私塾在线《研磨设计模式》
——跟着CC学设计系列精品教程

本节课程概览

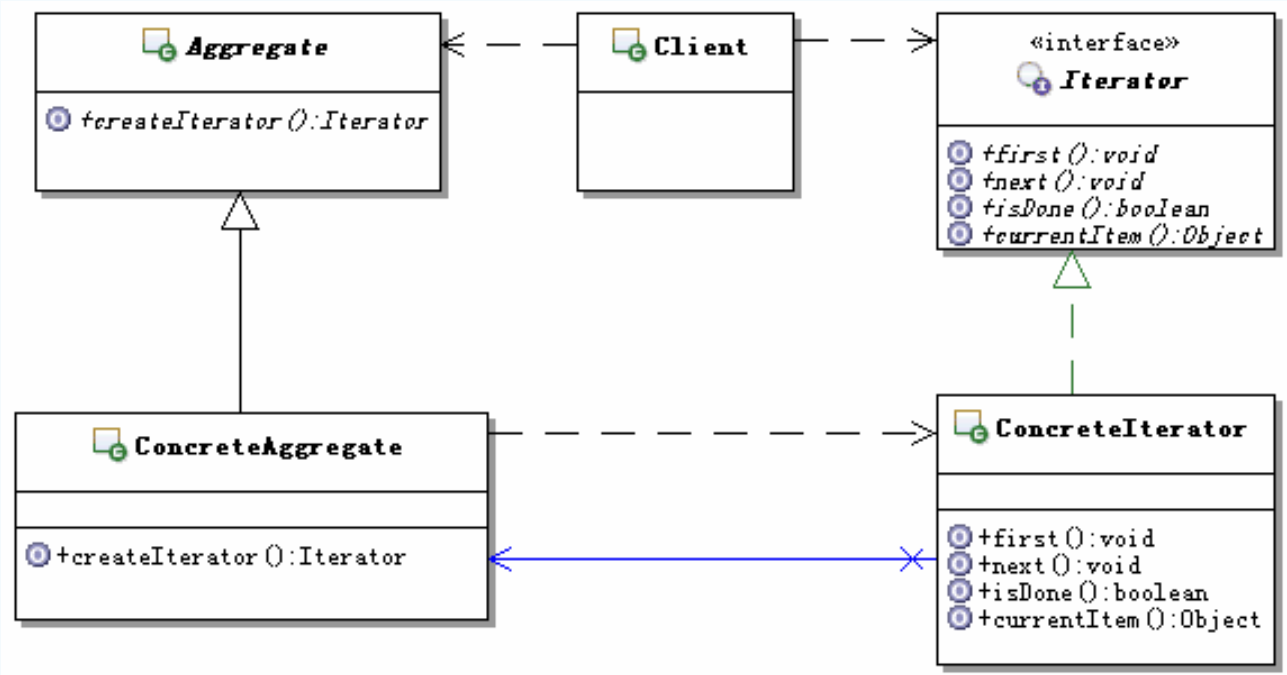
- 学习迭代器模式
  - 一：初识迭代器模式
    - 包括：定义、结构、参考实现
  - 二：体会迭代器模式
    - 包括：场景问题、使用模式的解决方案
  - 三：理解迭代器模式
    - 包括：认识迭代器模式、使用Java的迭代器、带迭代策略的迭代器 、双向迭代器 、迭代器模式的优缺点
  - 四：思考迭代器模式
    - 包括：迭代器模式的本质、何时选用、翻页迭代

初识迭代器模式

- 定义

  提供一种方法顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示 。

- 结构和说明



初识迭代器模式

Iterator：

迭代器接口。定义访问和遍历元素的接口。

ConcreteIterator：

具体的迭代器实现对象。实现对聚合对象的遍历，并跟踪遍历时的当前位置。

Aggregate：

聚合对象。定义创建相应迭代器对象的接口。

ConcreteAggregate:

具体聚合对象。实现创建相应的迭代器对象。

体会迭代器模式

- 工资表数据的整合

这个项目的背景是这样的，项目的客户方收购了一家小公司，这家小公司有自己的工资系统，现在需要整合到客户方已有的工资系统上。

现在除了要把两个工资系统整合起来外，老板还希望能够通过决策辅助系统来统一查看工资数据，他不想看到两份不同的工资表。那么应该如何实现呢？

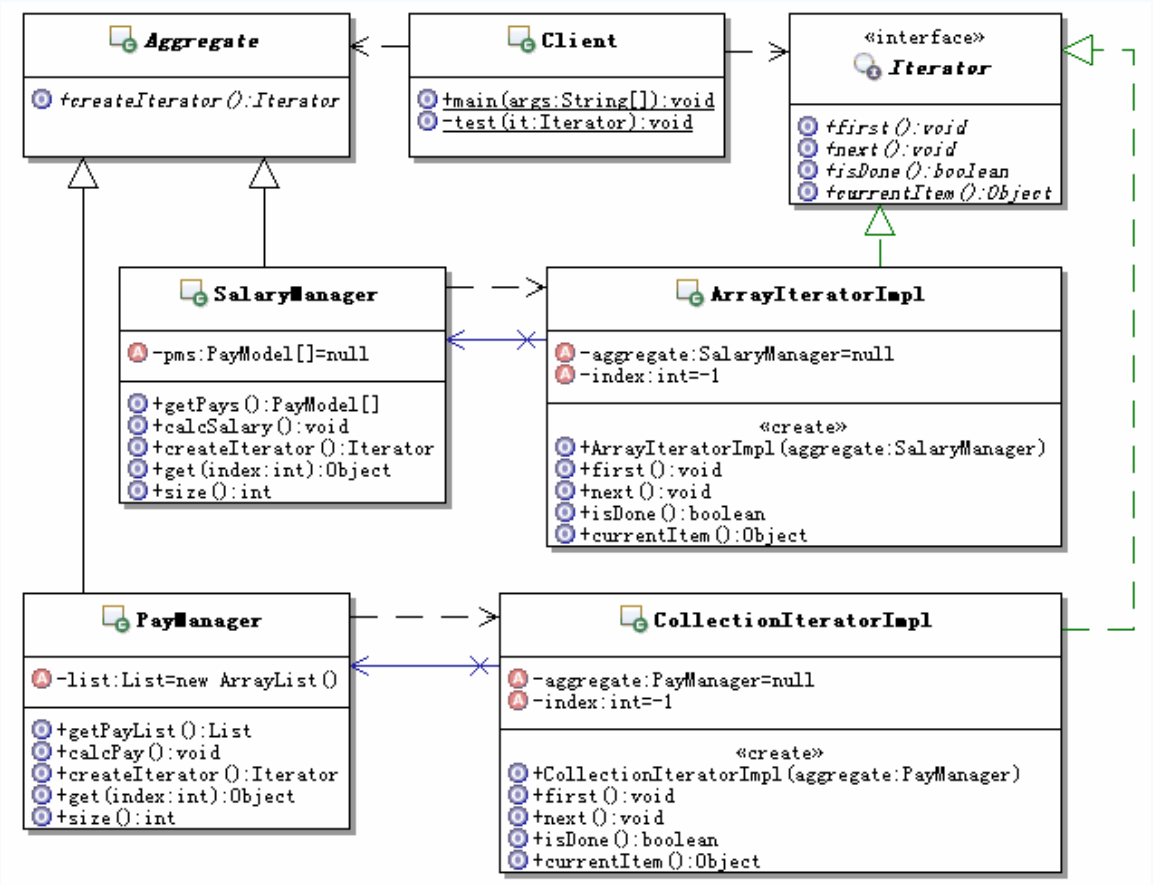
- 有何问题

本来就算内部描述形式不一样，只要不需要整合在一起，两个系统单独输出自己的工资表，是没有什么问题的。但是，老板还希望能够以一个统一的方式来查看所有的工资数据，也就是说从外部看起来，两个系统输出的工资表应该是一样的。

经过分析，要满足老板的要求，而且要让两边的系统改动都尽可能的小的话，问题的核心就在于如何能够以一种统一的方式来提供工资数据给决策辅助系统，换句说来说就是：**如何能够以一个统一的方式来访问内部实现不同的聚合对象。**

体会迭代器模式

- 使用模式的解决方案的类图



理解迭代器模式

n 认识迭代器模式

1：迭代器模式的功能

迭代器模式的功能主要在于提供对聚合对象的迭代访问。迭代器就围绕着这个“访问”做文章，延伸出很多的功能来。比如：

（1）以不同的方式遍历聚合对象，比如向前、向后等

（2）对同一个聚合同时进行多个遍历

（3）以不同的遍历策略来遍历聚合，比如是否需要过滤等

（4）多态迭代，含义是：为不同的聚合结构，提供统一的迭代接口，也就是说通过一个迭代接口可以访问不同的聚合结构，这就叫做多态迭代。上面的示例就已经实现了多态迭代，事实上，标准的迭代模式实现基本上都是支持多态迭代的。

但是请注意：多态迭代可能会带来类型安全的问题，可以考虑使用泛型

理解迭代器模式

2：迭代器模式的关键思想

聚合对象的类型很多，如果对聚合对象的迭代访问跟聚合对象本身融合在一起的话，会严重影响到聚合对象的可扩展性和可维护性。

因此**迭代器模式的关键思想就是把对聚合对象的遍历和访问从聚合对象中分离出来，放入到单独的迭代器中**，这样聚合对象会变得简单一些；而且迭代器和聚合对象可以独立的变化和发展，会大大加强系统的灵活性。

3：内部迭代器和外部迭代器

所谓内部迭代器，指的是由迭代器自己来控制迭代下一个元素的步骤，客户端无法干预，因此，如果想要在迭代的过程中完成工作的话，客户端就需要把操作传给迭代器，迭代器在迭代的时候会在每个元素上执行这个操作，类似于Java的回调机制。

所谓外部迭代器，指的是由客户端来控制迭代下一个元素的步骤，像前面的示例一样，客户端必须显示的调用next来迭代下一个元素。

总体来说外部迭代器比内部迭代器要灵活一些，因此我们常见的实现多属于外部迭代器，前面的例子也是实现的外部迭代器。

理解迭代器模式

4：Java中最简单的统一访问聚合的方式

如果只是想要使用一种统一的访问方式来访问聚合对象，在Java中有更简单的方式，简单到几乎什么都不用做，利用Java 5以上版本本身的特性即可。

但是请注意，这只是从访问形式上一致了，但是也暴露了聚合的内部实现，因此并不能算是标准迭代器模式的实现，但是从某种意义上说，可以算是隐含的实现了部分迭代器模式的功能。

理解迭代器模式

- 使用Java的迭代器

大家都知道，在java.util包里面有一个Iterator的接口，在Java中实现迭代器模式是非常简单的，而且java的集合框架中的聚合对象，基本上都是提供了迭代器的。

下面就来把前面的例子改成用Java中的迭代器实现，一起来看看有些什么改变。

1：不再需要自己实现的Iterator接口，直接实现java.util.Iterator接口就可以了，所有使用自己实现的Iterator接口的地方都需要修改过来

2：Java中Iterator接口跟前面自己定义的接口相比，需要实现的方法是不一样的

3：集合已经提供了Iterator，那么CollectionIteratorImpl类就不需要了，直接去掉

理解迭代器模式

- 带迭代策略的迭代器

由于迭代器模式把聚合对象和访问聚合的机制实现了分离，所以可以在迭代器上实现不同的迭代策略，最为典型的就是实现过滤功能的迭代器。

在实际开发中，对于经常被访问的一些数据可以使用缓存，把这些数据存放在内存中。但是不同的业务功能需要访问的数据是不同的，还有不同的业务访问权限能访问的数据也是不同的，对于这种情况，就可以使用实现过滤功能的迭代器，让不同功能使用不同的迭代器来访问。当然，这种情况也可以结合策略模式来实现。

在实现过滤功能的迭代器中，又有两种常见的需要过滤的情况，一种是对数据进行整条过滤，比如只能查看自己部门的数据；另外一种情况是对数据进行部分过滤，比如某些人不能查看工资数据。

理解迭代器模式

- 谁定义遍历算法的问题

在迭代器模式的实现中，常见有两个地方可以来定义遍历算法，一个就是聚合对象本身，另外一个就是迭代器负责遍历算法。

在聚合对象本身定义遍历的算法这种情况下，通常会在遍历过程中，用迭代器来存储当前迭代的状态，这种迭代器被称为游标，因为它仅用来指示当前的位置。

在迭代器里面定义遍历算法，会易于在相同的聚合上使用不同的迭代算法，同时也易于在不同的聚合上重用相同的算法。比如上面带策略的迭代器的示例，迭代器把需要迭代的数据从聚合对象中取出并存放到自己对象里面，然后再迭代自己的数据，这样一来，除了刚开始创建迭代器的时候需要访问聚合对象外，真正迭代过程已经跟聚合对象无关了。

理解迭代器模式

- 双向迭代器

所谓双向迭代器的意思就是：可以同时向前和向后遍历数据的迭代器。

在Java util包中的ListIterator接口就是一个双向迭代器的示例，当然自己实现双向迭代器也非常容易，只要在自己的Iterator接口中添加向前的判断和向前获取值的方法，然后在实现中实现即可。

理解迭代器模式

- 迭代器模式的优缺点
  1：更好的封装性
  2：可以以不同的遍历方式来遍历一个聚合
  3：迭代器简化了聚合的接口
  4：简化客户端调用
  5：同一个聚合上可以有多个遍历

思考迭代器模式

- 迭代器模式的本质

  迭代器模式的本质是：**控制访问聚合对象中的元素**

- 何时选用迭代器模式

1：如果你希望提供访问一个聚合对象的内容，但是又不想暴露它的内部表示的时候，可以使用迭代器模式来提供迭代器接口，从而让客户端只是通过迭代器的接口来访问聚合对象，而无需关心聚合对象内部实现。

2：如果你希望有多种遍历方式可以访问聚合对象，可以使用迭代器模式

3：如果你希望为遍历不同的聚合对象提供一个统一的接口，可以使用迭代器模式

思考迭代器模式

- 翻页迭代

在实际开发中很常用的翻页功能的实现，常见的翻页功能有如下几种实现方式：

（1）纯数据库实现

依靠SQL提供的功能实现翻页，用户每次请求翻页的数据，就会到数据库中获取相应的数据

（2）纯内存实现

就是一次性从数据库中把需要的所有数据都取出来放到内存中，然后用户请求翻页时，从内存中获取相应的数据

（3）上面两种方式各有优缺点：

第一种方案明显是时间换空间的策略，每次获取翻页的数据都要访问数据库，运行速度相对比较慢，而且很耗数据库资源，但是节省内存空间。

第二种方案是典型的空间换时间，每次是直接从内存中获取翻页的数据，运行速度快，但是很耗内存。

思考迭代器模式

（4）纯数据库实现 + 纯内存实现

思路是这样的：如果每页显示10条记录，根据判断，用户很少翻到10页过后，那好了，第一次访问的时候，就一次性从数据库中获取前10页的数据，也就是100条记录，把这100条记录放在内存里面。

这样一来，当用户在前10页内进行翻页操作的时候，就不用再访问数据库了，而是直接从内存中获取数据，这样速度就快了。

当用户想要获取第11页的数据，这个时候才会再次访问数据库，对于这个时候到底获取多少页的数据，简单的处理就是继续获取10页的数据，比较好的方式就是根据访问统计进行衰减访问，比如折半获取，也就是第一次访问数据库获取10页的数据，那么第二次就只获取5页，如此操作直到一次从数据库中获取一页的数据。这也符合正常规律，因为越到后面，被用户翻页到的机会也就越小了。