话说自从灰太狼抓到羊后，他们一家三口就过着幸福美满的生活。随着时间的流逝，灰太狼渐渐老去并挂掉，临死前他将自己的抓羊秘技传给了已经成年的小灰灰，并留下一件狼皮和一件羊皮。小灰灰也承担起了抓羊的重任。

        某一夜，北风呼呼的刮，雪花飘飘洒洒，小灰灰独自一狼上了梁山，混进了羊群。不幸的是，他在抓羊的时候被猎人撞见，只听见一声枪响，小灰灰他受了重伤，但他侥幸逃脱了，救他的是一只羊，从此他们约定三生，互诉着衷肠……（[汤潮](http://www.1ting.com/singer/6e/singer_2909.html" \t "http://blog.csdn.net/crayonyi/article/details/_blank)的歌声在夜空中响起）

        小灰灰和羊相爱了，但是在梁山上，狼只能一直披着羊皮生活，但是他还是很开心。直到有一天，小灰灰愁眉苦脸，像是有心事的样子。

        羊看见了，于是问小灰灰：“亲爱的，你怎么了？”

        小灰灰说：“我想我的妈妈了，自从上次离开家后，我都没有回去看过她。妈妈老了，不知道她现在过得怎么样，我很担心”。

        羊说：“亲爱的，别担心，明天我就陪你回去看你妈妈。”

        狼说：“可是，那可是狼国啊，对你太危险了。”

        羊说：“你爸爸临死前不是给了你一张羊皮和一张狼皮吗，交给我办，你能扮成羊，那我也能扮成狼”。经过一夜，羊改造成了一个新型的法宝——狼羊双向适配器。她把羊皮和狼皮缝在一起，正面羊皮，翻过来后就是狼皮，这样一件衣服，既能扮狼又能扮羊，非常的方便。为了安全起见，狼再次用程序模拟了一遍：

|  |
| --- |
| /\*\*   \* 一个双向适配器。可以把一匹狼适配成一头羊，也可以把一头羊适配成一匹狼。   \* **@author** jingping.yi   \*   \*/  **public class** ATwoFaceAdapter **implements** ISheep, IWolf {      ISheep sheep = **null**;  **public** ATwoFaceAdapter(ISheep sheep){  **this**.sheep = sheep;      }      @Override  **public void** wolfLooks() {         // 假装自己是一匹狼         System.out.println("I am a wolf,too!");         // 隐藏自己的外貌         // sheep.sheepLooks();      }        @Override  **public void** eatMeat() {         // 本质是吃草         sheep.eatGrass();      }        IWolf wolf = **null**;  **public** ATwoFaceAdapter(IWolf wolf){  **this**.wolf = wolf;      }      @Override  **public void** sheepLooks() {         // 假装自己是一头羊         System.out.println("I am a sheep,too.");         // 隐藏自己的外貌         // wolf.wolfLooks();        }        @Override  **public void** eatGrass() {         // 本质是吃肉         wolf.eatMeat();      }    } |
| **package** com.crayon.adapter.wolfandsheep2face;  /\*\*   \* 双向适配器——可以有披着羊皮的狼，也可以有披着狼皮的羊。   \* **@author** jingping.yi   \*   \*/  **public class** Main {  **public static void** main(String[] args) {         // 把一匹狼适配成一头羊         ISheep AFakeSheep =  **new** ATwoFaceAdapter(**new** ARealWolf());         AFakeSheep.sheepLooks();         // AFakeSheep.eatGrass();           // 把一头羊适配成一匹狼         IWolf AFakeWolf = **new** ATwoFaceAdapter(**new** ARealSheep());         AFakeWolf.wolfLooks();         // AFakeWolf.eatMeat();      }  }  **// 输出**  I am a sheep,too.  I am a wolf,too! |

通过程序测试了这件无敌法宝——狼羊双向适配器很安全后，小灰灰终于答应带着妻子回去看红太狼了。也是因为有了这个双向适配器，他们自由的穿梭在狼的王国和羊的王国，为了河蟹社会贡献自己的一份力量。

适配器也可以实现双向的适配，前面我们讲的都是把Adaptee适配成为Target，其实也可以把Target适配成为Adaptee，也就是说这个适配器可以同时当作Target和Adaptee来使用。

继续前面讲述的例子，如果说由于某些原因，第一版和第二版会同时共存一段时间，比如第二版的应用还在不断调整中，也就是第二版还不够稳定。客户提出，希望在两版共存期间，主要还是在使用第一版，同时希望第一版的日志也能记入到数据库中，也就是客户虽然操作的接口是第一版的日志接口，界面也是第一版的界面，但是可以使用第二版的把日志记录到数据库的功能。

也就是说希望两版能实现双向的适配，结构如下图4.12所示：

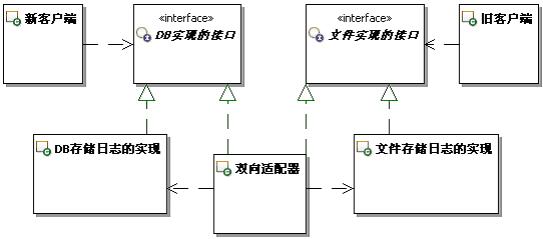


图4.12  双向适配器示意图

还是写点代码简单的示意一下，以利于大家理解。

这里只加了几个新的东西，一个就是DB存储日志的实现，前面的例子是没有的，因为直接被适配成使用文件存储日志的实现了；另外一个就是双向适配器，其实与把文件存储的方式适配成为DB实现的接口是一样的，只需要新加上把DB实现的功能适配成为文件实现的接口就好了。

（1）先看看DB存储日志的实现，为了简单，这里就不去真的实现和数据库交互了，示意一下，示例代码如下：

|  |
| --- |
| /\*\*   \* DB存储日志的实现，为了简单，这里就不去真的实现和数据库交互了，示意一下   \*/  public class LogDbOperate implements LogDbOperateApi{      public void createLog(LogModel lm) {         System.out.println("now in LogDbOperate createLog,lm="+lm);      }      public List<LogModel> getAllLog() {         System.out.println("now in LogDbOperate getAllLog");         return null;      }      public void removeLog(LogModel lm) {         System.out.println("now in LogDbOperate removeLog,lm="+lm);      }      public void updateLog(LogModel lm) {         System.out.println("now in LogDbOperate updateLog,lm="+lm);      }  } |

（2）然后看看新的适配器的实现

由于是双向的适配器，一个方向是：把新的DB实现的接口适配成为旧的文件操作需要的接口；另外一个方向是把旧的文件操作的接口适配成为新的DB实现需要的接口。示例代码如下：

|  |
| --- |
| /\*\*   \* 双向适配器对象   \*/  public class TwoDirectAdapter implements  LogDbOperateApi,LogFileOperateApi{      /\*\*       \* 持有需要被适配的文件存储日志的接口对象       \*/      private LogFileOperateApi fileLog;      /\*\*       \* 持有需要被适配的DB存储日志的接口对象       \*/      private LogDbOperateApi  dbLog;      /\*\*       \* 构造方法，传入需要被适配的对象       \* @param fileLog 需要被适配的文件存储日志的接口对象       \* @param dbLog 需要被适配的DB存储日志的接口对象       \*/      public TwoDirectAdapter(LogFileOperateApi fileLog  ,LogDbOperateApi dbLog) {         this.fileLog = fileLog;         this.dbLog = dbLog;      }  **/\*-----以下是把文件操作的方式适配成为DB实现方式的接口-----\*/**      public void createLog(LogModel lm) {         //1：先读取文件的内容         List<LogModel> list = fileLog.readLogFile();         //2：加入新的日志对象         list.add(lm);         //3：重新写入文件         fileLog.writeLogFile(list);      }      public List<LogModel> getAllLog() {         return fileLog.readLogFile();      }      public void removeLog(LogModel lm) {         //1：先读取文件的内容         List<LogModel> list = fileLog.readLogFile();         //2：删除相应的日志对象         list.remove(lm);         //3：重新写入文件         fileLog.writeLogFile(list);      }      public void updateLog(LogModel lm) {         //1：先读取文件的内容         List<LogModel> list = fileLog.readLogFile();         //2：修改相应的日志对象         for(int i=0;i<list.size();i++){             if(list.get(i).getLogId().equals(lm.getLogId())){                list.set(i, lm);                break;             }         }         //3：重新写入文件         fileLog.writeLogFile(list);      }  **/\*-----以下是把DB操作的方式适配成为文件实现方式的接口-----\*/**      public List<LogModel> readLogFile() {         return dbLog.getAllLog();      }      public void writeLogFile(List<LogModel> list) {         //1：最简单的实现思路，先删除数据库中的数据         //2：然后循环把现在的数据加入到数据库中          for(LogModel lm : list){             dbLog.createLog(lm);         }      }  } |

（3）看看如何使用这个双向适配器，测试一下，示例代码如下：

|  |
| --- |
| public class Client {      public static void main(String[] args) {         //准备日志内容，也就是测试的数据         LogModel lm1 = new LogModel();         lm1.setLogId("001");         lm1.setOperateUser("admin");         lm1.setOperateTime("2010-03-0210:08:18");         lm1.setLogContent("这是一个测试");         List<LogModel> list = new ArrayList<LogModel>();         list.add(lm1);           //创建操作日志文件的对象         LogFileOperateApi fileLogApi = new LogFileOperate("");         LogDbOperateApi dbLogApi = new LogDbOperate();           //创建经过双向适配后的操作日志的接口对象         LogFileOperateApi fileLogApi2 =  new TwoDirectAdapter(fileLogApi,dbLogApi);         LogDbOperateApi dbLogApi2 =  new TwoDirectAdapter(fileLogApi,dbLogApi);           //先测试从文件操作适配到第二版，  //虽然调用的是第二版的接口，其实是文件操作在实现         dbLogApi2.createLog(lm1);         List<LogModel> allLog = dbLogApi2.getAllLog();         System.out.println("allLog="+allLog);           //再测试从数据库存储适配成第一版的接口，  //也就是调用第一版的接口，其实是数据实现         fileLogApi2.writeLogFile(list);         fileLogApi2.readLogFile();      }  } |

去运行一下，看看结果，体会一下双向适配器。