## 第19章 地铁安检，谁都逃不掉——迭代器模式

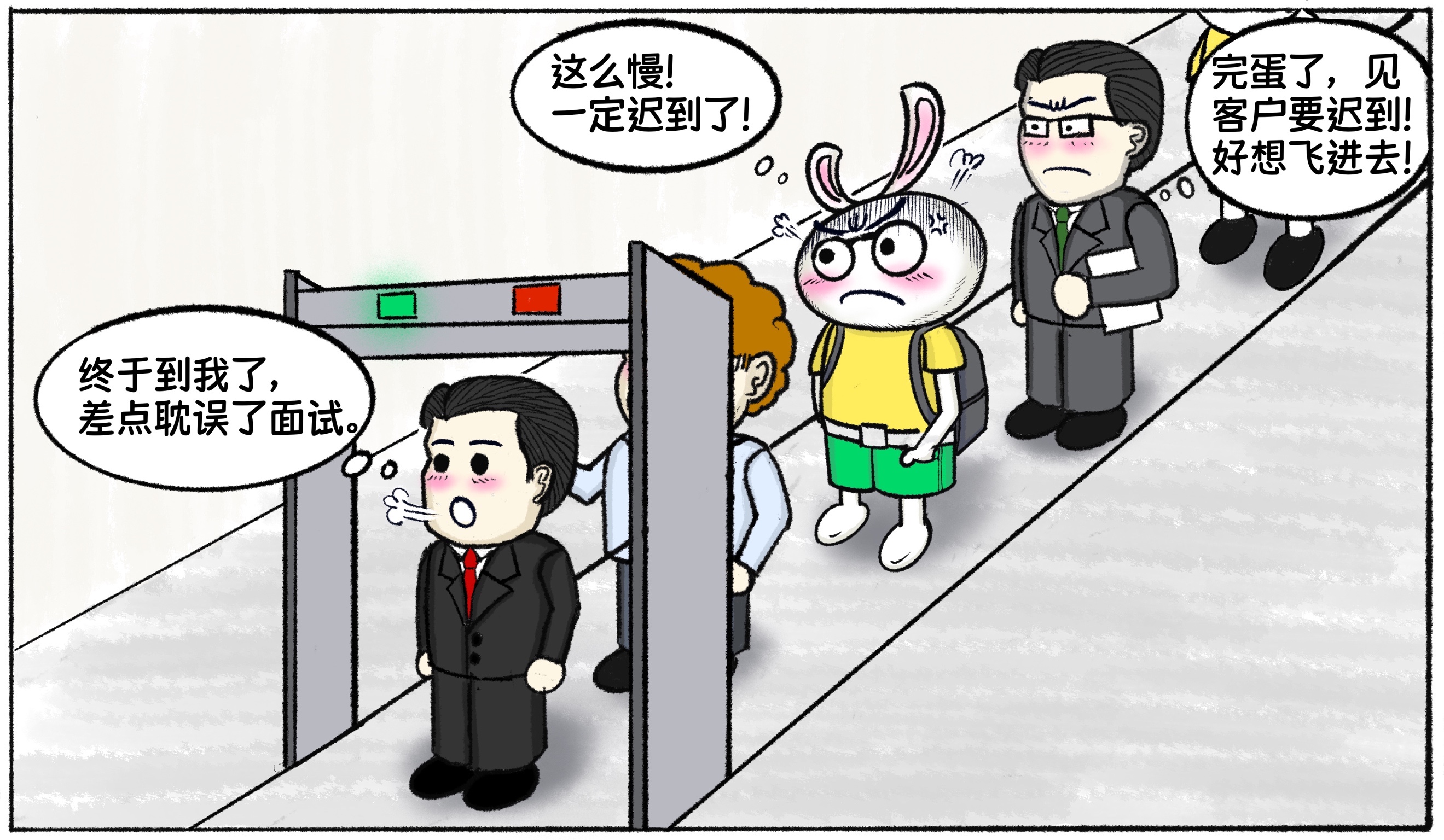
## 19.1 兔小白上班迟到

周一早上9:35，技术团队已经开始晨会，兔小白才急急忙忙地跑进办公室。会后，兔小白向熊小猫抱怨起来。

兔小白：今天早上可急死我了，紧赶慢赶还是迟到了。

熊小猫：你是不是睡懒觉起晚了？

兔小白：我还真没睡懒觉，迟到是因为地铁安检耽误时间了。今天可能有重要活动，地铁安检比平时严格，导致安检队伍非常长，过安检比平时多花了10分钟。



熊小猫：上班时间不能卡这么死呀！多花10分钟，你就迟到了。软件设计也是同样的道理，两个组件之间要解耦，要留有变化的空间。做任何事情都要留余地，否则一旦变化发生，就难以应对。

兔小白：你说的在理。早上我干着急没办法，安检的队伍只能一个个地通过，谁也逃不掉。

熊小猫：安检队伍好比Java中的集合。安检员迭代集合中的每位乘客，如果漏了一位，那不就出Bug了。

兔小白：真有你的，这都能联想到开发上来。我倒希望有Bug发生，让我快点通过安检。

熊小猫：安检是为了保护每一位乘客的安全，肯定不能出Bug呀！再说了，就算在程序中迭代集合，也不至于漏掉里面的元素吧？

兔小白：这可不一定，我就犯过这种错误。

熊小猫：哦？让我看看你是怎么写出bug来的。

## 19.2 迭代不就是写for循环吗？

熊小猫：咱们就以安检为例，写一个非常简单的小练习。需求是迭代乘客列表，对每一位乘客进行安检。

兔小白：这还不简单，5分钟就搞定。

大约5分钟后，兔小白写完了代码。

兔小白：我定义了Passenger类，属性withDangerousGoods表示是否携带危险品。

public class Passenger {  
    private boolean withDangerousGoods;  
​  
    public Passenger(boolean withDangerousGoods) {  
        this.withDangerousGoods = withDangerousGoods;  
   }  
​  
    public boolean isWithDangerousGoods() {  
        return withDangerousGoods;  
   }  
}

客户端代码中，声明了一个List，放入几个Passenger对象，然后用for循环迭代乘客List，进行安检。

List<Passenger> passengers = new ArrayList<>();  
passengers.add(new Passenger(false));  
passengers.add(new Passenger(false));  
passengers.add(new Passenger(true));  
passengers.add(new Passenger(false));  
​  
for (int i = 0; i < passengers.size(); i++) {  
  Passenger passenger = passengers.get(i);  
  if (passenger.isWithDangerousGoods()) {  
    System.out.println("安检未通过");  
 } else {  
    System.out.println("安检通过");  
 }  
}

输入出如下，符合预期。

安检通过  
安检通过  
安检未通过  
安检通过

这段程序可能出Bug的地方是for循环的写法。我现在很熟悉，肯定不会犯错。但是对于一些初学者，可能会把i的初始值设为了1，也可能将i < passengers.size()的边界判断，写成了i <= passengers.size()。

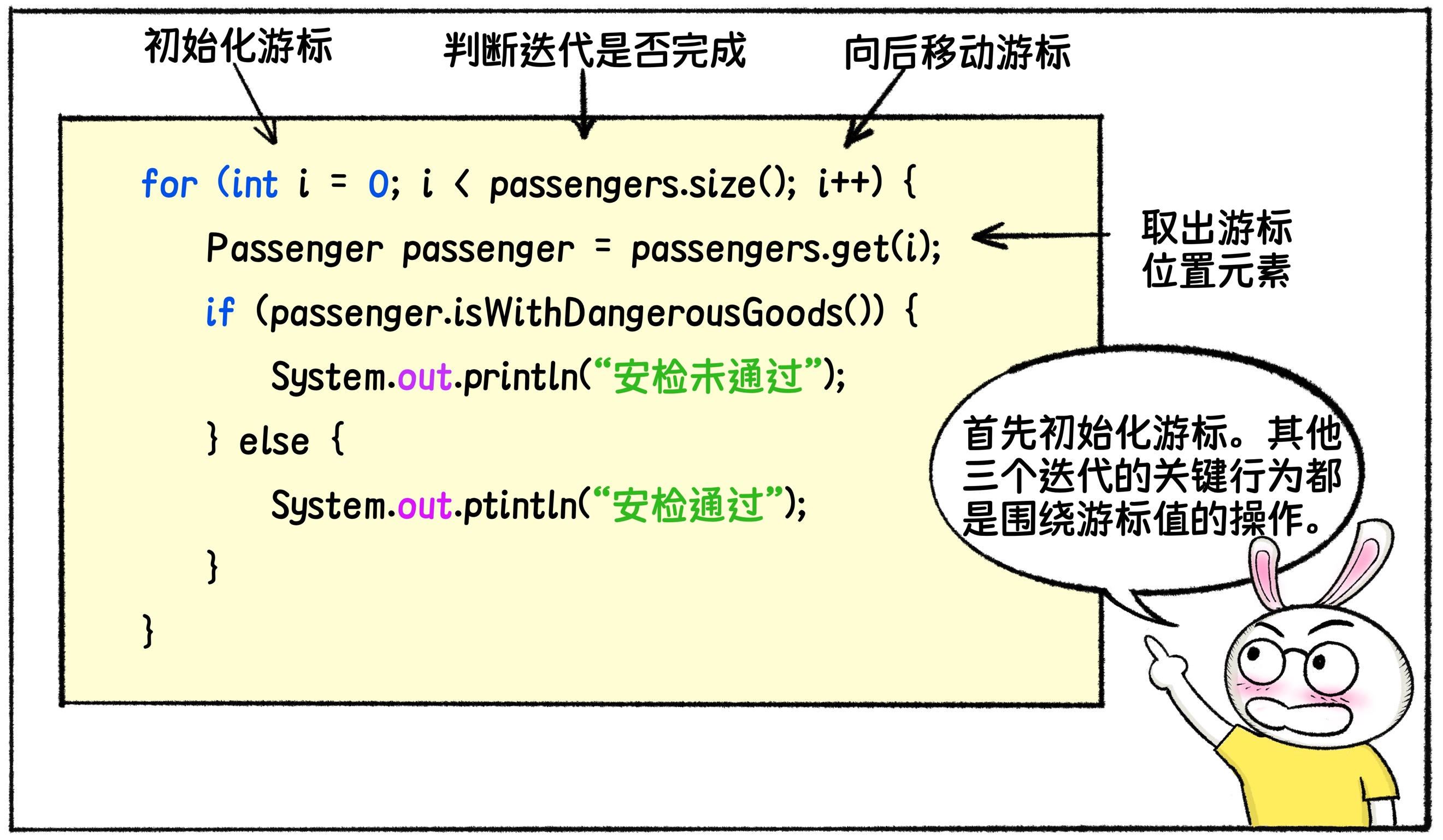
熊小猫：怪不得你以前会出错，原来你还在用最原始的方法写集合迭代。使用迭代器会更好，避免犯低级错误。

兔小白：我一直用for循环迭代集合呀！你说的迭代器怎么用？

## 19.3 迭代不只有for循环

熊小猫：从面向对象的角度，迭代器是被抽象出来专门负责迭代集合的类。其实你代码中的for循环就实现了迭代器的功能，你来讲一下这段代码的实现吧。

兔小白：for循环为了实现迭代，确实做了不少事情。



熊小猫: 这个for循环中，不但有局部变量，还有多种行为逻辑，配合起来才能完成迭代。你能写的如此顺手，有赖于Java的程序设计，使得用for写迭代既简洁又方便。但是，你每写一次这样的for循环，就相当于重复实现了一次迭代器。重复是软件开发中的大忌。而且如你所说，一不小心还可能写错。

兔小白：回想一下，我以前可真没少写这样的for循环，相当于我在一次又一次的写迭代器。那我确实不如写一个可以复用的迭代器，避免重复工作，还不会出错。

熊小猫：没错，看到重复就要想到提炼。迭代器其实也是一种设计模式，就叫做迭代器模式。下面你来参考for循环的实现，写一个List的迭代器吧！

兔小白：没问题，我写过那么多for循环，迭代的逻辑我熟悉。

10分钟后，兔小白完成了迭代器的开发。

兔小白：for循环中的i变量起到游标的作用，指示迭代的位置。迭代涉及三个关键操作。

1. 移动游标
2. 判断是否迭代完成
3. 取出当前游标位置的元素。

我为迭代器接口定义了3个相对应的方法：next、isDone、getCurrentItem。

public interface Iterator<E> {  
    void next();  
    E getCurrentItem();  
    boolean isDone();  
}

ListIterator实现Iterator接口，cursor属性为游标，指示迭代位置。

public class ListIterator<E> implements Iterator{  
    private int cursor;  
    private List<E> list;  
​  
    public ListIterator(List<E> list) {  
        this.list = list;  
   }  
​  
    public void next() {  
        cursor++;  
   }  
​  
    public E getCurrentItem() {  
        return list.get(cursor);  
   }  
​  
    public boolean isDone() {  
        return cursor == list.size();  
   }  
}

客户端代码如下。

List<Passenger> passengers = new ArrayList<>();  
passengers.add(new Passenger(false));  
passengers.add(new Passenger(false));  
passengers.add(new Passenger(true));  
passengers.add(new Passenger(false));  
​  
ListIterator<Passenger> listIterator = new ListIterator<>(passengers);

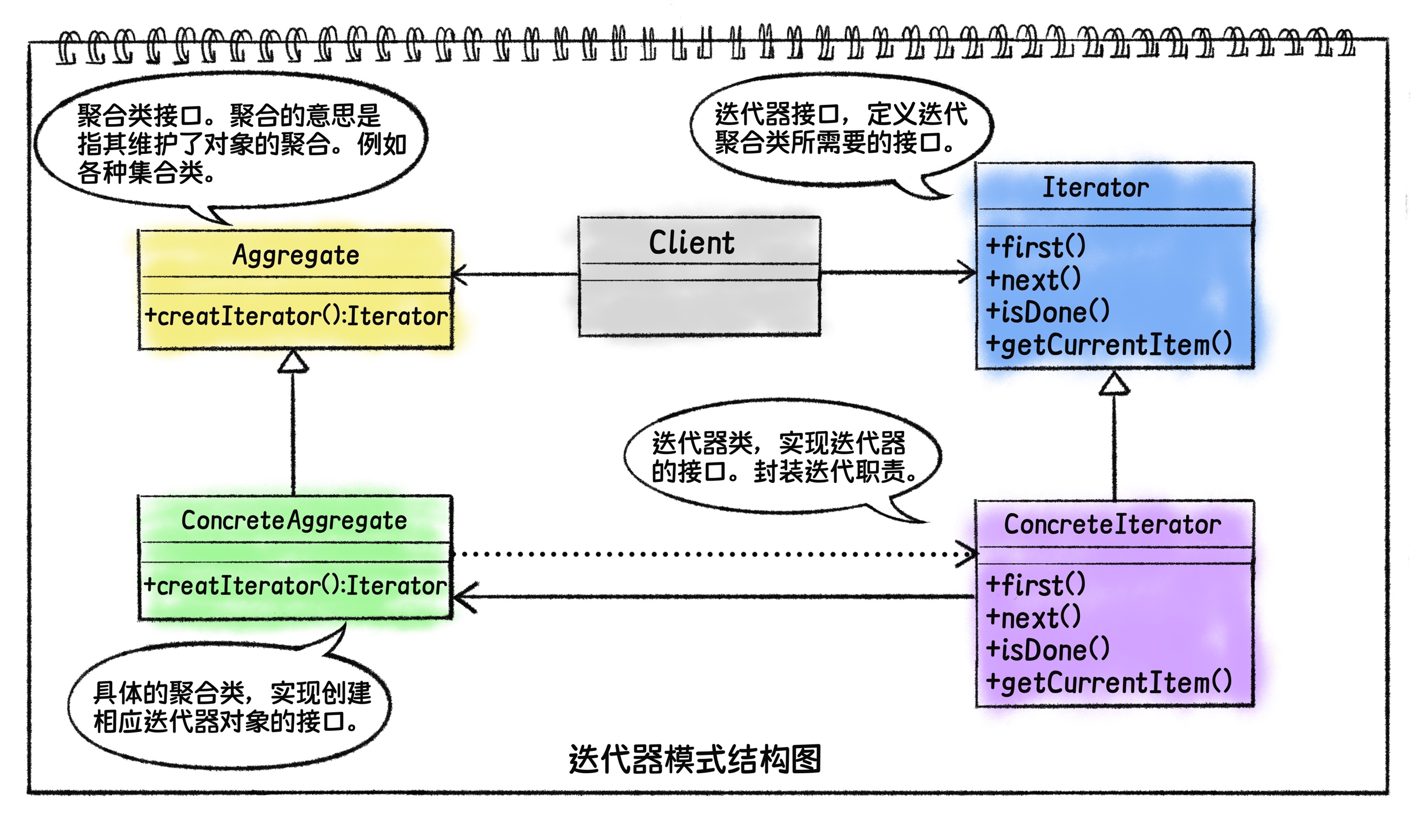
while (!listIterator.isDone()){  
  Passenger passenger = listIterator.getCurrentItem();  
​  
  if(passenger.isWithDangerousGoods()){  
    System.out.println("安检未通过");  
 }else{  
    System.out.println("安检通过");  
 }  
​  
  listIterator.next();  
}

兔小白：有了这个迭代器，以后就不用担心for循环写错了。

熊小猫：迭代器带来的收益可不止于此。下面我给你详细讲讲迭代器模式。

## 19.4 详解迭代器模式

熊小猫：我们先来看看迭代器模式结构图。



Aggregate是聚合的接口，例如Java中的List接口。ConcreteAggregate是实现接口的聚合类，例如Java中的ArrayList。Iterator是迭代器接口，你的程序中也有同名接口。它的实现是ConcreteIterator类，对应你程序中的ListIterator类。

Aggregate接口中定义的createIterator方法返回它自身集合类型对应的Iterator实例。

Iterator接口定义了4个方法，其中3个你已经用过，此外还有一个first方法。这4个方法的职责如下。

1. first方法用于初始化或者重置迭代器游标，使游标指向集合中的第一个元素。
2. next方法用于将迭代器游标向后移动一位。
3. isDone方法用于判断游标是否已经移过集合中最后一个元素，判断迭代是否结束。
4. getCurrentItem方法用于返回迭代器游标当前所处位置的元素。

兔小白：我没有用到first方法，也没什么问题呀！

熊小猫：这是因为你的迭代器默认将游标初始位置设置在集合中的第一个元素。另外，你的程序也不需要重置迭代器，用不到first方法。

迭代器的适用场景没什么好说的，需要迭代集合中每一个元素的场景，都适合使用迭代器模式。但是在迭代处理逻辑中，如需获取元素的位置，迭代器无法直接提供。需要通过计数变量等方式实现。当然也可以用最原始的for循环实现，直接取得游标值。

迭代器还有一个重要的知识点——外部迭代器和内部迭代器。内、外部指的是迭代发生在迭代器内部还是外部。练习的例子中，迭代发生在客户端代码里，所以你写的迭代器是外部迭代器。内部迭代器接收客户端提交的一个待执行操作，然后在迭代器内部完成迭代，对每一个元素实施该操作。

兔小白：看来我的代码还有提升的空间，第一个是迭代器的实例应该由集合类创建并返回。另外我再想想如何实现内部迭代器。

熊小猫：你这种追求技术卓越的精神值的表扬！不过......其实Java语言已经为开发者实现了迭代器，而且兼备内、外部迭代器的特性，可以直接拿来使用。

兔小白：啊！那你还让我自己开发！

熊小猫：自己动手才印象深刻，多动动手没坏处。迭代器由于使用太过广泛，已经成为了一种基础工具，大多数高级编程语言都会提供，不再需要自己开发。

兔小白：那我也不用再写内部迭代器了，研究一下Java提供的迭代器就好啦。

## 19.5 浅析Java中的迭代器

熊小猫：Java在1.2版本中定义了Iterator接口。

public interface Iterator<E> {  
    boolean hasNext();  
      
    E next();  
      
    default void remove() {  
        throw new UnsupportedOperationException("remove");  
   }  
      
    default void forEachRemaining(Consumer<? super E> action) {  
        Objects.requireNonNull(action);  
        while (hasNext())  
            action.accept(next());  
   }  
}

Java提供的Iterator接口在迭代器模式的基础上做了改进。hasNext方法对应迭代器模式中的isDone。next方法将迭代器模式中的next和getCurrentItem方法合二为一，实现将游标向后移动一位，并取出游标位置的元素[[1]](#footnote-1)1。remove方法用于安全移除处于迭代中的集合内的元素，它是一个可选的方法。forEachRemaining方法用于内部迭代，它接收Consumer类型的函数，然后将该函数应用到每一个元素之上。Consumer属于函数式编程的范畴，今天我就不展开说了。

几乎Java中的每一个集合类都在自己内部实现了对应的迭代器。开发者并不需要自己去编写迭代器。我把安检的例子改为使用Java提供的迭代器来实现。

List<Passenger> passengers = new ArrayList<>();  
passengers.add(new Passenger(false));  
passengers.add(new Passenger(false));  
passengers.add(new Passenger(true));  
passengers.add(new Passenger(false));  
​  
Iterator<Passenger> iterator = passengers.iterator();  
​  
while ((iterator.hasNext())){  
  Passenger passenger = iterator.next();  
  if(passenger.isWithDangerousGoods()){  
    System.out.println("安检未通过");  
 }else{  
    System.out.println("安检通过");  
 }  
}

此外，Java还提供了更方便的foreach语法，开发者可以继续用for语句来实现迭代。上面的while可以按如下改写。foreach的执行过程其实和直接用Iterator是一样的。

for(Passenger passenger:passengers){  
  if(passenger.isWithDangerousGoods()){  
    System.out.println("安检未通过");  
 }else{  
    System.out.println("安检通过");  
 }  
}

兔小白：今天学了不少迭代的方法呀！

熊小猫：还有内部迭代呢。使用内部迭代前，需要先定义一个Consumer类型的函数。可以在客户端代码中定义如下方法。

private static Consumer<Passenger> checkLanguage() {  
  return (passenger) -> {  
    if(passenger.isWithDangerousGoods()){  
      System.out.println("安检未通过");  
   }else{  
      System.out.println("安检通过");  
   }  
 };  
}

客户端代码不需要再写迭代的过程，只需要将该Consumer函数传递给迭代器，调用forEachRemaining方法。

//省略passengers构造过程  
Iterator<Passenger> iterator = passengers.iterator();  
iterator.forEachRemaining(checkLanguage());

这还不算完，还有更过分的“内部迭代”——不使用迭代器，直接在集合类的内部实现迭代。实现方式和forEachRemaining方法类似。大多数集合类都实现了Collection接口，而Collection接口继承自Iterable接口。Iterable接口中定义了一个名为forEach方法，代码如下。

default void forEach(Consumer<? super T> action) {  
        Objects.requireNonNull(action);  
        for (T t : this) {  
            action.accept(t);  
       }  
}

forEach方法同样接受一个Consumer类型的函数，将该函数作用于集合中的每一个元素。实现Iterable的集合类可以直接使用forEach方法执行内部迭代，不需要通过迭代器。

List<Passenger> passengers = new ArrayList<>();  
passengers.add(new Passenger(false));  
passengers.add(new Passenger(false));  
passengers.add(new Passenger(true));  
passengers.add(new Passenger(false));  
​  
passengers.forEach(checkLanguage());

兔小白：今天接触的信息有点多，让我捋一捋迭代的实现方式。

1. 使用基础for循环。最基础的迭代方式。
2. 使用Iterator。通过集合对象获取Iterator对象，对集合进行迭代。
3. 使用Foreach语法。可以看作是Iterator的语法糖，底层还是使用Iterator实现。
4. 使用Iterator的forEachRemaining方法。Iterator内部迭代，需要定义好处理逻辑的Consumer函数。
5. 使用集合的forEach方法。直接使用集合对象进行内部迭代，同样需要先定义好处理逻辑的Consumer函数。

熊小猫：这几种方式，我比较推荐使用Foreach和集合类的forEach方法。Foreach相比直接使用Iterator更为简便。使用集合类的forEach方法，不需要声明迭代器，也不需要写循环体，但需要了解函数式编程的基础。

基础for循环，虽然麻烦一点，但可以方便的获取游标位置。直接使用Iterator可以提供额外的功能，比如说remove方法。当用Iterator迭代几个元素之后，需要对后面的元素执行另外一种操作时，可以使用forEachRemaining。

总之，尽量不要自己去实现Iterator，这是永远排在最后面的选项。

1. 1 不同集合类对Iterator接口的实现略有区别，但思路类似。 [↑](#footnote-ref-1)