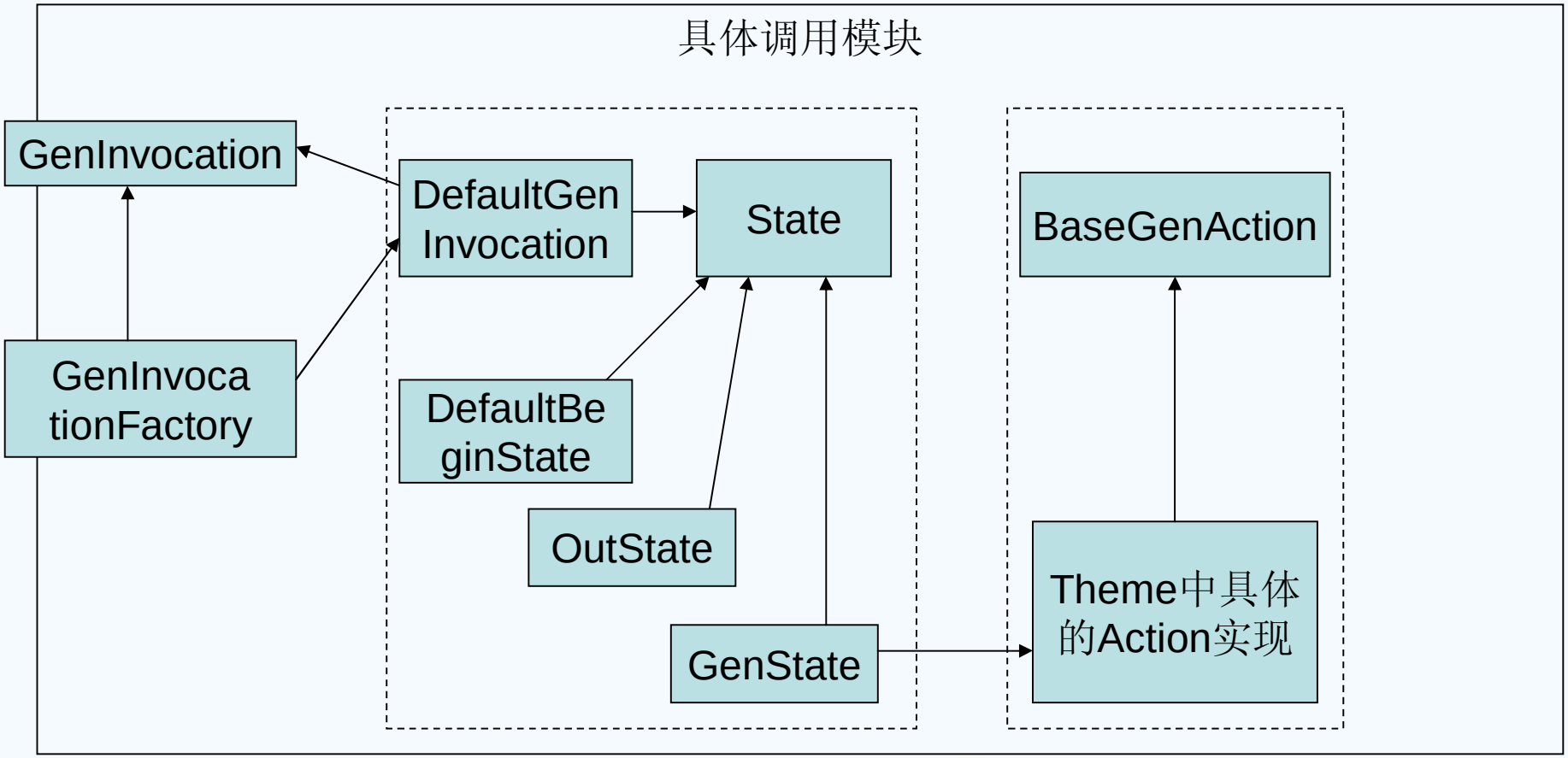
- 使用模板方法模式来解决问题的思路

  既要约束子类的功能，又要为子类提供公共的功能，很明显应用抽象类来做一个公共的父类。而公共功能中某些步骤执行的功能并不一样，那就可以把这些不一样的功能延迟到子类来执行，在父类中只要定义相应的抽象方法就好了。

  从而把调用过程看成是算法的骨架，而调用action的处理方法，当作原语操作，只有子类才知道如何实现具体的generate；而在action操作前后调用的方法，并不是每个generate都需要的功能，因此可以当作钩子方法来实现，也就是在父类中给出默认的实现，而子类根据需要来覆盖。

应用模板方法模式

- 此时具体调用模块的结构示意如图



加入工厂方法模式

- 面临的问题

  观察上面的实现，发现在模板方法generate里面，调用的第一步，需要得到用来封装generate内容的对象。

  但是在父类里面，它并不知道究竟要generate成为什么样子，只有theme中的action才知道具体是什么类型，也就是说父类不知道某个对象的类型，也不知道如何获取这个对象，该怎么办呢？

- 用工厂方法模式来解决

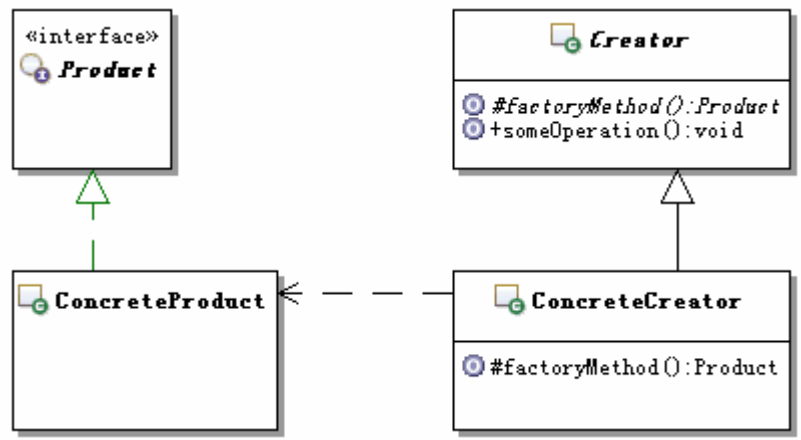
- 工厂方法模式基础回顾

初识工厂方法模式

- 定义

  定义一个用于创建对象的接口，让子类决定实例化哪一个类，Factory Method使一个类的实例化延迟到其子类。

- 结构和说明



Product：定义工厂方法所创建的对象的接口，也就是实际需要使用的对象的接口
ConcreteProduct：具体的Product接口的实现对象。
Creator：创建器，声明工厂方法
ConcreteCreator：具体的创建器对象，覆盖实现Creator定义的工厂方法，返回具体的Product实例

工厂方法模式的知识要点

n 工厂方法模式的知识要点

1：工厂方法的主要功能是让父类在不知道具体实现的情况下，完成自身的功能调用，而具体的实现延迟到子类来实现

2：工厂方法的实现中，通常父类会是一个抽象类，里面包含创建所需对象的抽象方法，这些抽象方法就是工厂方法

3：子类在实现抽象方法的时候，通常并不是真的由子类来实现具体功能，而是在子类的方法里面做选择，选择具体产品实现对象，有些类似于简单工厂的功能，也就是“选择实现”

4：可以在抽象方法里面传递参数，以便在子类实现的时候根据参数进行选择，创建需要的对象实例

5：还有一个重要的问题就是：谁在使用工厂方法创建的对象？通常情况下工厂方法创建的对象应该是在父类里面的其它方法来使用的，一般不直接把这个对象返回给客户端

6：工厂方法模式的和IoC/DI从思想层面上是相似的，都是**“主动变被动”**，进行了**“主从换位”**，从而获得了更灵活的程序结构

7：所谓参数化工厂方法指的就是：通过给工厂方法传递参数，让工厂方法根据参数的不同来创建不同的产品对象

8：要理解平行的类层次结构，工厂方法模式可以用于连接平行的类层次

思考工厂方法模式

- 工厂方法模式的本质

  工厂方法模式的本质是：**延迟到子类来选择实现**

- 何时选用工厂方法模式

1：如果一个类需要创建某个接口的对象，但是又不知道具体的实现，这种情况可以选用工厂方法模式，把创建对象的工作延迟到子类去实现

2：如果一个类本身就希望，由它的子类来创建所需的对象的时候，应该使用工厂方法模式

应用工厂方法模式

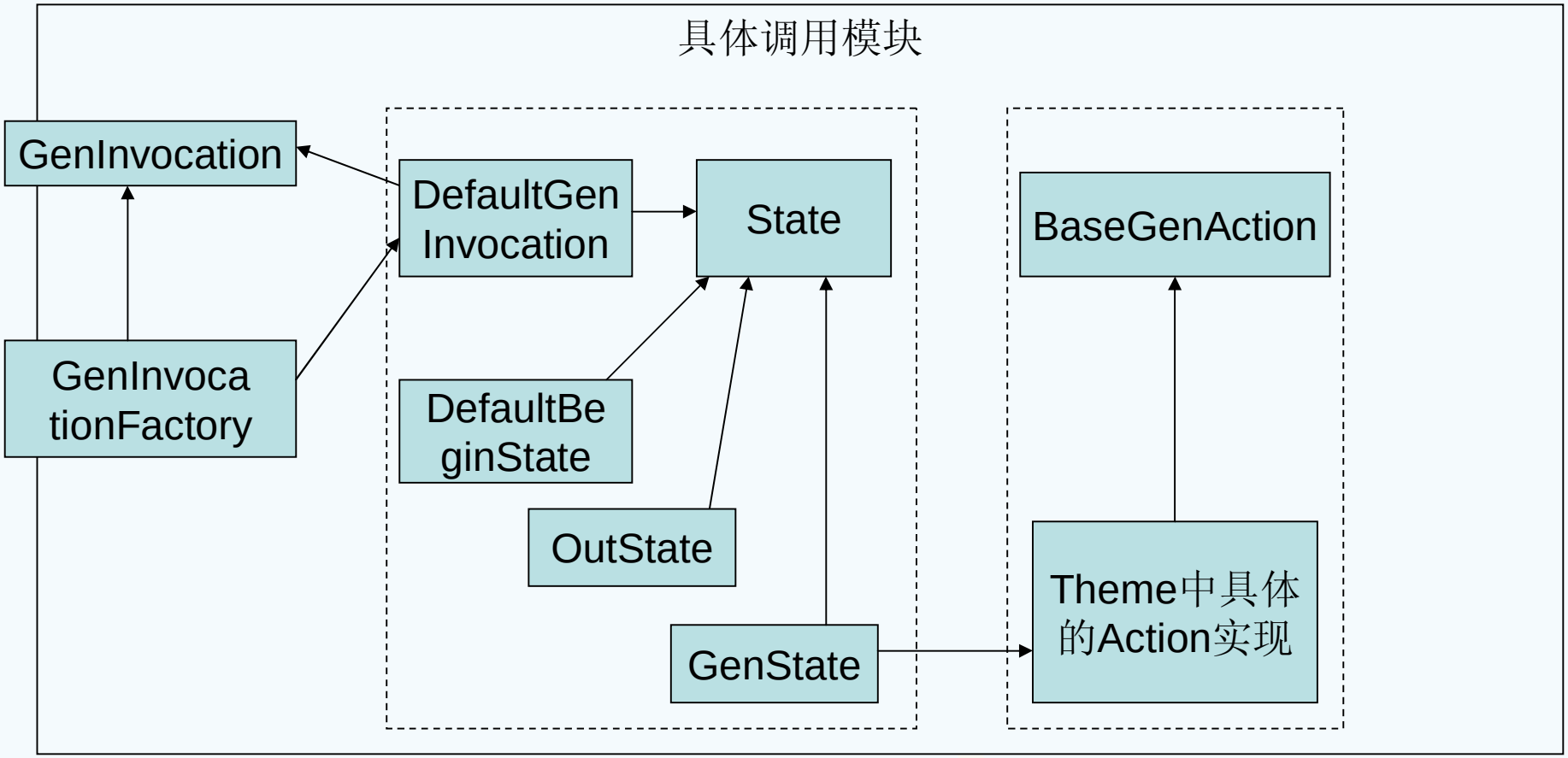
- 使用工厂方法模式来解决问题的思路

  要解决上面的问题，一个很简单的思路就是：在父类里面定义一个抽象方法，返回模板方法需要的对象，把初始化对象的任务延迟到子类去。

  这个方法就是一个工厂方法，同时也作为模板方法模式里的原语操作之一。

应用工厂方法模式

- 此时具体调用模块的结构示意如图



加入装饰者模式

- 面临的问题

  分析上面的实现，在调用模板方法generate的过程中，可能需要在具体的Action之前执行一定的功能，也可能需要在具体的Action之后执行一定的功能，而且这些功能具体是什么，具体的Action方法并不知道。

  也就是说，需要在Action方法不知情的情况下，给他添加上新的功能，可能会在action方法之前或者之后执行一些功能。该怎么办呢？

- 用装饰者模式来解决

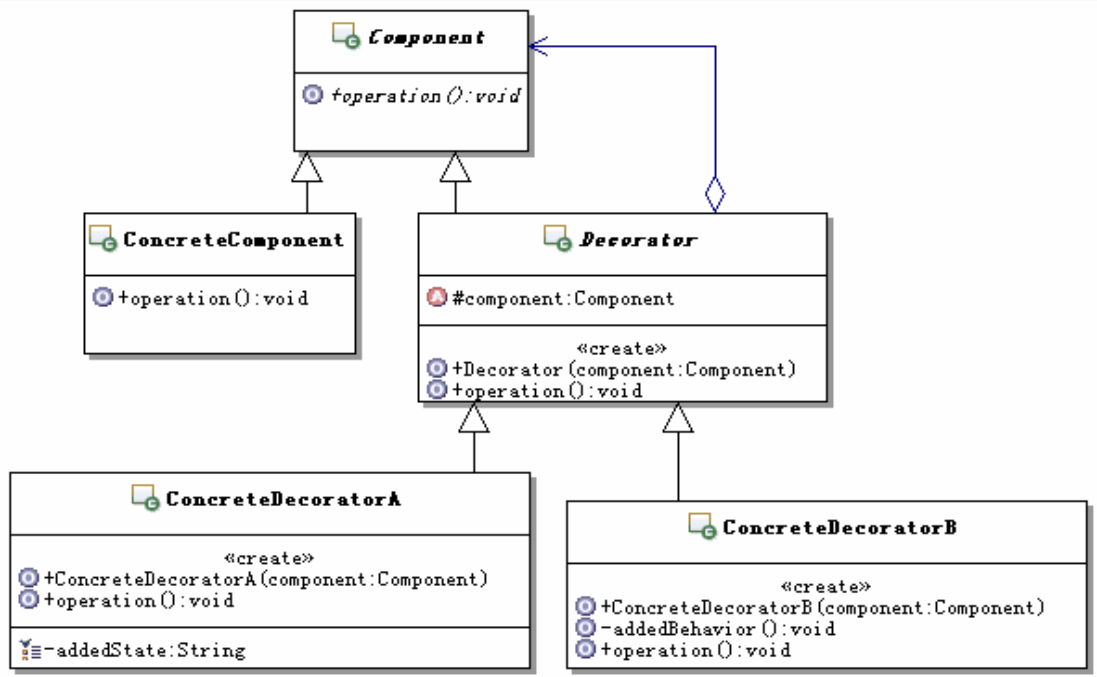
- 装饰者模式基础回顾

初识装饰模式

- 定义

  动态地给一个对象添加一些额外的职责。就增加功能来说，装饰模式比生成子类更为灵活。

- 结构和说明



装饰者模式的知识要点

- 装饰者模式的知识要点

1：装饰模式能够实现动态的为对象添加功能，是从一个对象外部来给对象增加功能，相当于是改变了对象的外观，增强了系统的灵活性，而且也使得装饰器功能得到复用

2：装饰模式的思考起点是“尽量使用对象组合，而不是对象继承”来扩展和复用功能

3：装饰器是能够包含其它的装饰器对象的，相当于组合结构中的树枝对象，只不过装饰器对象只能包含一个装饰器对象，而不是多个。因此装饰模式的结构图根组合模式的结构图是相似的

4：装饰器对象的调用，也是要使用递归调用的

5：在装饰器里不仅仅是可以给被装饰对象增加功能，还可以根据需要选择是否调用被装饰对象的功能，如果不调用被装饰对象的功能，那就变成完全重新实现了，相当于动态修改了被装饰对象的功能

装饰者模式的知识要点

6：装饰器是用来装饰组件的，装饰器一定要实现和组件类一致的接口，保证它们是同一个类型，并具有同一个外观，这样组合完成的装饰才能够递归的调用下去

7：组件类是不知道装饰器的存在的，装饰器给组件添加功能是一种透明的包装，组件类毫不知情

8：装饰模式和AOP要实现的功能是类似的，只不过AOP的实现方法不同，会更加灵活，更加可配置；另外AOP**一个更重要的变化是思想上的变化——“主从换位”**，让原本主动调用的功能模块变成了被动等待，甚至毫不知情的情况下被织入了很多新的功能

思考装饰模式

- 装饰模式的本质

  装饰模式的本质是：**动态组合**

- 何时选用装饰模式

1：如果需要在不影响其它对象的情况下，以动态、透明的方式给对象添加职责，可以使用装饰模式，这几乎就是装饰模式的主要功能

2：如果不合适使用子类来进行扩展的时候，可以考虑使用装饰模式，因为装饰模式是使用的“对象组合”的方式。所谓不适合用子类扩展的方式，比如：扩展功能需要的子类太多，造成子类数目呈爆炸性增长。

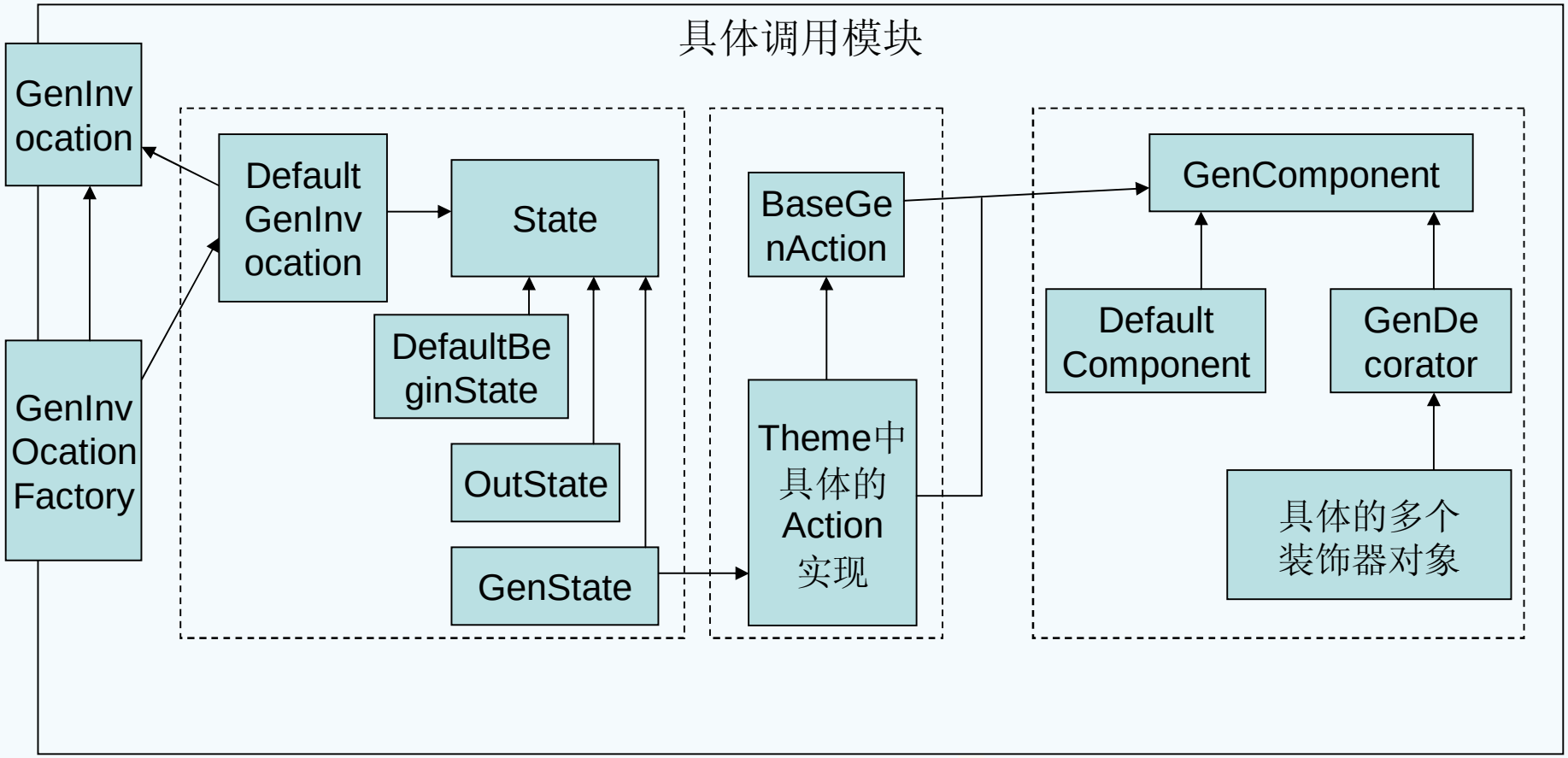
应用装饰者模式

- 使用装饰者模式来解决问题的思路

  要解决上面的问题，一个很简单的思路就是：定义一个公共的组件对象，作为原始对象和装饰对象的父类，然后在装饰对象里面，采用动态组合的方式来给传入的组件对象透明的添加新的功能

应用装饰者模式

- 此时具体调用模块的结构示意如图



加入观察者模式

- 面临的问题

  继续分析上面的实现，在加入了模板方法模式、工厂方法模式和装饰器模式过后，在DefaultGenInvocation的executeGen方法里面，调用每个theme的action来生成内容的功能就很容易实现出来了。

  可是，把内容输出去的步骤应该怎么实现呢？

  按照现在的要求，同一个generate的功能，可能会有多个输出，输出形式还不一样，那么，该怎么实现这个功能呢？

- 用观察者模式来解决

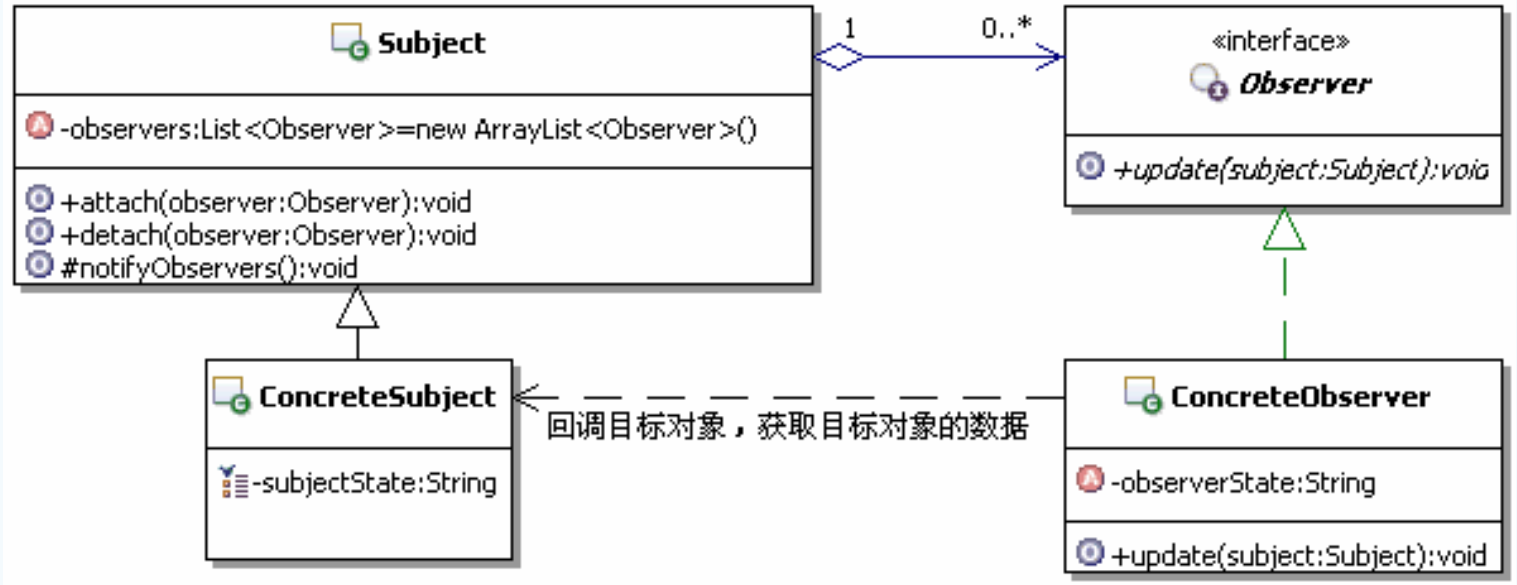
- 观察者模式基础回顾

初识观察者模式

- 定义

  定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

- 结构和说明



观察者模式的知识要点

n 观察者模式的知识要点

1：观察者模式中的目标和观察者之间是典型的一对多的关系，但是也可以只有你一个观察者，也就是可以变相实现了目标和观察者之间一对一的关系

2：一个观察者可以观察多个目标，通常在这种情况下，观察者应该为不同的观察者目标，定义不同的回调方法

3：在观察者模式中，观察者和目标是单向依赖的，只有观察者依赖于目标，而目标是不会依赖于观察者的

4：观察者模式的目标对象在实现的时候，通常需要维护观察者的注册信息

5：目标在通知观察者的时候，一般要确保目标的状态更新已经结束，否则会导致观察者和目标对象的状态不一致

6：如果出现目标和观察者相互观察的情况，一定要特别小心处理，因为可能会出现死循环的情况

7：观察者模式在具体实现中，又分为推模型和拉模型，可以同时实现这两种模型

8：可以在程序运行期间，通过动态的控制观察者，来变相的实现添加和删除某些功能处理，这些功能就是观察者在update的时候执行的功能

思考观察者模式

- 观察者模式的本质

  观察者模式的本质是：**触发联动**

- 何时选用观察者模式

1：当一个抽象模型有两个方面，其中一个方面的操作依赖于另一个方面的状态变化，那么就可以选用观察者模式。

2：如果在更改一个对象的时候，需要同时连带改变其它的对象，而且不知道究竟应该有多少对象需要被连带改变，这种情况可以选用观察者模式，被更改的那一个对象很明显就相当于是目标对象，而需要连带修改的多个其它对象，就作为多个观察者对象了。

3：当一个对象必须通知其它的对象，但是你又希望这个对象和其它被它通知的对象是松散耦合的，也就是说这个对象其实不想知道具体被通知的对象，这种情况可以选用观察者模式，这个对象就相当于是目标对象，而被它通知的对象就是观察者对象了。

应用观察者模式

- 使用观察者模式来解决问题的思路

要解决上面的问题，一个很简单的思路就是： 把生成调用当成目标对象，而需要输出的对象当成多个观察者，这样一来，当目标对象的状态变化的时候，也就是需要输出的内容已经generate过后，就可以通知这多个观察者，让这些观察者自行输出内容。

应用观察者模式

- 此时具体调用模块的结构示意如图