南京大學

本 科 毕 业 论 文（设计）

院　系 软件学院

题　目 基于 Web2.0 的破产管理系统的设计与实现

学生姓名 陈曼曼 学　号 071251011

年　级 四年级 专　业 软件学院

指导教师 刘海涛 职　称 讲师

论文提交日期 2011 年 5 月 30 日

摘　要

对于 eBay 这样的全球电子商务平台，拥有很多的卖家，而且来自不同的国家。当某个卖家由于某些原因（通常是经济遭遇困难）破产时，根据不同的法律，ebay 会对其采取不同的措施，但是 ebay 都必须要继续跟踪记录这些破产人的相关信息。由于数据量较大，通过手动在 Hardcopy Tool 系统中查询相关的破产证明文件和手动操作 Excel 表格的传统工作方式显得效率低下且易于出错，多个工作人员共享 excel 文件就会导致数据的丢失和冲突。所以需要开发一个系统来帮助工作人员完成这些任务。

该系统叫 Bankruptcy Tool, 作为一个办公自动化系统，主要目的就是为所有的破产 case 提供一个灵活的集中管理的平台，提高 eBay 工作人员的工作效率。它以页面的形式帮助工作人员查看 Case 信息以及收集 Case 数据。此外系统需要良好的可扩展性和可修改性，以便在后续的版本中增加新的功能。

Bankruptcy 系统最终会部署在 WIMP 运行环境上。系统采用 MVC 的设计思想，使用 PHP 实现。为了方便页面开发，使用 HTMLTree API，即将所有 HTML 的标签视为对象，标签的嵌套关系看成对象的父子关系，最终把一个 HTML 页面抽象成一颗树，通过遍历这颗树，打印树的结点与属性，可以把树还原成 HTML 页面。

Bankruptcy 系统在 Web 表现层大量使用 Ajax 技术创建更友好的界面。为了方便在后续版本中增加新的功能，在 Case 信息显示模块使用了 CaseFacet 接口，在 Case 视图中的各种信息，都实现了 CaseFacet 接口。为了加快页面的访问速度，提高工作人员的工作效率，系统还采用了 Memcached 缓存技术。

本文主要从系统的背景，需求分析，系统的架构，设计以及实现来详细阐述整个项目。

关键词：bankruptcy 系统，PHP，MVC，AJAX

Abstract

eBay, a global e-commerce platform, has a lot of sellers and buyers and they come from different countries. When some sellers for some reasons like financial difficulty go bankrupt, eBay will take different measures on them according to different legal, but eBay must keep track of sellers' information that under bankrupt. Staffs usually search bankruptcy documents in Hardcopy Tool and read or modify excel documents manually, but this is inefficient and proves to be prone to error due to a large amount of data, in addition, multiple shared excel files results in data loss and conflicts. So it is required to develop a tool to help staffs to deal with these tasks.

This system is called Bankruptcy Tool, as office automation, the main purpose of the system is to provide a flexible platform to manage bankruptcy cases. It shows case information to staffs in the page. And it can also help workers to collect case data. Besides, the system should have a good scalability and modification so that we can add new features easily.

The runtime environment of the system is WIMP A Windows box running IIS with a database in MySQL with code running in PHP. MVC design pattern is used realized by PHP. It developsHTMLTree API in order to facilitate the page, the main idea is regarding all HTML tags as object, tag nested relation is regarded as parent -child relationship. Ultimately an HTML page is abstract into a tree. Print the HTML page by going through all tree nodes and attributes.

Bankruptcy system uses PHP combing with AJAX to show friendly interface. It uses caseFacet interface to implement display case information dynamically. Each kind of case information implements the caseFacet interface.Specially, Memcached cache technology is used to speed up accessing page and improve efficiency.

It describes the background of the system, requirements analysis and system architecture, design and implement in details in this paper.

Keywords：bankruptcy system, PHP,AJAX, MVC

目　录

图目录

第一章 概述/绪论

1.1 项目背景

破产是一种法律程序宣告债务人无力偿付债务及其后的一连锁还款予债权人的过程。不同国家的破产法律有很大的差异。对于 eBay 这样的全球电子商务平台，拥有很多的卖家，而且来自不同的国家。当卖家在 eBay 平台上出售商品的时候，eBay 会向卖家收取成交费（成交价的 2-5%不等），此时 eBay 就成为了债权人。当某个卖家由于某些原因（通常是经济遭遇困难）破产时，破产保护的相关文件规定他必须通知他的所有债权人。由于不同国家的破产法律不同，根据国家相应的法律规定，会采取不同的措施。在美国，这意味着 eBay 不允许关闭破产人的账户并且 eBay 必须停止向债务人索取债务,但是 eBay 必须要继续跟踪记录这些破产人的相关信息。

在过去的很多年,eBay 的工作人员一直是使用 hardcopy tool 和很多 excel 文件来跟踪 eBay 用户的破产案例（下面文中统一称为 Case）。Hardcopy tool 是一个文件管理系统，主要是用来管理传真或者邮寄过来的破产通知的文件，然后系统通过扫描这些文件，将扫描得到的图片存储到系统中，并存储了与这些文件相关的 Case 的信息。当工作人员使用这套工作方案的时候，需要在 excel 文件中查询破产 Case 的相关信息又需要在 hardcopy tool 中查找相关的文件，由于各个国家相关的破产法律措施的不同，对不同的国家的破产 Case 有不同的处理，再加上一个破产 case 可能关联很多账户，所以工作效率很低。而且所有的成员共享 excel 文件，可能导致数据的丢失和冲突。对于新的工作人员，熟悉这个工作流程需要一段时间。所以 eBay 决定开发一个系统来取代传统的工作方式，通过集成 Hardcopy tool 中的信息为工作人员提供一个直观，快捷，方便的公共平台跟踪破产 Case。

1.2 背景词汇

- Case（案件）：一个 ebay 的用户破产即为一个 Case,但是一个用户可能在 ebay 中注册了多个账号。
- Payment Plan(付款计划)：是指被告知破产用户将有个付款或者说还款计划。
- Suspected Bankruptcy（疑似破产）：是指虽已破产但是公司没有从政府或者律师那边收到任何的证明材料。

- Bankruptcy（破产）：是指公司已经收到破产证明。
- Situation（情况）: Case 也有很多种情况，比如 Bankruptcy, Payment Plan, Suspected Bankruptcy and Restore from Bankruptcy 都是 Situation 的例子。
- Case Type(Case 类型)：一个 Case 类型的处理方式相同。由于不同的国家的破产法律不同，所以对 case 的处理方式也不同。因而 Case Type 是由 Situation 和 Case 所处的国家决定的。
- Case Owner(Case 的拥有者)：即负责对 Case 进行跟踪记录信息，并处理该 Case 的人。

1.3 论文组织结构

第一章叙述了 Bankruptcy 系统的背景和系统的概要介绍，以及与同类产品的比较。

第二章依次介绍了 Bankruptcy 系统实现中用到的 PHP、AJAX 技术、MVC 设计模式以及 WIMP 运行环境。

第三章从用例，功能等方面详细分析了 Bankruptcy 系统并提出了概要设计方案。

第四章根据第三章对 Bankruptcy 系统的分析以及概要设计，从数据库，接口，页面三个方便阐述了系统的详细设计。

第五章阐述了 Bankruptcy 系统的具体实现，尤其系统实现中的难点进行了详细介绍。

第六章总结了本文所作的工作，并提出了不足和需要改进的地方。

第二章 相关技术概述

2.1 PHP 技术

PHP 是英文超级文本预处理语言 Hypertext Preprocessor 的缩写。PHP 是创建动态交互性站点的强有力的服务器端脚本语言，是一种在服务器端执行的嵌入 HTML 文档的脚本语言，语言的风格类似于 C 语言，主要用途是处理动态网页[1]。PHP 最早由 Rasmus Lerdorf 在 1995 年发明，而现在 PHP 的标准由 PHP Group 和开放源代码社区维护。PHP 以 PHP License 作为许可协议，不过因为这个协议限制了 PHP 名称的使用，所以和开放源代码许可协议 GPL 不兼容[2]。PHP 的应用范围相当广泛，尤其是在网页程序的开发上。一般来说 PHP 大多运行在网页服务器上，通过运行 PHP 代码来产生用户浏览的网页。PHP 可以在多数的服务器和操作系统上运行，而且使用 PHP 完全是免费的。

同时，对于像微软 ASP 这样的竞争者来说，PHP 无疑是另一种高效率的选项。

从 PHP 语言方面来，PHP 独特的语法混合了 C、Java、Perl 以及 PHP 自创新的语法。它可以比 CGI 或者 Perl 更快速的执行动态网页。用 PHP 做出的动态页面与其他的编程语言相比，PHP 是将程序嵌入到 HTML 文档中去执行，执行效率比完全生成 HTML 标记的 CGI 要高许多；PHP 还可以执行编译后代码，编译可以达到加密和优化代码运行，使代码运行更快。PHP 具有非常强大的功能，所有的 CGI 的功能 PHP 都能实现，而且支持几乎所有流行的数据库以及操作系统。

2.2 AJAX 技术

AJAX 即“Asynchronous JavaScript and XML”（异步 JavaScript 和 XML），AJAX 并非缩写词，而是由 Jesse James Garrett 创造的名词，是指一种创建交互式网页应用的网页开发技术[3]。AJAX 是一种在 2005 年由 Google 推广开来的编程模式，AJAX 不是一种新的编程语言，而是一种使用现有标准的新方法，用于创建更好更快以及交互性更强的 Web 应用程序。AJAX[4]即：

运用 XHTML+CSS 来表达信息；

运用 JavaScript 操作 DOM（Document Object Model）运行动态效果；

运用 XML 和 XSLT 进行数据交换及操作；

运用 XMLHttpRequest 为 Agent 与网页服务器进行异步数据交换；

运用 JavaScript 技术实现。

传统的 Web 应用允许用户端填写表单（form），当提交表单时就向 Web 服务器发送一个请求。服务器接收并处理传来的表单，然后送回一个新的网页，但这个做法浪费了许多带宽，因为在前后两个页面中的大部分 HTML 码往往是相同的。由于每次应用的沟通都需要向服务器发送请求，应用的回应时间就依赖于服务器的回应时间。这导致了用户界面的回应比本机应用慢得多。

与此不同，通过 AJAX，您的 JavaScript 可使用 JavaScript 的 XMLHttpRequest 对象来直接与服务器进行通信。通过这个对象，您的 JavaScript 可在不重载页面的情况与 Web 服务器交换数据。AJAX 在浏览器与 Web 服务器之间使用异步数据传输（HTTP 请求），这样就可使网页从服务器请求少量的信息，而不是整个页面。AJAX 可使因特网应用程序更小、更快，更友好。AJAX 是一种独立于 Web 服务器软件的浏览器技术。

2.3 Memcached 缓存技术

Memcached 是一个高性能的分布式内存对象缓存系统，用于动态 Web 应用以减轻数据库负载。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提供动态、数据库驱动网站的速度。Memcached 基于一个存储键/值对的 hashmap。其守护进程（daemon ）是用 C 写的，但是客户端可以用任何语言来编写，并通过 memcached 协议与守护进程通信[6]。但是它并不提供冗余（例如，复制其 hashmap 条目）；当某个服务器 S 停止运行或崩溃了，所有存放在 S 上的键/值对都将丢失。

Memcached 由 Danga Interactive 开发，其最新版本发布于 2010 年，作者为 Anatoly Vorobey 和 Brad Fitzpatrick[7]。用于提升 LiveJournal.com 访问速度的。LJ 每秒动态页面访问量几千次，用户 700 万。Memcached 将数据库负载大幅度降低，更好的分配资源，更快速访问。

2.4 WIMP 运行环境

WIMP 是 Windows+IIS+Mysql+PHP 的简称。

IIS 是 Internet Information Services 的缩写[8]，即互联网信息服务，是一个 World Wide Web server，是由微软公司提供的基于运行 Microsoft Windows 的互联网基本服务。Gopher server 和 FTP server 全部包容在里面。 IIS 意味着你能发布网页，并且有 ASP（Active Server Pages）、JAVA、VBscript 产生页面，有着一些扩展功能。IIS 是在 Windows 操作系统平台下开发的，这也限制了它只能在这种操作系统下运行。

WIMP 就是说有 MySQL 数据库，代码运行在 PHP 上的，有 IIS 的 windows

环境。

2.5 MVC 设计模式

MVC 模式是软件工程中的一种架构模式，把软件系统分为三个基本部分，模型（Model）、视图（View）和控制器（Controller）[9]。MVC 模式的目的就是实现 Web 系统的职能分工。Model 层实现系统中的业务逻辑，通常可以用 JavaBean 或 EJB 来实现。View 层用于与用户的交互，通常用 JSP 来实现。Controller 层是 Model 与 View 之间沟通的桥梁，它可以分派用户的请求并选择恰当的视图以用于显示，同时它也可以解释用户的输入并将它们映射为模型层可执行的操作。

模型（Model）：“数据模型”（Model）用于封装与应用程序的业务逻辑相关的数据以及对数据的处理方法。“模型”有对数据直接访问的权力，例如对数据库的访问。“模型”不依赖“视图”和“控制器”，也就是说，模型不关心它会被如何显示或是如何被操作。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

视图（View）：视图是用户看到并与之交互的界面。视图层能够实现数据有目的的显示（理论上，这不是必需的）。在视图中一般没有程序上的逻辑。

控制器（Controller）：控制器接受用户的输入并调用模型和视图去完成用户的需求。控制器起到不同层面间的组织作用，用于控制应用程序的流程。当单击 Web 页面中的超链接和发送 HTML 表单时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

2.6 本章小结

这一章主要介绍了开发 Bankruptcy 系统所使用的相关技术和架构模式。首先，Web 界面部分采用 PHP 和 AJAX 技术，在数据存取部分，使用了 Memcached 缓存技术。给用户提供更好更快的界面交互体验。接着本章还介绍了系统的 WIMP 运行环境，WIMP 是由 eBay 的 TAO Team 部署和维护的。整个系统采用 MVC 的设计模式，使视图层和业务逻辑层分离，这样就允许更改视图层代码而不用重新编译模型和控制器代码。因此，采用 MVC 的设计模式可以有效的降低系统的耦合度，提高系统的可修改性和扩展性。

第三章 概述系统分析与概要设计

3.1 系统概述

图 3-1 是一个工作人员完整的处理 Case 的流程图，首先用户登录，打开到

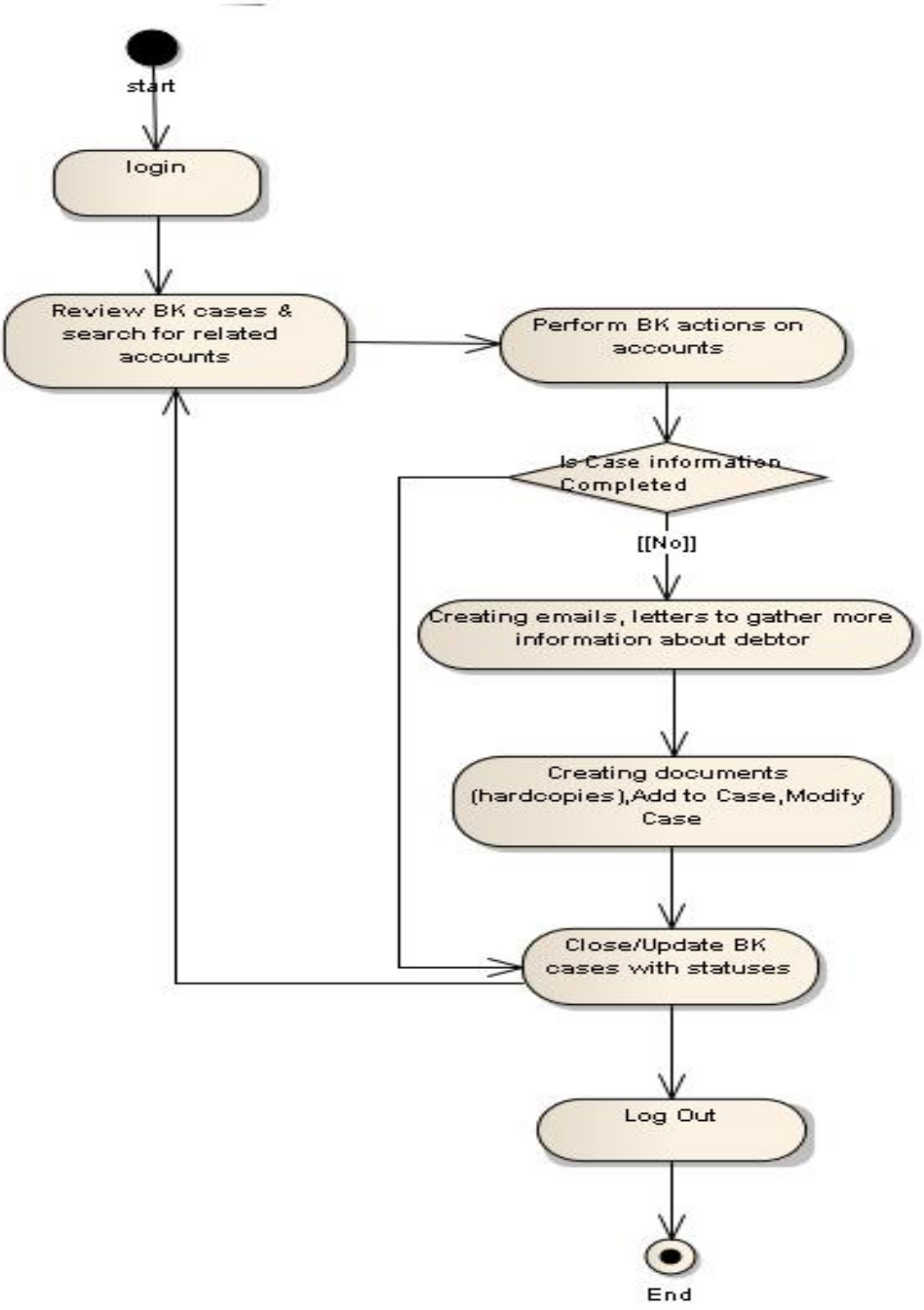

图 3- 1 工作人员处理流程图

自己需要处理的 Case 的页面，查看 Case 的详细信息，查看 Case 的所关联的账户，根据 Case 所属的类型对账户进行相关的操作。如果发现 Case 的信息不完整，则会让 debtor 提供更多的信息，根据 debtor 提供的信息再进行修改，也有可能如果缺少 hardcopy,则讲 debtor 提供的 hardcopy 加入到 Hardcopy tool 系统中，再在 case 的信息中增加 hardcopy。工作人员处理最后，会将该 Case 的状态更新，如 Pending(需要更多信息)，Completed(case 已经处理完成)。

因此项目的名字叫 Bankruptcy Tool, 主要目的就是为所有的 Case 提供一个灵活的集中管理的平台，提高 Billing Operation Team 的工作人员的工作效率。

该系统基于上述的背景和 Case 的特征设计了一系列的业务服务。在系统的第一个正式版本中，包含以下功能:

1. 用户可以向系统中导入要处理的 Case,并在页面中将所有的需要处理的 Case 显示出来。
2. 用户可以选择某个单独的 Case 去处理，成为该 Case 的 Owner。当发现有两个 Case 其实属于同一个账户的时候，Case Owner 可以合并他们。
3. 允许用户查看 Case 的详情包括用户处理 Case 所需要的辅助数据。
4. 系统需要记录工作人员即系统用户的工作日志。
5. 系统用户可以搜索 Case。

该系统现在只需要针对英国、德国、法国和美国国家的破产，但是以后会有更多国家的 Case 已经更多 Situation 的 Case，所以系统必须可扩展以便支持更多的国家和 Situation。

为了方便在后续版本中加入新的业务功能（比如对 Case 账户的操作），所以系统在设计上必须具备良好的可扩展性和可修改性。

3.2 系统用例分析与设计

表 3-1 系统用户表

ebay 工作人员	普通的系统授权用户	系统普通用户
	Case Owner	负责跟踪 case 信息以及处理的用户

如表 3-1 所示，系统主要有两种用户角色：系统的用户即是系统的授权用户，也就是客服的工作人员，当非系统授权用户打开系统的时候，不能够查看系统的任何信息。系统的授权用户又根据 Case 分为普通的系统授权用户和 Case Owner。

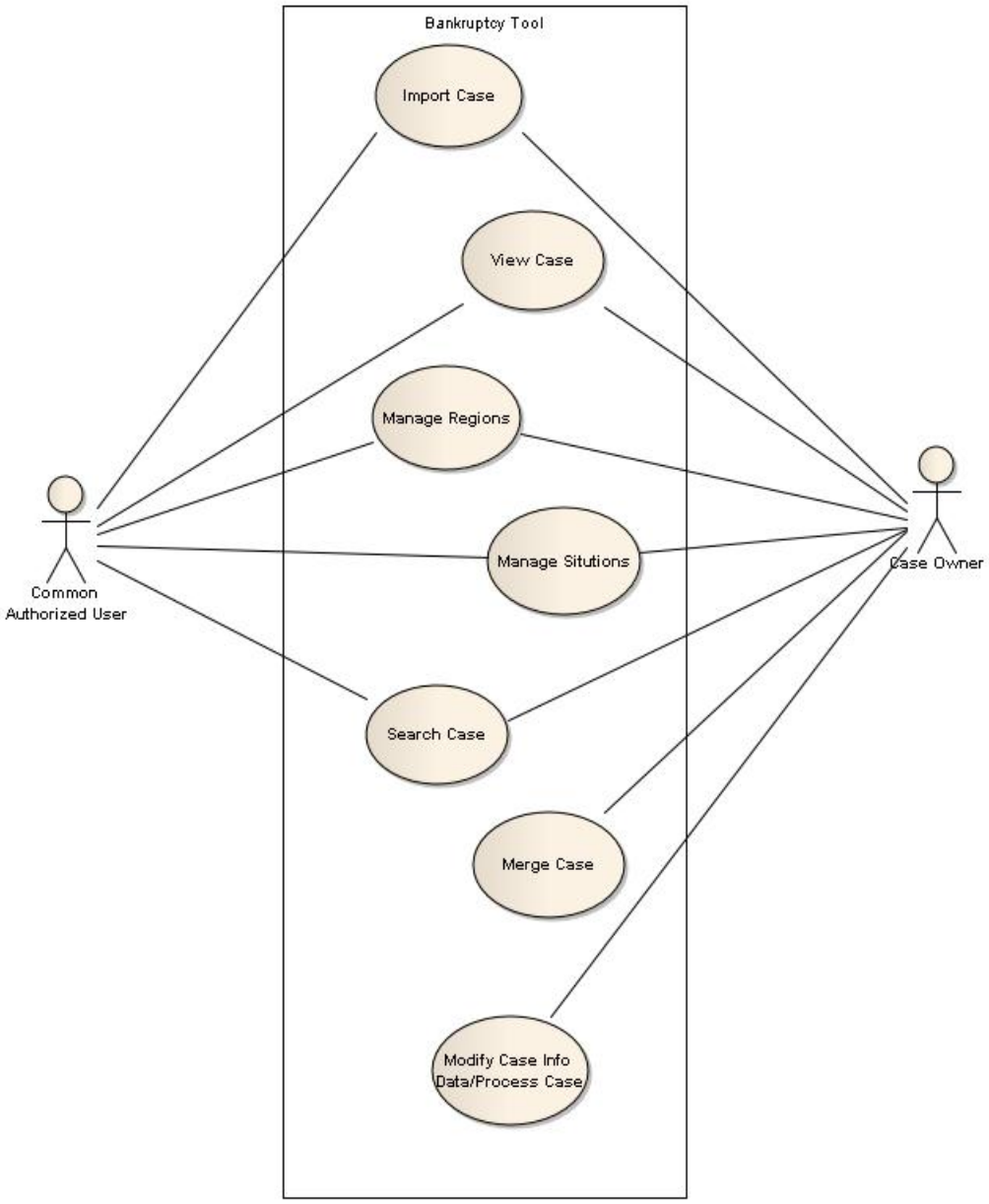

图 3- 2 系统用例图

如图 3-2 所示，系统的用户即 Billing Operation Team 的工作人员可以导入 Case 到系统中，查看 Case 的详细信息。为了更好的支持系统的可扩展性，系统的用户还可以管理 Region(国家)和 situation，也就是说对他们进行增加删除修改。为了方便用户快速的找到 Case,用户还可以在系统中搜索 Case。

而对于某个 Case Owner，作为这个 Case 的负责跟踪者和处理者，需要对 Case 有更高的权限，所以除了拥有跟上述普通的系统授权用户的权限之外，他还有权限对自己所拥有的 Case 的信息进行和合并 case。

3.3 系统功能分析

3.3.1 导入 Case

用户可以指定 Case Type 后通过以下三种方式将 Case 导入到系统中:

- 从账户标识符导入 Case

通过账户标识符的类型和账户的标识符创建 Case，只有当该账户存在且没有关联到其他的 Case 的时，系统可以成功创建一个 Case，此时，该账户则关联到新创建的 Case 上且为这个新 Case 的主账户，且将该账户的相关信息赋给 Case。

- 从 Hardcopy 导入 Case

根据 HardcopyID 导入 Case。由于一个 Hardcopy 是破产凭证，所有一个 Hardcopy 只能关联到一个 Case。只有当这个 Hardcopy 存在，并且没有关联到其他 Case 时，导入 Case 成功，并将 Hardcopy 的相关信息赋值给 Case。

- 从 Hardcopy 队列导入 Case

这里是通过一个 Hardcopy 队列的 ID 导入 Case,为这个 Hardcopy 队列中每个的 hardcopy 都新建一个 Case 中（与上述从 hardcopy 导入 Case 一样）。

3.3.2 查看 Case 视图

系统的用户可以查看 Case 的详细信息，Case 的信息主要包括：

Case Type：导入 Case 的时候就已经设置了 Case Type。

Local Status：Case 当前的状态。

Member Name，Member Address：Case 主账户人的信息，一般在导入 Case 的时候就已经将信息导入进来。

Court/District: 指 Case 所属的地区具体到哪个城市，由 Case Owner 设置。

Date Received:指收到破产文件的时间，也即破产文件扫描到 Hardcopy Tool 的时间。

Case Owner: 即处理 Case 的工作人员。

Power Seller：显示 Case 的账户中实力卖家。

Insolvency Type:破产类型，由 Case Owner 设置。

Hardcopy:显示用户破产文件凭证的具体信息，方便 Case Owner 查看。

Notes&Log: 显示 Case Owner 对此 Case 所做的标注和记录。

Case Account:即与 Case 相关的所有的 Account 的详细信息，同时也要将主要的 Account 标注出来。

除以上外，还有 case 合并的详细信息已经其他一些辅助信息。

Case 的每一个数据信息都是单独的一块，方便用户修改，为了用户查看方

便，对于数据量比较大的信息显示在单独的显示上。由于对不同 Case Type 的处理不同，所以不同 Case Type 所需要显示的 Case 的信息也不同，因而用户可以对每个类型的 Case 所需要显示的内容进行选择配置。

对于 Case Owner，不仅可以查看 Case 的信息，而且可以修改 Case 的信息。用户可以单独的修改上述 Case 中的信息。另外 Case Owner 有权限 unlock Case, 即将释放 Case, Case 的 Owner 会变为空，以便让其他用户来处理这个 Case。

其实工作人员大部分处理 Case 都不是在该系统上面的，本系统只是提供一个 Case 管理和 Case 信息数据收集的平台。当工作人员处理 Case 的时候在该系统上面查看 Case 的信息，如果发现有不完整的或者不正确的，则会修改 Case 的数据或者 Case 的状态。当处理完成的之后，Case 就不会出现在待处理的 Case 列表中。

3.3.3 配置 Case 视图

系统的用户可以根据各个类型的 Case 需求配置 Case 视图中需要显示的信息，同时用户可以设置各类信息显示的顺序。另外，用户还可以增删改该 Case 类型中的所有状态，设置这个类型所关联的 hardcopy 队列。

3.3.4 合并 Case

用户通过两个 Case 的 ID 即可将两个 Case 合并，但是合并的 Case 必须满足 Owner 相同，Case Type 相同且之前未被合并过，创建时间比较早的 Case 即为父 Case,另一个为子 Case,并将子 Case 标注为已合并。如果子 Case 的某些信息为空，则将父 Case 的信息赋值给子 Case,并将子 Case 的所有子孙设为其父 Case 的直系儿子，同样，将父 Case 的信息赋值给这些 Case。

3.3.5 案件备注

用户可以给每个 Case 做留言或者备注，这样方便以后工作查看，同时用户在可以在该页面查看到该 Case 所有的状态的改变记录，各种信息的数据的变化记录以及该 Case 的 Owner 变化的记录，方便工作人员查看，也体现了系统的安全性

3.3.6 搜索 Case

为了方便用户快速定位到 Case，系统提供使用关键字搜索 Case 的功能。在 Case 的详细信息中查找该关键字，将满足条件的 Case 返回给用户。

系统的第一个正式版本中只支持简单的搜索功能，即 Case 中的信息需要完全匹配用户输入的关键字。

3.3.7 管理 Case 辅助信息

Region（国家），Situation（情况）是每个 Case 必不可少的信息，为了规范统一和以后的可扩展，而且方便用户使用，当用户需要修改 Case 的这类信息时候，只要从已存在的列表中选择一个。

系统为用户提供对 Region 和 Situation 的管理功能，即可以增加,修改 Region, Situation。

Case Type 是由 Region 和 Situation 组成的，用户通过可以选择 Region 和 Situation 来新建一个 Case Type,如果 Case Type 暂时不需要使用，将其设置为 Inactive，如果用户想让该 Case Type 不对外可见，则可以将 Case Type 设置为私有的。一个 Case Type 的属性对该类型所有的 Case 都适用，同一个 Case 类型的 Case 的处理方式也是相同的。

除了上述的 Case 的数据，系统还需要提供其他一些辅助数据（State, Insolvency Type，Court/District），以规范所有 Case 中的这类信息，同时方便用户使用，在用户修改这类信息数据的时候，只需要从系统提供的列表中选择所需要的一个。

因此系统需要提供对这些数据的管理，也即增加，修改功能，当有些已存在的数据暂时不需要使用时，可以将其设为退休，需再次使用时只需要重设为可用的就可以了。

3.4 系统总体架构设计

本系统采用总体采用 MVC 的架构设计，但是系统并没有使用任何框架，因为现存的框架比较复杂，不适合 bankruptcy 系统开发的要求。MVC 设计模式将用户的显示界面，流程控制和模型层分开，因此在开发过程中体现了便捷，高效，整合的思想，使系统具备良好的可修改性和可扩展性[10]。具体结构如图 3-3 所示：

系统采用 MVC 的架构思想，分为：View, Controller, Model 层。Model 层主要负责与 bankruptcy 等数据库的交互，对外提供数据读写的接口(如 Case, Facet)。Controller 层负责解析用户发送的请求，并处理事件，将结果返回在试图上，主要是 case 的事件处理和 Facet 的事件处理。View 层主要是负责页面的显示，包括 case 详细信息的试图，导入 case 的页面，管理 Case 各种信息的页面。

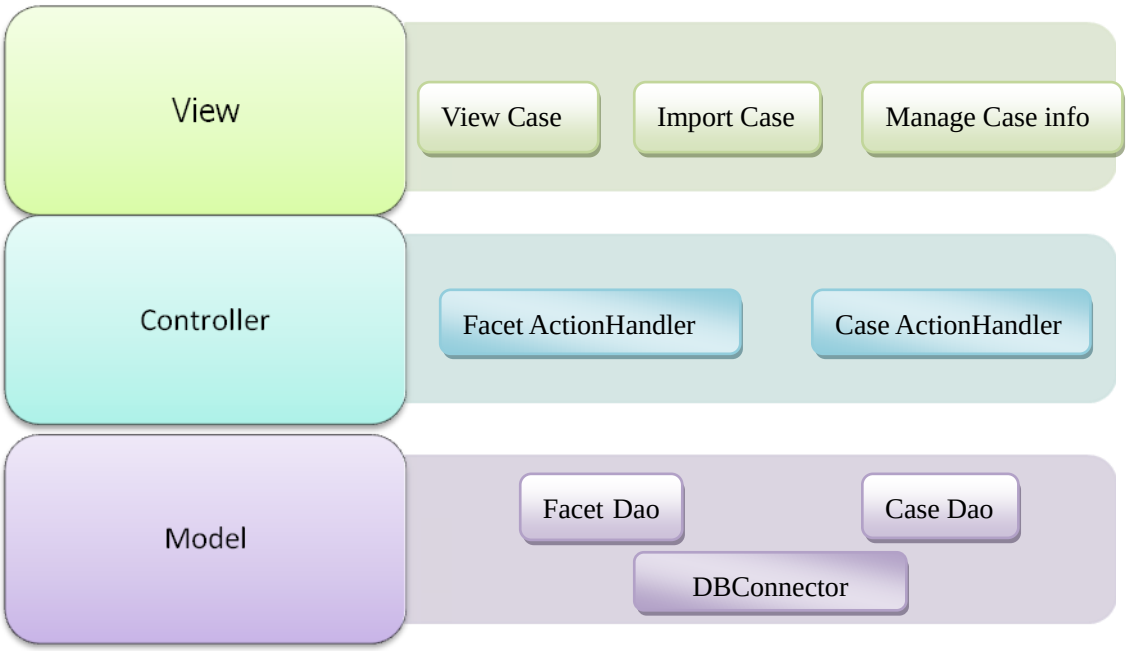

图 3- 3 系统架构设计图

3.5 本章小结

本章主要介绍了 Bankruptcy 系统的功能设计和系统的架构设计。

首先通过列表的形式介绍了系统的主要用户和以及相应的用例。普通的用户关注于 Case 信息的显示，而 Case 的 Owner 还关注于对 Case 的信息的修改于完善。接下来本章又介绍了对 Case 处理的流程，进而详细分析了系统的功能需求，然后根据系统的功能分析，设计了系统的总体架构。

第四章 系统详细设计

4.1 数据库设计

4.1.1 数据库 E-R 图

本系统主要的功能就是 Case 的管理和 Case 的数据的收集。如图所示，系统的 E-R 图模型就是围绕 Case 这个核心建立的。

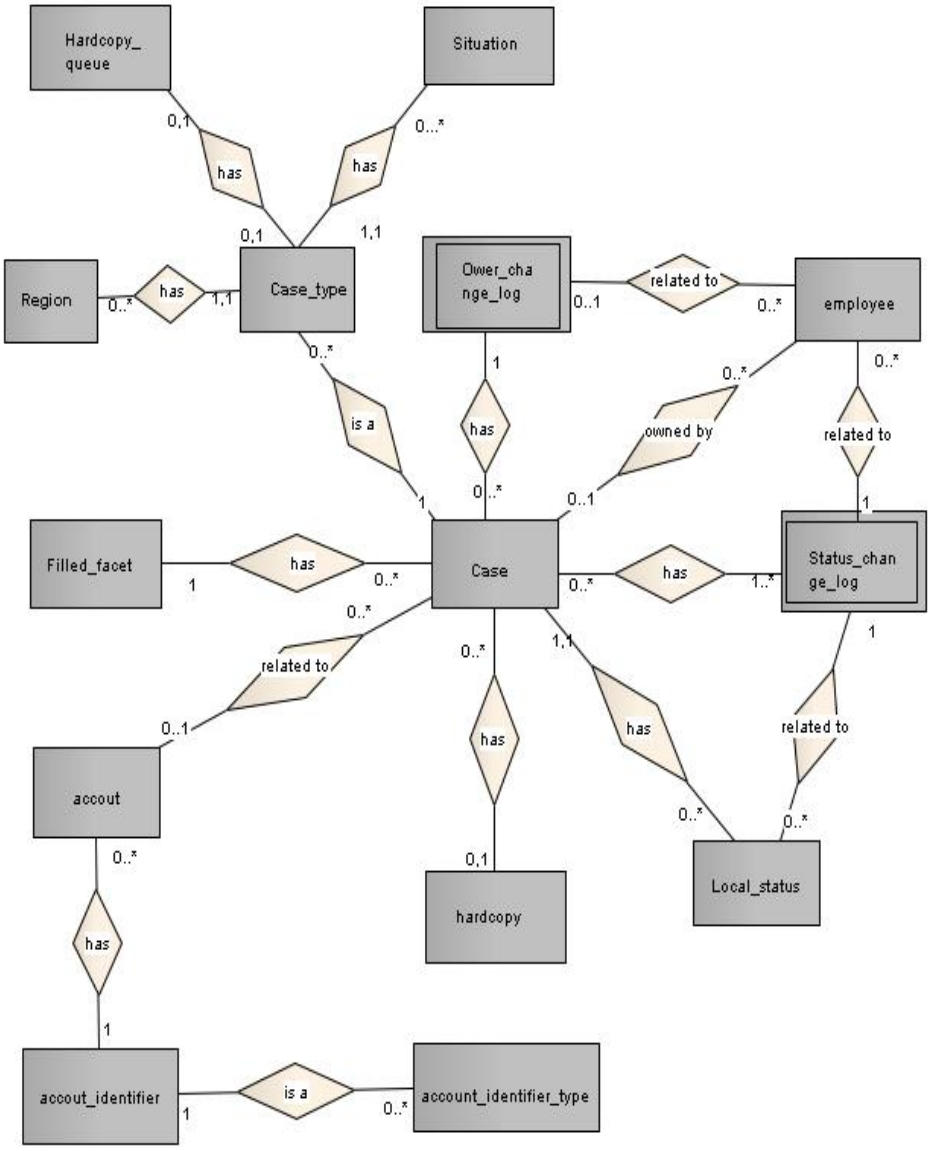

图 4- 1 Case 信息数据库 E-R 图

图 4-1 是一个关于 Case 详细信息的数据库 E-R 模型，这个 E-R 图模型中,Case

都属于某个特定的 Case Type(Case 类型)，这些 Case Type 可能属于不同的国家（Region），对应到不同的破产情况，比如疑似破产等，而每个国家的破产情况又不同，所以系统使用 Region 和 Situation 来定义一个 Case Type,一个 Case Type 还可能对应了一个 Hardcopy Queue（这个实体已经存在 Hardcopy Tool 系统的数据库中）。

在 Case 和 Employee（工作人员）之间存在占有（Owner）关系，也就是占有 Case 的 Employee 负责 Case 的处理，数据的收集，为了系统的安全和可追踪，派生出了弱实体 Owner_Change_Log（拥有者变化日志），记录着 Case 的 Owner 的变化以及变化时间。Case 在任何时刻都对应着一个 Local_Status(状态)，Case 的状态变化都是由 Owner 的操作的，因此又关联到了一个弱实体 Status_change_log(状态变化日志),该表用来记录所有 Case 状态变化的记录，并且记录了操作的工作人员。

每个 Case 还关联了 Account(账户)，一个 Account 可能有多个属于不同 account_identifier_type (账户标识符类型)的 account_identifier（账户标识符）。另外 Case 还有 Hardcopy(该实体已经存在 Hardcopy Tool 系统的数据库)，Hardcopy 作为该 Case 的破产凭证，所以一个 Hardcopy 都只能属于一个 Case，一个 Hardcopy 可能有多个文件。

Filled_Facet 是记录 Case 信息中每一个独立信息的实体，每个 Filled_Facet 属于一个 Filled_Facet_Type, Filled_Facet_Type 实体即记录的 Case 的信息的种类，里面存储了该 facet 所对应类的名字以及路径。

图 4-2 主要是一个关于 Case 显示的数据库 E-R 模型，由于 Case 具体显示哪些 Filled_Facet(信息)是由 Case_Type 决定的，所以下图是以 Case_Type 和 Filled_Facet_Type 为核心建立的。进而派生出了两个弱实体 Case_Display_Facet_Type_Map(Case Type 显示 Facet Type 的匹配) 和 Case_Aux_Display_Facet_Type_Map（Case 独立窗口显示 Facet Type 的匹配）。

Filled_Facet_Type 实体，记录的是 Case 中各个信息类型的实体。

如图中所示 Court/District，Insolvency_Type, Note 都是 Filled_Facet_Type 的一种，由于数据的特殊性，所以单独作为一个实体存储数据，没有存在 Filled_Facet 中。

State 实体是州的意思，即某个国家里的州或者省，所以与 Region 有关联。Court/District 实体记录的是该 Case 所处的法院链接，一个 State 中的法院可以就好几个，因而 Court/District 是由 Name, State, url 所定义的。

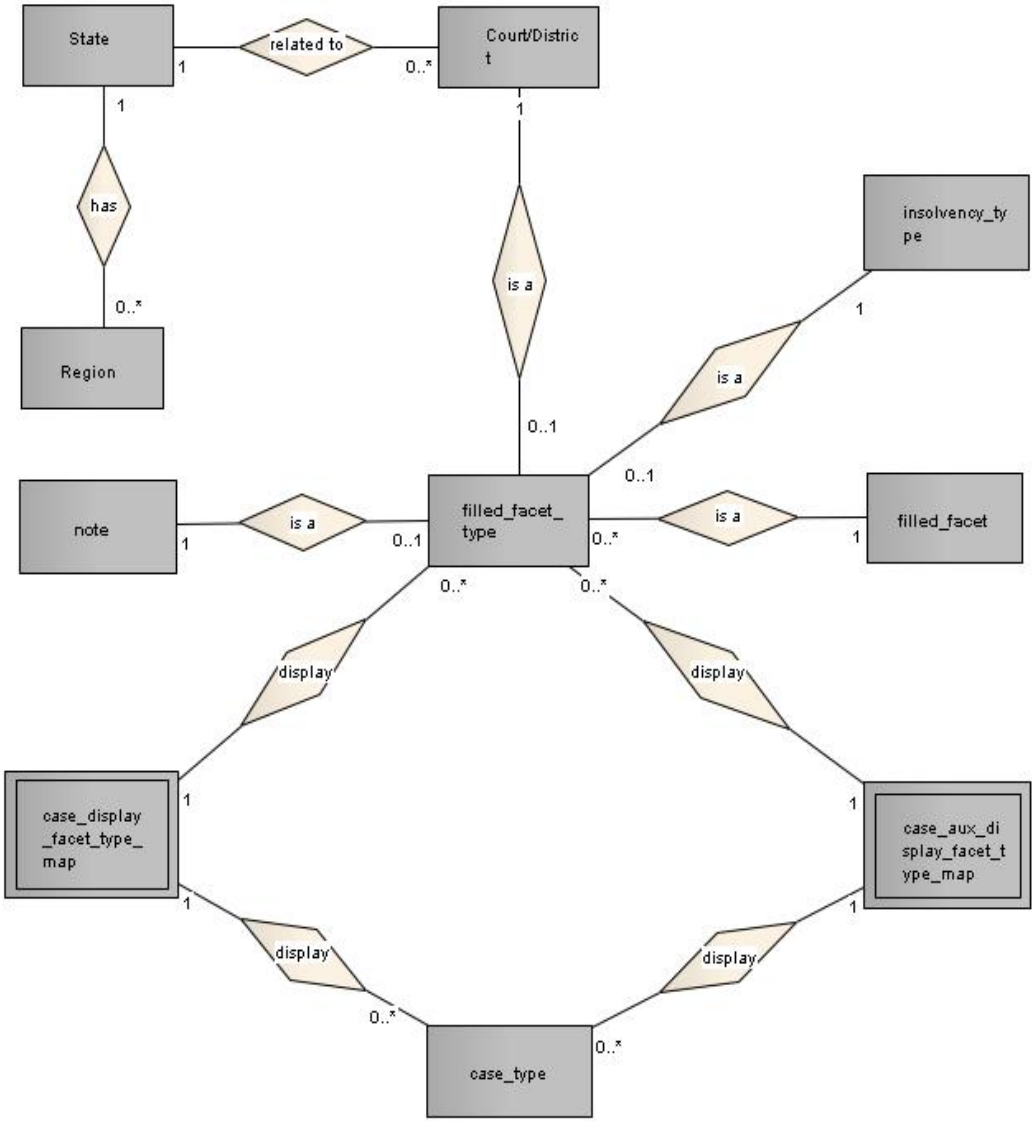

图 4- 2 Case 显示数据库 E-R 图

4.1.2 数据库表结构的设计

系统的数据库表结构是根据数据库的三范式基于上述的 E-R 模型建立的，在设计的过程中，还充分考虑了表结构的可维护性和可扩展性。

在 ebay，所有的表都不设置外键，防止在删除表的时候影响到其他的表，所以在下面的具体的表结构中，没有设置任何外键。

由于数据库中的表稍多一点，所以分成了如下三个部分：

- Case 详细信息数据库表结构

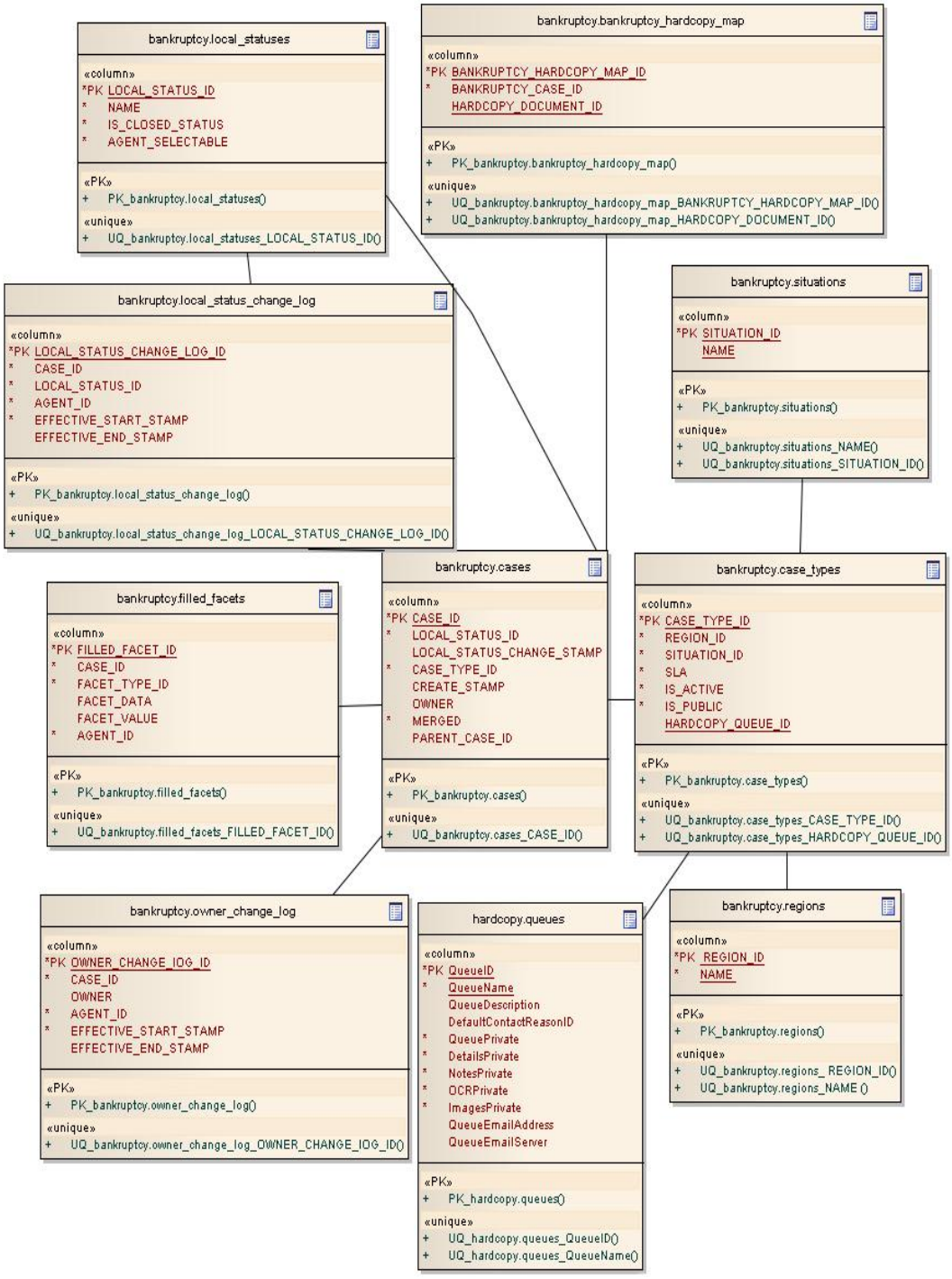

图 4- 3 Case 详细信息数据库表结构图

如图 4-3 所示,Situations, Regions 两张表记录了国家和破产情况的信息。Queues 表是已经存在的，记录了 Hardcopy 队列的信息，存在 Hardcopy 数据库中。Case Types 表记录的是 Case 类型的信息，其中的 Region_ID, Situation_ID, Hardcopy_Queue_ID 分别指向了 Regions 表，Situations 表，Queues 表。

Local_Statuses 存放的是 Case 的各个状态的信息。Cases 表存放了各个 Case

的信息，merged 代表是否被合并过，如果被合并过则父 case 的 ID 存在 Parent_Case_ID，Case 里的数据 Onwer 记录的是 Case 的占有者得 ID，指向 users 数据库中 cs_employee_comp（工作人员）表。

Hardcopy_document 实体之前就已经存在在其他的数据库，bankruptcy_ hardcopy_ map 用来表示 hardcopy 和 case 之间多对一的关系。Local_status_change_log 和 owner _change_log 用来记录 case 状态变化和 owner 变化的记录，Local_ status_change_log 中 case_id 和 local_status_id 分别指向 cases 表和 local_status 表，agent_id 指向 cs_employee_comp 表，记录操作 case 状态变化的工作人员，effective_start_stamp 和 effective_end_stamp 记录该 Case 的 Owner 起始时间和结束时间。owner_change_log 表跟 Local_status_change_log 类似。

Filled_facets 表记录的是 Case 的具体信息，表中 case_id 指向 Cases 表，facet_Type_id 指向 facet_types 表, agent_id 指向 cs_employee_comp 表，记录了最后一次修改 facet 信息的工作人员，表中 facet_data 和 facet_value 记录了 facet 的详细信息，当 filled_facets 中的 facet_type 为系统独立出来的 facet_type,如 inslovency_type、court 和 note 时，facet_value 指向了 inslovency_types 等，而其他的 facet_type 的数据存在 facet_data 中。

- Case 账户数据库表结构

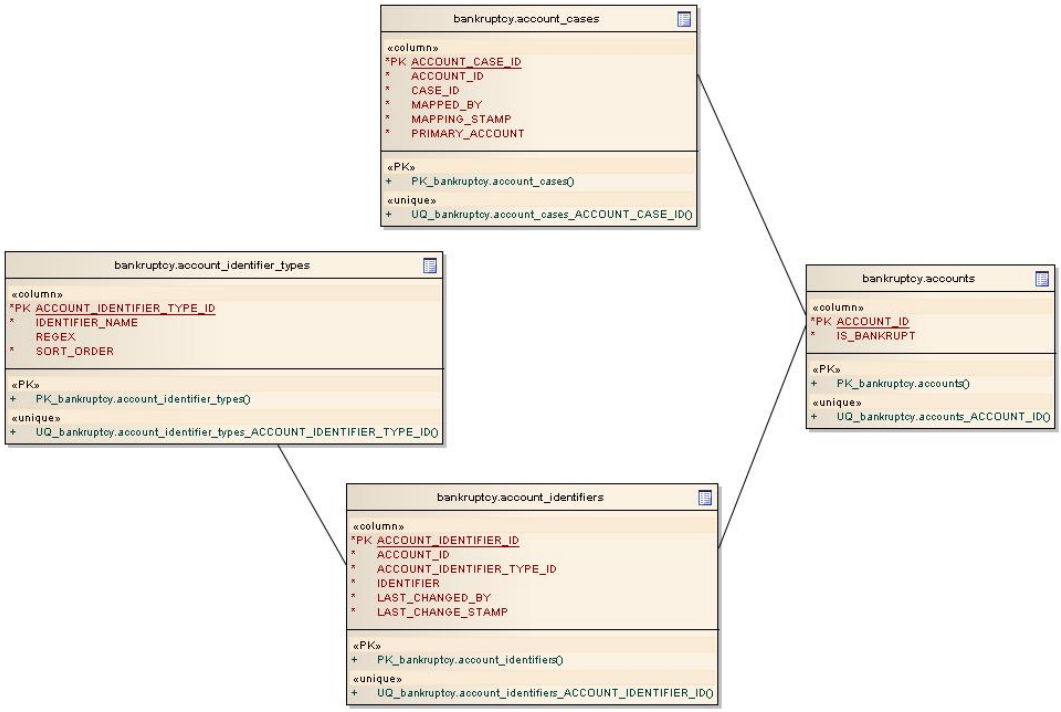

图 4- 4 Case 账户数据库表结构图

如图 4-4 所示,accounts(账户)中记录 bankruptcy 系统中存在的 case，

account_cases 用来表示 account 和 case 之间的关系,表中的 case_id 指向 cases 表，account_id 指向 accounts 表，mapped_by 指向 cs_employee_comp 表，记录的是将 case 和 account 关联起来的工作人员，primary_account 用来标识这个账户是否为改 case 的主账户。

Account_identifer_types(账户标识符类型)表记录了账户标识符类型的信息，account_identifiers(账户标识符)表记录各个账户的标识符，表中 account_id 指向 accounts 表， account_identifier_type_id 指向 account_identifier_types 表，last_changed_by 指向 cs_employee_comp。

- Case 显示数据库表结构

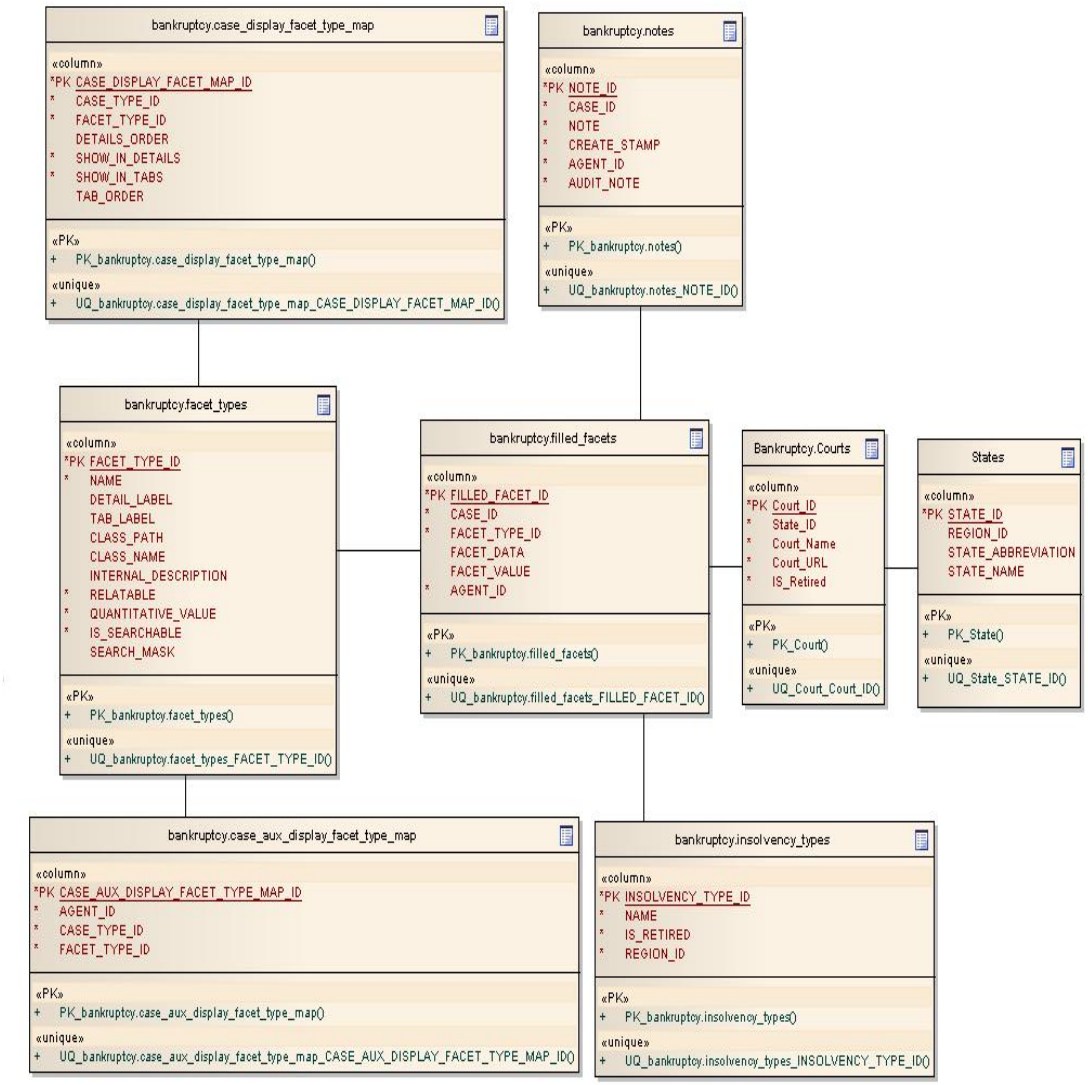

图 4- 5 Case 显示数据库表结构图

如图 4-5 所示，下图以 facet_types 表为核心建立的。Facet_types 记录了

facet_type 的详细数据。表中的 Name 记录的是 facet_type 的名字，detail_label 记录的是该类型在显示在 case 详细信息时候的标签，tab_label 是该类型显示在 Case 页面的选项卡上的标签，class_name 和 class_path 分别记录的是该 facet_type 在代码中的所处的类的名字和类的路径。

Case_display_facet_type_map 和 case_aux_display_facet_map 表记录的是 case_type 和 facet_type 显示关系。Case_display_facet_type_map 表中的 case_type_id 指向 case_types 表,facet_type_id 直线 facet_types 表，show_in_details 和 show_in_tab 记录的是该 facet_type 显示在 details 上还是 tab 上，details_order 和 tab_order 记录的是在 details 和 tab 上显示的顺序。

4.2 接口设计

4.2.1 HTML Tree API 接口设计

HTML Tree API（Application Programming Interface，应用编程接口）[11]是一些预先定义的函数，影藏内部实现的细节，开发人员只需了解如何使用这些 API 就可以了。

这套 API 的主要思想是将一个 HTML 页面抽象为一棵树，HTML 代码中的标签都抽象成一个个类，比如 Head 类，Body 类，各个标签的属性做作为其对应类的属性（attribute）。标签中的嵌套关系在 API 中则抽象为父子关系，高层的为父节点，下一层为子节点，这样就形成了一个 HTML 页面的树，树的节点即使 HTML 页面中的标签。在打印出 HTML 页面的时候，即是使用深度优先的原则遍历这棵树，将树上的每个节点打印出来。

将 HTML 页面抽象成 API 的优点是方便开发人员的使用，在前端开发时，开发人员不需要关注 HTML 的语法，比如标签是否正确，是否遗漏了标签的结束符。这样，开发人员可以着重关注整个页面的结构和代码的逻辑关系，提高了开发的效率。而且一个不太熟悉 HTML 的开发者无需了解内部的实现机制即可很快的写出一个可运行的 HTML 页面。

图 4-6 是 HTML Tree 的核心部分：

1. HTMLNode: 是整个 API 的核心接口，大部分的类都实现了这个接口。接口中提供了添加属性，添加子节点，取得和删除子节点，以及验证子节点,父节点和标签的方法。其中最核心的方法是 printNode（$indent）方法，即打印出该节点，生成 HTML 页面，各个类在继承时，都对这个方法进行了实现。

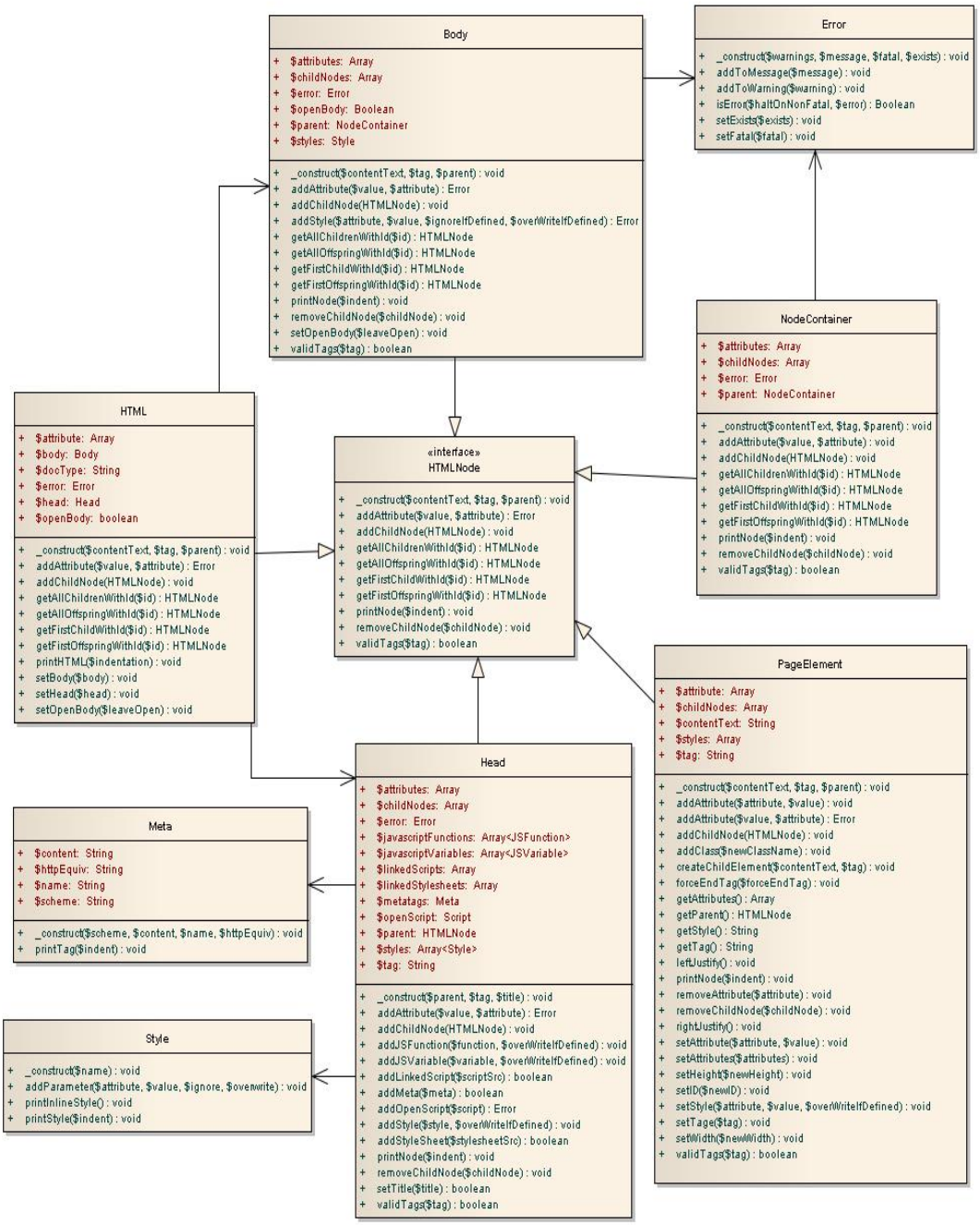

图 4- 6 HTMLNode 的接口设计图

2. Meta: 封装了 HTML 的<meta>标签，将其独立成一个类，类中封装了 httpEquiv, name, context, scheme 等属性。printTag（$indent）方法则是根据缩进将 Meta 打印成 HTML 页面中的 meta 标签形式。

3. Style: 封装了 HTML 的<style>标签，printStyle($indent)方法是根据缩进将将这个 style 打印成 html 标签的格式。

4. Body: 封装了 HTML 的<body>标签，它实现了 HTMLNode。

Head: 封装了 HTML 的<head>标签，它实现了 HTMLNode。该类提供了 addMeta（$meta）方法，即向 Head 里增加 Meta；addStyleSheet($styleSheetSrc) 方法，根据 css 路径给 Head 增加 style。

HTML: 把一个 HTML 页面封装成 HTML 对象，HTML 类中将 Head, Body 作为其属性，printHTML（$indentation）方法将整个 HTML 页面显示出来，在这个方法中通过顺序调用了 head, body 中的 printNode 的方法将 head，body 打印在 html 页面上。

图 4-7 是以 NodeContainer 为核心建立的类图：

NodeContainer 实现了 HTMLNode 接口，是 API 库中的核心类之一。NodeContainer 类没有跟任何 HTML 中的标签对应起来，只是一个抽象出来的类。NodeContainer 就像一个容器，可以往这个容器里面装东西。NodeContainer 的目的是为了更好的组织节点之间的逻辑关系，而不给本身的 HTML 代码增加任何复杂性。NodeContainer 实现了 printNode 方法。

AjaxNode:继承了 NodeContainer 类，封装了 Ajax，使用者无需了解 Ajax 技术，即可实现 Ajax 技术。

通过构造函数__construct($parent, $ajaxURL = null, $ajaxOnLoad = true, $loadingText = null, $delayInterval = null, $autoRefresh = false)创建 AjaxNode 对象，parent 是这个段 ajax 所放在哪个位置，即它的父节点，ajaxurl 指定了加载的内容的 url。

getTrigger（）方法是返回 loadAJAXHTMLNodeContent 的代码，使用者调用这个方法即可实现刷新页面。

Spinner:继承了 NodeContainer 类，该类封装了微调数据的控件。通过__construct($parent, $name = "spinner", $spinnerID = null, $default Value = 0, $min = 0, $max = 1000000)即可创建 Spinner 对象，$parent 即是该 spinner 所放的位置，即它的父节点，$defaultValue 为该 spinner 的初始显示的默认值。

DBFlagCheckBox:继承了 NodeContainer 类，封装了 checkbox 和自动处理。即当用户勾选或者勾掉该 DBFlagCheckBox 时，系统会自动修改数据库中的相应数据。创建 DBFlagCheckBox 对象时，通过向构造函数中传 parent，database，table，keyField 分别来制定该对象的父节点，数据库，表名以及列名来实现自动修改数据库中的相应数据。

TwoListSwapper:继承了 NodeContainer 类，封装了两个 list 交换的控件并带有保存按钮，点击交换按钮即可将两个列表中的内容对换，点击左移或者右移按钮则选中的元素则会相应的左移或者右移。通过向构造函数中传递$parent, $leftListArray，$rightListArray 即可创建该对象。

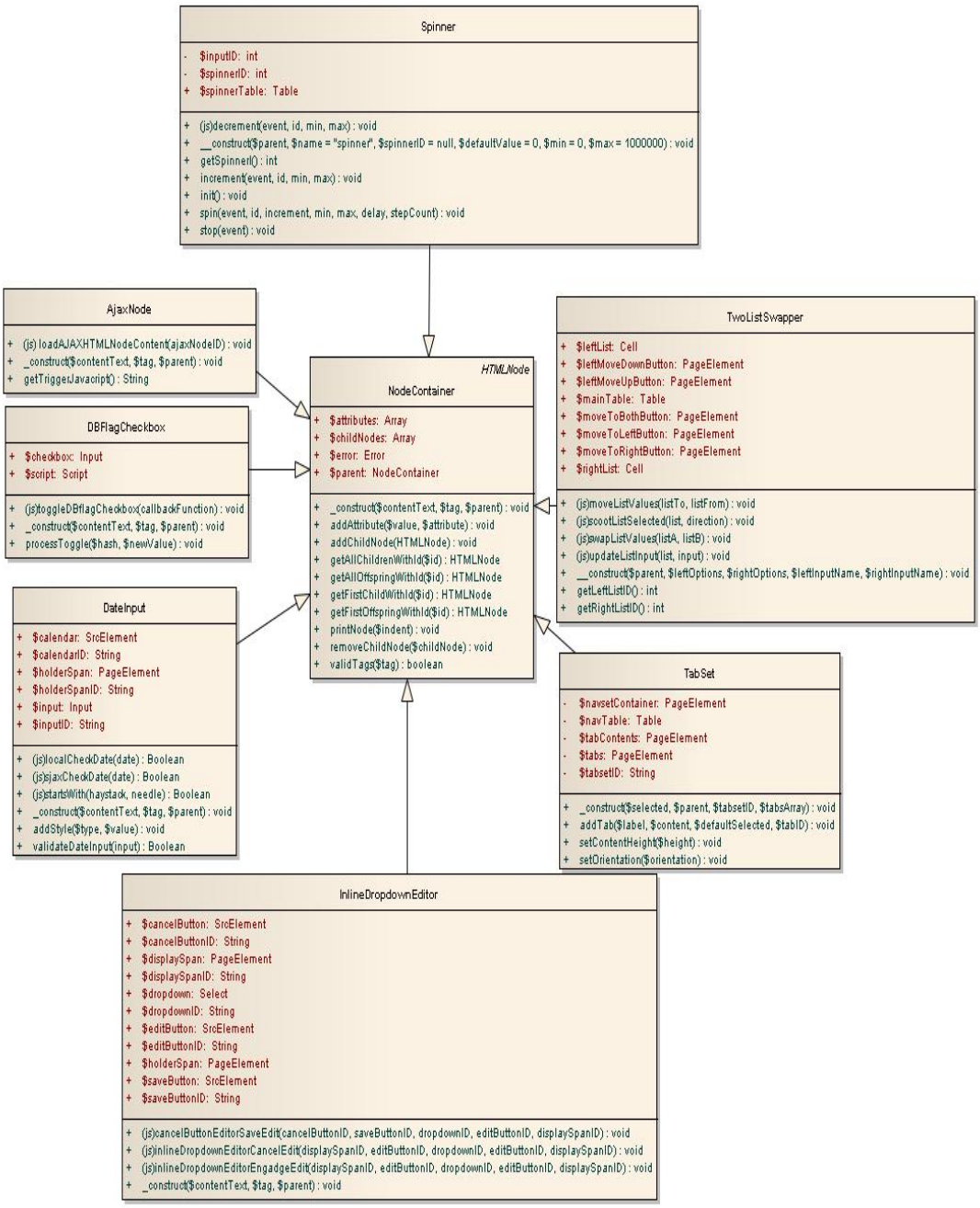

图 4- 7 NodeContainer 接口设计图

DateInput:继承了 NodeContainer 类，封装了日期选择的控件。方便用户使用，避免了用户输入时间的错误。

TabSet:继承了 NodeContainer 类，封装了一组选项卡控件，开发者可以增加或者删除其中某个选项卡。通过构造函数__construct($parent, $tabsetID = null, $tabsArray = array(), $selectedTab = null)建立对象，addTab($label, HTMLNode

$content, $defaultSelected = false, $tabID = null)方法是向其中加入选项卡的方法。

InlineDropdownEditor:继承了 NodeContainer 类，封装了修改内容的控件。点击修改按钮，即可修改内容，通过点击保存即可保存修改的内容，点击取消按钮撤销之前的操作，回到最初的状态。通过 construct($parent, $optionsArray = array(), $selectedValue = null, $name = null, $onSave = null, $onChange = null)构造函数来创建对象，其中$parent 为该对象的父节点，$optionArray 为下拉框中的选项，$selectedValue 为默认值，即最初选择的值。

图 4-8 是以 PageElement 为核心建立的类图：

PageElement 实现了 HTMLNode 接口，也是 API 库中的核心类之一。PageElement 类在实现了 HTMLNode 的方法的基础上，还增加了左对齐方法 leftJustify(),右对齐方法 rightJustify()，设置长宽的方法 setWidth($newWidth)，setHeight($newHeight)等方法。PageElement 与 NodeContainer 的最大的区别就是在打印 HTML 代码的时候，NodeContainer 打印的时候只遍历他的子节点，本身并不添加任何的 HTML 代码；而 PageElement 打印的时候首先会添加自身的 HTML 代码，再向他自身的标签中间添加子节点的 HTML 代码。

大部分封装 HTML 标签的类都继承了 PageElement,如 Table,Form，Select 等。

Table: 封装了表格的类，继承了 PageElement。一个 Table 是有多个 Row 组成的，一个 Row 是由多个 Cell 组成的，其中 Row 和 Cell 类都继承了 PageElement 类。通过构造函数 __construct($parent, $rows=null, $columns=null, $styleArray=array())传递$parent, $rows（行数）, $columns(列数)的数据创建该类的对象。

Select: 封装了一个下来菜单的类，继承了 PageElement。一个下拉菜单是有多个 Option 组成的，其中 Option 也是继承了 PageElement。

Script:封装了 JS 的类，继承了 PageElement。一个 Script 里面可能有多个 JsFunction,Script 中的 addFunction 就是向 Script 加入方法。JsFunction 中可能会有参数，即 JSVariable.

ScrElement: 继承了 PageElement,用于添加指定路径的资源，如图片。通过向构造函数__construct($parent, $type=null, $src="", $styleArray=array())中传递路径来创建对象。

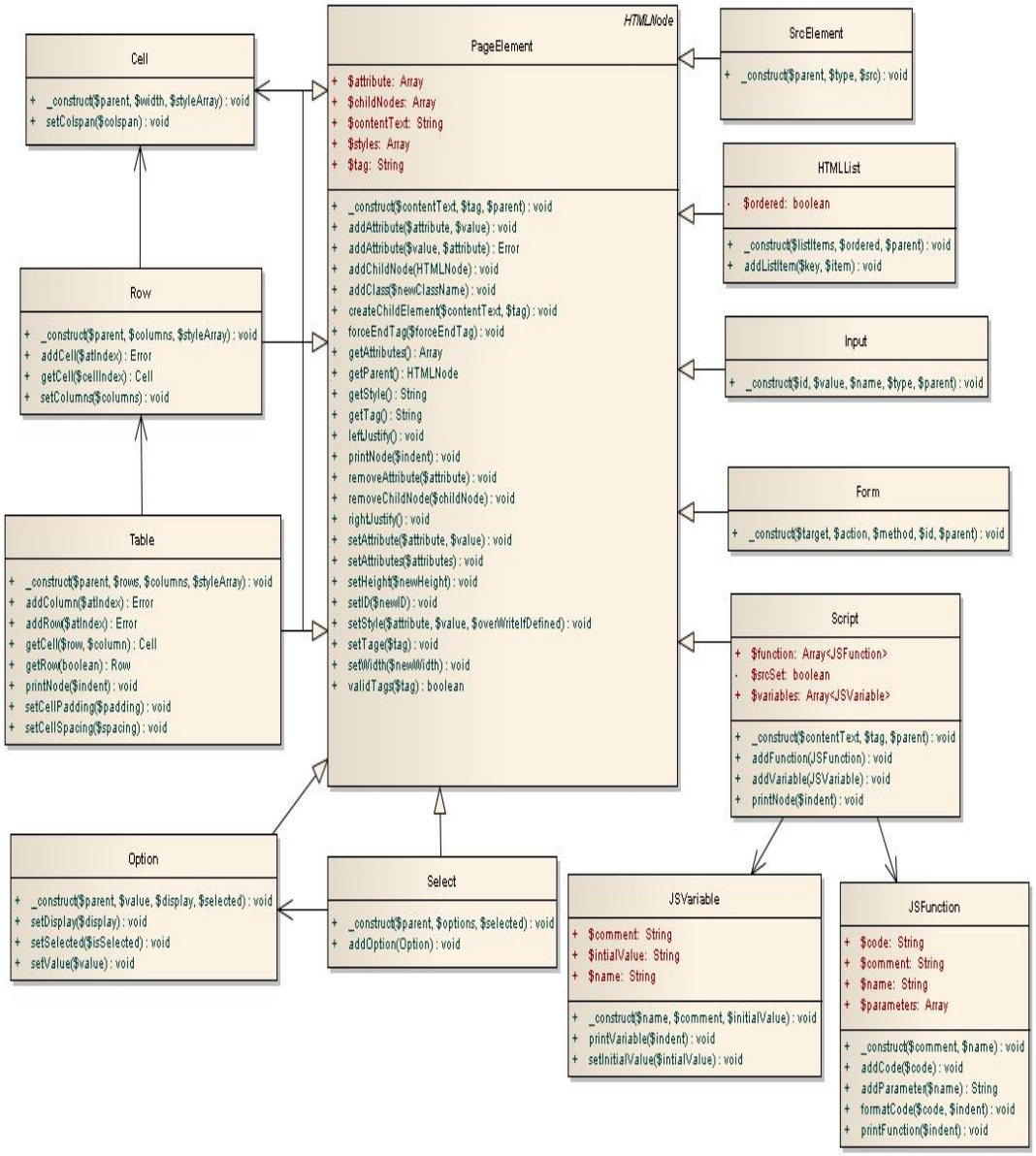

图 4- 8 PageElement 接口设计图

4.2.2 Bankruptcy 接口设计

1. MVC 三个主类

BKHTMLGenerator: 是 MVC 中的 View 部分，该类主要是用来显示页面。BKHTMLGenerator 里有 BKDBConnector 的引用，类中生成 HTML 页面使用了上述所述的 API，通过创建相应的类来实现页面。下面是类中的主要方法：

__construct(BKDBConnector $dbConn = null)：该类的构造函数。

wrapPageContent($pageContentTree, $pageName, $requiredAuth = null):该方法是将 pageContentTree 包裹成一个完整的 HTML 页面(包含系统的导航栏)的树，

pageName 作为这个 HTML 页面的标题，通过 printPage 的方法将整个页面打印成 HTML 的格式。每个页面显示之前都是调用 wrapPageContent 的方法，或者 wrapAUXPageContent，然后通过 printPage 方法将页面生成出来。

wrapAUXPageContent($pageContentTree, $pageName, $requiredAuth = null): 该方法与 wrapPageContent 方法类似，只是生成的 HTML 页面没有导航栏。

getWarningPage($msg)：该方法根据 message 信息生成警告页面，返回类型为 NodeContainer。

getNavMenu():该方法返回生成系统页面的导航栏，返回类型为 PageElement。

getMergeCaseView($childCaseID, $parentCaseID):该方法根据子 Case 的 ID 和 p 父 Case 的 ID，将子 Case 的信息和父 Case 的信息显示到页面上，并提供确认合并的按钮，返回类型为 pageElement。

getSituationAdmin(): 该方法返回 situtation 的管理页面，返回类型为 NodeContainer。

getNewCaseView($caseTypeID)：该方法返回导入新 Case 的页面，返回类型为 NodeContainer。

getQueueView($caseTypeID)：该方法返回此 caseType 的所有 case 列表页面，返回类型为 PageElement。

getCaseView($caseID):该方法返回此 case 的详细信息页面，返回类型为 NodeContainer。

getCaseTypeConfig($parent, $caseTypeID):该方法是返回该case类型的配置页面，即配置显示哪些信息，哪些状态等。$parent 指该节点的父节点。

getHighlightScript:该方法返回列表中某一个行突出效果的 js 代码，返回类型为 Script。该 Script 里有两个 js 方法，一个是 highlightRow，另一个是 clearRowHighlight。

checkIdentifier()：该方法是纯 ajax 实现检测用户输入的标识符是否正确，如果不正确，则会在系统显示错误的信息给用户。

checkCaseType（caseID,searchType,defaultSelectID）：该方法是纯 ajax 实现检测 Case 的类型是否正确，并给用户显示相应的信息。

- BKActionHandler: 是 MVC 中 Controller 部分，该类主要进行一些事件的定位和处理。

下面是类中主要方法：

__construct(BKDBConnector $dbConn = null)：该类的构造函数。

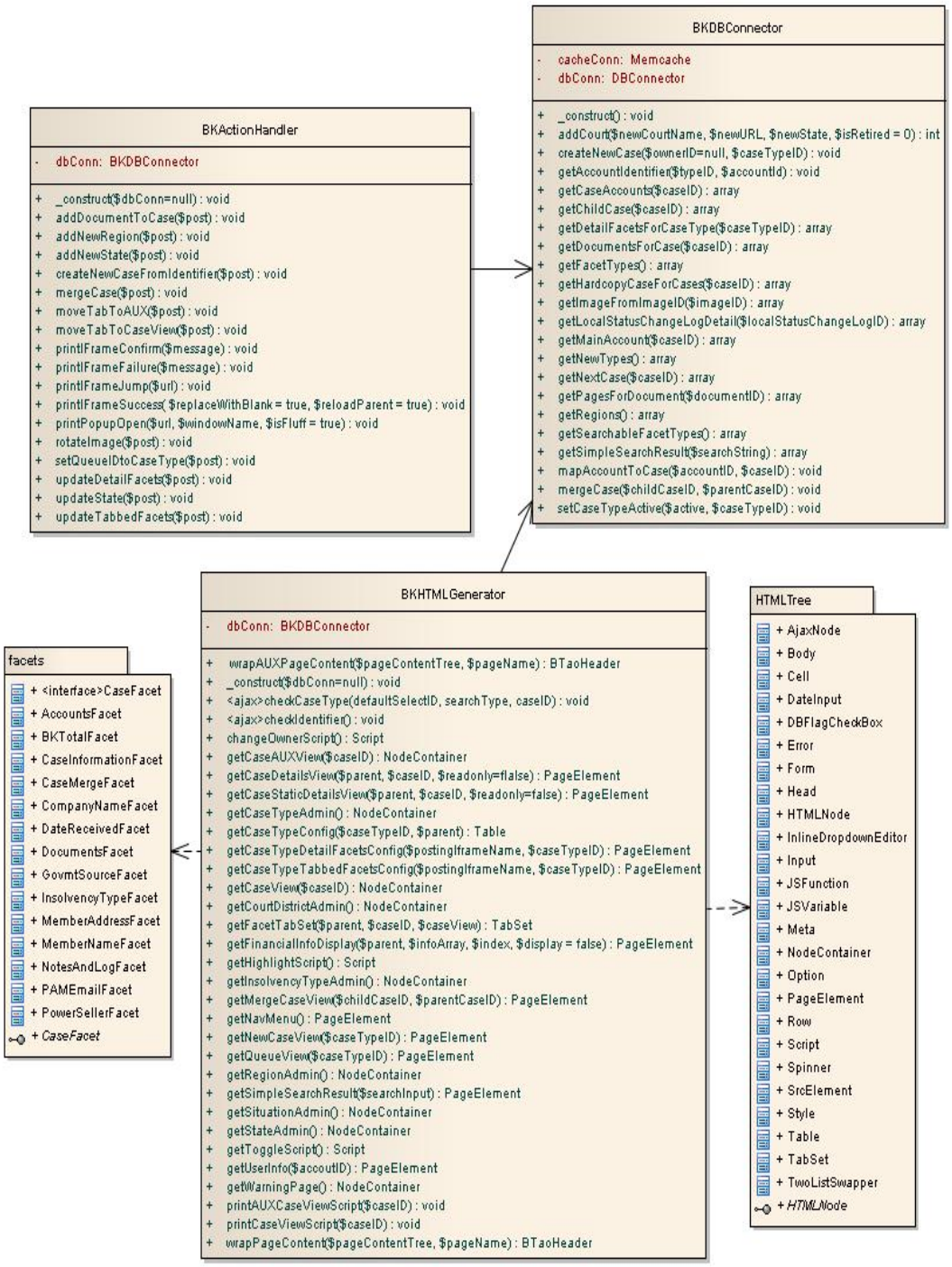

图 4- 9 PHP 三个类接口设计图

printIFrameFailure($message)：该方法是在页面上弹出错误信息，即传入的message。

printIFrameJump($url)：该方法将页面重定向到给定的 url 页面。

printIFrameCallback($funtionName, $args)：该方法调用父页面的函数，$functionName 即是函数的名字，$args 是传递给函数作为参数的数组。

printIFrameSuccess($reloadParent = true, $replaceWithBlank = true)：如果 reloadParent 如 true 则刷新父页面，否则如果 replaceWithBlank 将当前页面重定向到系统的首页面。

printPopupOpen($url, $windowName)：该方法在当前的页面上再打开一个新的固定大小的窗口。$url 即新窗口的 url,$windowName 即新窗口的名字。

addNewSituation($post):该方法根据 situationName 增加一个破产 situation，增加成功后刷新页面。

updateSituation($post):该方法根据 sitution 的 ID 更新该 situation 的信息。

confirmMerge($post)：该方法是对要合并的两个 Case 进行检查，如果不符合要求则会给出相应的错误信息提示，如果符合要求则会打开新的页面显示父子 Case 的信息。

mergeCase($post)：该方法是合并父子 Case，并更新数据库中的数据，成功合并后，系统会自动关闭当前的窗口，将父窗口重定向到父 Case 的视图页面。

createNewCaseFromIdentifier($post)：该方法用来处理从标识符和标识符的类型创建的 Case，并且如果不能成功创建 Case,系统会给出相应的错误信息。

importHardcopyCaseAsNew($post)：该方法用来处理从一个 Hardcopy 导入 Case。

importQueueCaseAsNew($post)：该方法用来处理从一个 Hardcopy 队列导入 Case,并且一个 Hardcopy 导入一个 Case，如果不能成功创建 Case,系统会给出相应的错误信息。

setQueueIDtoCaseType($post)：该方法用来处理设置某 Case 类型所关联的 Hardcopyd 队列，如果不能成功创建 Case,系统会给出相应的错误信息。
updateDetailFacets($post)：该方法用来设置某个 Case 类型所显示的的信息。
rotateImage($post)：该方法用来处理旋转图片的事件。

- BKDBConnector: 是 MVC 中的 Model 部分，该类提供了与数据库交互的所需要的接口。该类中有对 DBConnector 类的引用。以下是类中的主要方法：

__construct()：该类的构造函数，链接到数据库。

getCaseTypes()：该方法返回从数据库中取得所有的 Case 类型以及详细信息的数组。

getOpenCasesForQueue($queueID)：该方法返回根据 hardcopy 队列的 ID 从数据库中取得的可用的 hardcopy 以及其详细信息的数组。

createNewCase($caseTypeID, $ownerID=null)：该方法用来在数据库中增加一个新的 Case,并且 case 的类型 ID 为$caseTypeID，case 的 owner ID 为$ownerID，如果创建成功，则返回该新 case 的 ID，否则返回 false。

getChildCase($caseID)：该方法返回此 Case 的所有子 Case 以及其详细信息的数组。

mergeCase($childCaseID, $parentCaseID)：该方法用父 case 的信息来更新子 Case 数据库中的数据。

mapAccountToCase($caseID, $accountID)：该方法将 case 和 account 关联起来并将其存到数据库中。

startNewStamp($caseID, $agentID)：该方法用来向数据库增加一条用户在某个 Case 上工作开始工作的时间日志记录。

extendStamp($workTimeLogID)：该方法用来向数据库增加一条用户在某个 Case 上结束工作的时间日志记录，

registerBKAccount()：该方法用来注册一个新的账户，即向数据库中增加一条账户记录，并返回该账户的 ID。

addIdentifierToAccount($accountID, $identifier, $idType, $agentID)：该方法用来向数据库中给账户增加标识符，$accountID 是账户 ID，$identifier 为待增加的标识符，$idType 为该标识符所属的类型，$agentID 为当前操作的工作人员等的 ID。

addCaseType($regionID, $situationID)：该方法用来向数据库中添加一个新的 CaseType。

updateSituation($situationID, $newSituationName)：该方法用来根据 situation ID 更新已存在数据库中的 situtation 的信息。

2. Facet 类

如图 4-10 所示，Case 的各个部分的独立信息都被封装成一个类，存在 Facet 包中。这个包中的所有类都实现了 CaseFacet 接口。比如 InsolvencyFacet 是将 case 中显示 Insolvency Type 信息的部分封装成一个类。

CaseFacet 类主要方法有：

__construct(BKDBConnector $dbConn = null)：该类的构造函数。

getTabInterface($caseID, $facetTypeID, $filledFacetID = false, $readonly = false)：该方法返回封装了这个 case 在此 facetTypeID 上的详细信息的显示模块，$readonly 参数的作用是在这个模块中，是否提供可修改的按钮。

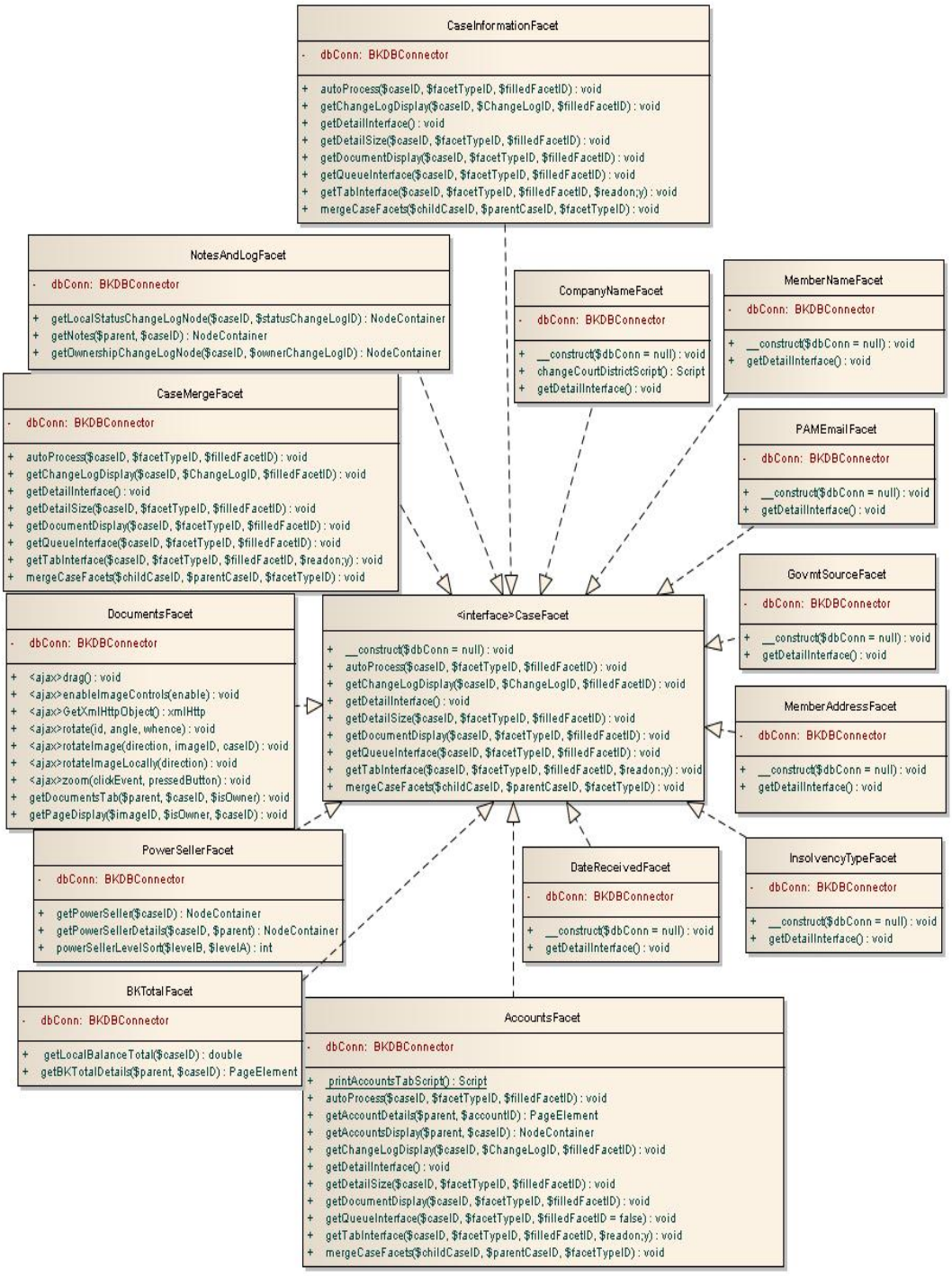

图 4- 10 Facet 包接口设计图

getDetailInterface($caseID, $facetTypeID, $readonly = false, $filledFacetID = false)：该方法与 getTabInterface 类似，唯一的区别是 getTabInterface 用于加载在选项卡上显示出来，而该方法用于显示在 Case 视图上半部分小数据量信息。

mergeCaseFacets($childCaseID, $parentCaseID, $facetTypeID, $filledFacetID =

false)：该方法用来合并父子 case 在此 facet 上的信息，即如果子 Case 在该 facet 上的信息为空，那么将父 case 的信息赋值给子 case。

图中 NotesAndLogFacet, DocumentsFacet, AccountsFacet 等类都是继承了 Facet 类，由于这类信息是显示在选项卡中，这些类都实现了 getTabInterface,mergeCaseFacets 方法。而对于 DateReceivedFacet, InsolvencyTypeFacet, MemberAddressFacet 等类也继承了 Facet 类，但是并不需要显示在选项卡中，所以没有实现 getTabInterface 方法，而是实现了 getDetailInterface 方法，当然同时也实现了 mergeCaseFacets 方法。

4.3 页面模块设计

4.3.1 系统导航边栏

为了延续一贯的工具风格，系统在绝大部分页面的左侧提供导航边栏。这样不管用户在哪个页面，都可以快捷地通过导航边栏找到自己的目标页面。本系统的导航边栏分为两个部分：Cases, Tool(工具)。

Cases 按顺序显示出所有的 Case Type 以及这个类型的 Case 的总数，用户可以通过点击某个 Case 类型，进入这个 Case 类型的所有的 Case 列表页面。

Tool(工具)导航部分，为用户提供了一些辅助系统的功能。首先为用户提供一个搜索 Case 的功能，用户通过输入关键字即可查找到相应的 Case。Tool 导航部分还为用户提供了 Region, Situation, Case Types, Insolvency Type, Court/District 等链接，用户只需要点击，就可进入相关的页面。

4.3.2 Case 列表页面

Case 列表页面中以列表的形式按照 case 导入系统中的时间的顺序依次显示了某 Case 类型的所有 Case。在一行中显示该 Case 的导入时间，case ID，case 状态以及 Case 的 Owner 名字和 open 链接，用户只需点击 open，系统则打开该 Case 详细信息的页面。

此外还为用户提供了创建 Case 的按钮，通过点击这个按钮，用户就定向到导入新 Case 的页面。

4.3.3 导入新 Case 页面

Bankruptcy 系统给用户提供了三种方式来创建 Case 的途径。

在页面的左半部分显示的是从 Hardcopy/Hardcopy Queue 导入 Case。对于从 Hardcopy 导入 Case，页面提供一个输入框供用户输入 Hardcopy 的 ID，点击导

入按钮，即可以导入创建 Case，如果创建成功，则跳转到新建的 Case 页面，如果失败，系统会给出相应的错误信息。对于从 Hardcopy 队列导入 Case,系统自动显示了跟该 Case 类型相关联的 Hardcopy 队列的名字，点击队列名字，用户可以查看到这个队列的详细情况，用户通过点击创建按钮，即创建 Case，如所有的创建成功，则会自动跳转到此 Case 类型的所有 Case 的列表页面，如果失败，系统会给出相应的错误信息。如果该 Case 类型没有关联的 Hardcopy 队列，则不能从 Hardcopy 队列导入 Case。

4.3.4 Case 视图页面

Bankruptcy 系统的主要目的就是方便用户跟踪记录 Case 的信息，以方便用户处理 Case。所以 Case 视图页面也是系统的核心页面之一。

Case 视图即显示 Case 详细信息的页面，主要两部分组成，一部分信息显示在页面的上半部分，主要是显示信息量比较小的信息，比如时间，Case 的状态等，按照用户对此 Case 类型的配置的显示顺序依次将信息显示到页面,如果这些信息一个页面显示不下，则需要分栏显示，当用户需要看下面栏的显示时，只需按下按钮，系统会自动显示出来。另外还有一部分信息显示在页面的下半部分，不同类型的信息显示在不同的选项卡，比如有 Hardcopy 选项卡，账户选项卡等。为了用户的不同需要和方便用户使用，Case 详细信息的页面上需要提供其他辅助按钮：

- “Next Case”按钮，用户点击此按钮，系统则会跳转到下一个还未处理的 Case 页面。
- “Return To Queue”按钮，用户点击此按钮，系统则会跳转到显示这个 Case 类型的所有 Case 列表的页面。
- “UP Hidden”按钮，用户点击此按钮，系统则会将显示在页面上半部分的信息隐藏，此时只剩下显示在选项卡中的 Case 的信息。
- “Left Hidden”按钮，用户点击此按钮，系统则将页面左边的导航边栏隐藏。

如果系统当前登录用户为此 Case 的拥有者，则系统在每个信息的显示位置为用户提供按钮来修改此信息。

- 对于 Region, Situation, Insolvency Type，Court/District 这样的信息，用户点击修改按钮，系统则会给出所有这类信息的数据列表供用户选择，用户点击确定后，系统就会更新 Case 的数据，并将修改后的数据显示在页面上，如果点击取消，则会取消之前的操作。

- 对于需要用户自己输入的信息比如 Member Address 的数据,用户点击修改按钮，系统则会给出输入框供用户输入数据，点击确定按钮，更新 Case 的数据，页面将显示出最新的数据。
- 对于 Document(即 hardcopy)信息，用户通过输入 Hardcopy 的 ID 给 case 增加 Hardcopy。如果输入的 ID 已经关联到某个 Case 或者 ID 错误,那么系统在页面显示错误信息。由于 Hardcopy Tool 系统存储的图片有可能不是正放的，所以用户可以对 Hardcopy 里的每个文件图片进行旋转，方便阅读。如果该 Hardcopy 有多页，则
- 对于账户信息，用户可以通过输入账户的标识符和选择标识符所属的类型来给该 Case 添加账户，同样如果这个账户已经关联到其他 Case 或者根本不存在，系统会在页面上显示出错误信息。由于一个 Case 可能关联到很多账户，所以用户可以设置其中一个为这个 Case 的主账户，方便用户处理 Case。
- 对于时间类的信息的修改，用户只需要点击按钮，系统则会提供一个日历表供用户选择日期。

4.3.5 合并 Case 页面

打开显示 Case 详细信息的页面，如果当前登录的用户为该 Case 的 Owner，则页面上会提供输入框供用户输入 Case ID 来与当前 Case 合并，系统会对输入的 CaseID 进行验证，如果验证通过，系统会给出一个再次确认的窗口，窗口上显示了两个 Case 的相信信息，确认和取消按钮，当用户点击确认，系统则会合并两个 Case。

4.3.6 Case Type 管理页面

该页面以列表的形式列出了当前系统中所有的 Case Type 的列表，列表中显示了这个 Case Type 的 region 名字，situation 名字，options 按钮，以及设置该 case type 是否 active, public 的复选框。当点击 options 按钮的时候，系统跳转到该 Case Type 配置的页面。

在列表的下方，页面还提供了增加 Case Type 的按钮，用户只需在下拉框中选择 region 和 situation 即可添加，如果添加成功，系统会自动刷新页面，以便新添加的显示在列表上。

4.3.7 Case Type 配置页面

为了实现系统的灵活性，处理各种类型的 Case，系统需要为用户提供定制

Case 视图的功能。系统为每个 Case 类型提供了一个视图配置的页面，提供以下配置的内容：

- 用户从 Hardcopy 队列的下拉单中选择一个 Hardcopy 队列，点击确认按钮，系统会则将其设置为与该 Case 类型相关联的 Hardcopy 队列。
- 系统给用户提供所有 Case 可显示信息的列表，用户通过选择其中的部分，设置其为该 Case 类型的所有 Case 在 Case 视图上所需显示的信息，选择的顺序则为在 Case 视图上显示的顺序，此信息显示在 Case 视图的上半部分。对于 Case 视图下半部分的选项卡显示内容，通过同样的方式设置。
- 系统为每个 Case 类型所设的 SLA Minutes（即 Case 从创建到自动关闭的时间）的缺省值为 7200 分钟，用户点击微调按钮来修改 SLA Minutes。
- Case 有很多个状态，比如最新创建，创建，关闭，合并等，用户只需点击按钮，可以将已存在的状态设置为不可用，也可以通过在输入框中输入状态的名字来为该 Case 类型增加状态的个数。

4.3.8 搜索 Case

系统提供输入框供用户输入关键字，点击搜索按钮，系统则会在 Case 的详细信息中搜索关键字，并以列表的形式返回查询结果。

为了方便用户的使用，搜索框放在系统的导航边栏里。

4.3.9 管理 Case 属性页面

对于 Region, Situation, Court/District, State, Insolvency Type 等页面，页面结构都相似，都是在页面的左边显示增加的按钮，当添加成功后，会自动刷新页面。页面的右边显示已存在的列表，并且提供对已存在的进行修改的按钮。

4.4 本章小结

本章为系统的详细设计部分，从数据库，接口，页面三个部分来介绍。首先根据设计的 E-R 关系图，建立了各个数据库中的各个表结构。，接着又从 HTML Tree API，PHP 前端代码两部分介绍了系统的接口设计。最后根据系统的功能需求，设计了系统的各个页面。

第五章 系统实现

5.1 导入 Case 的实现

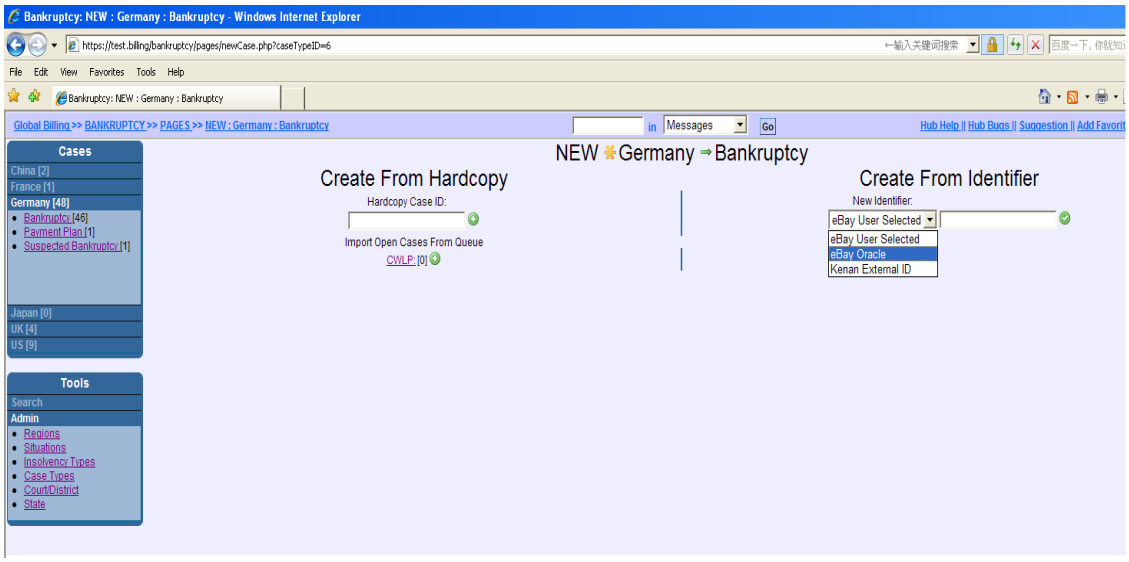

图 5- 1 导入新 Case 图

图 5-1 是向系统中导入新 Case 的界面，界面的左侧是系统的导航边栏，主体部分是用户通过三种方式导入 Case，其中上面的 Germany->Bankruptcy 是 case Type 的名字，Germany 是 Region，Bankruptcy 是 Situation, Create From Hardcopy 部分的链接,其中 CWLP 为该 case Type 所关联的 Hardcopy 队列的名字,[0]代表 CWLP 对列中状态为可用的 Hardcopy 的个数，点击这个链接，用户即可查看该队列的详细信息。包括队列中所有可用的 Hardcopy 的列表以及每个 Hardcopy 的详细信息。

右侧下拉框为所有标识符类型的列表，右边的钩按钮为”check identifier”，如果验证通过则可以导入 Case，进行相关的创建 Case 的活动，下文将会一一介绍。

另外在第一次打开系统的页面的时候，需要用户输入自己的工作账号以及密码(这部分是已经提供好的)，并写到了 PHP 的 cookie 当中，这样方便在系统中得到当前登录的用户的信息。

以下详细介绍了三种导入 Case 的方法：

- 用账户的标识符和标识符的类型来创建 Case

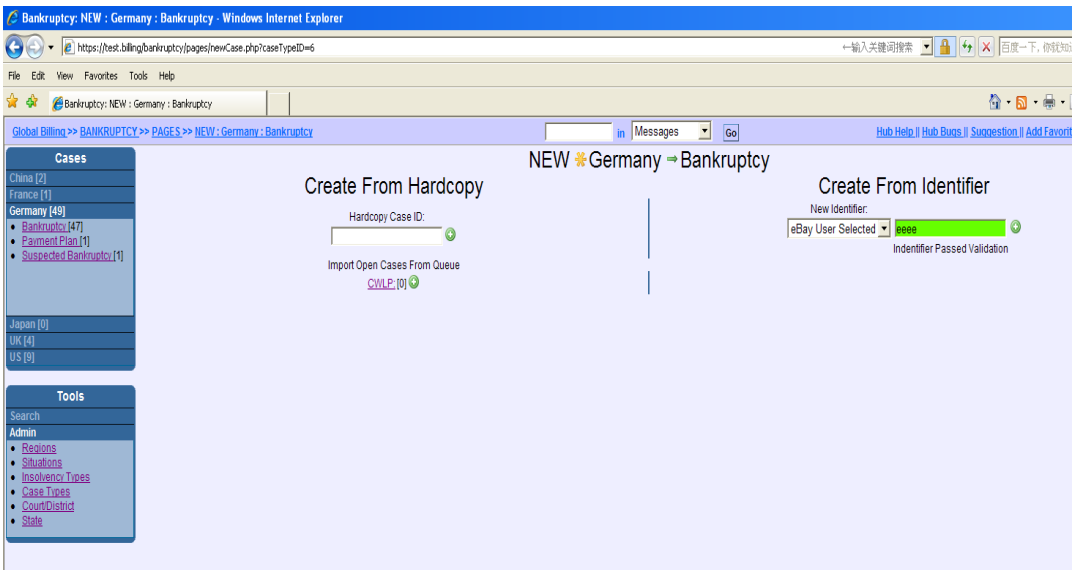

图 5- 2 成功导入 Case 图

图 5-2 图为用户选择了 eBay User Selected 的标识符类型，并输入”eeee”的标识符，点击按钮，通过验证后，页面提示“Identifier passed Validation”,并且输入框背景颜色变为绿色，钩状的按钮变为添加按钮，点击添加，即导入新 Case。

如果输入的标识符错误，比如标识符的类型选择为“eBay Oracle“，输入框中输入”ctest”，点击 check 按钮，检测出标识符错误，则输入框背景色变为红色，且提示“Invalid Identifier”；如果输入的标识符已经存在在系统中，则钩状（check）按钮变为箭头（go to Existing Case），用户点击按钮即可进入此 Case 页面。

如图 5-3 所示用户选择标识符的类型并且输入用户的标识符,系统将 request 发 BKActionHandler 处理， BKActionHandler 根据请求的 url 定位到 BKActionHandler 类得 createNewCaseFromIdentifier($post)方法进行处理，具体处理如图 5-4 所示：

（1） 根据用户选择的标识符类型从数据库获得该类型的详细信息，其中包括该类型标识符的正则表达式。

（2） 将用户输入标识符与从数据库获得的正则表达式做匹配，如果不匹配，则返回错误信息给用户，结束处理。如果匹配正确，则获得该账户的所有标识符信息。

（3） 如果有标识符存在，则得到的标识符数组中做循环在数据库中根据标识符类型和标识符查找是否已经存在，如果已经存在，则给用户打开其所关联的 Case 的视图，处理结束。

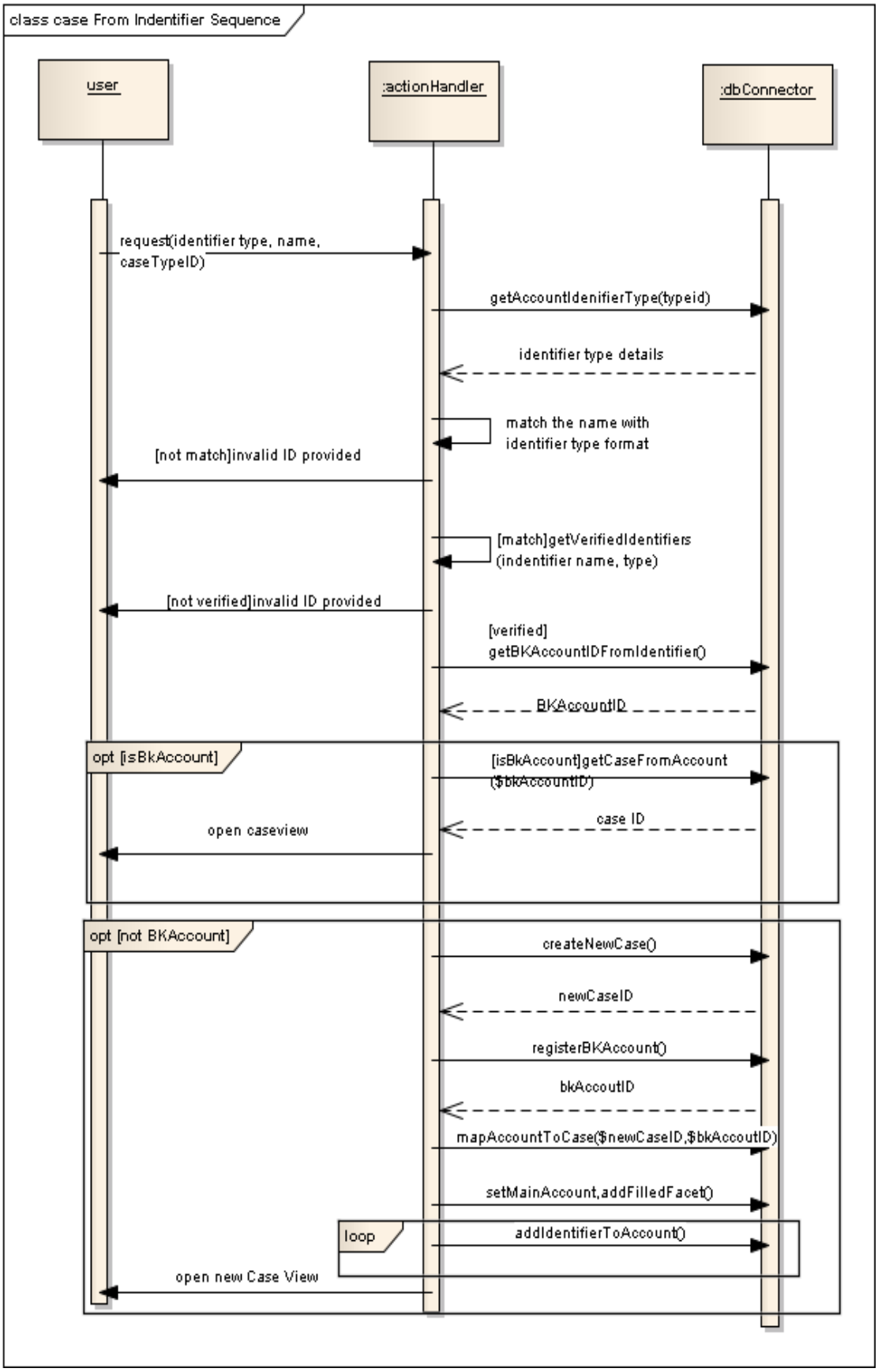

图 5- 3 通过标识符导入 Case 顺序图

（4） 如果不存在，注册一个 BK 账户，即向数据库增加一条账户记录，返回新账户的 ID，将这个账户与新建的 Case 关联起来，并且将其设为主账户，将账户的名字和地址设为这个 case 的 member name 和 member address(通过 addFilledFacet 方法)并存到数据库。

（5） 将该账户所有的标识符与 BK 账户关联起来，并更新到数据中。

（6） 成功创建 Case,导入过程结束，系统自动跳转到新 Case 页面。

- 从 Hardcopy 导入 Case

如图 5-4 所示，用户输入需要创建 Case 的 Hardcopy Id,系统将 request 发给 BKActionHandler 处理,BKActionHandler 根据请求的 url 定位到 BKActionHandler 类得 importHardcopyCaseAsNew($post)方法进行处理，具体处理如图 5-5 所示：

（1） 根据用户输入的 hardcopyID, 系统从 hardcopy 数据库获得该 hardcopy 中的所有的文档，返回一个文档的数组。

（2） 如果有文档存在，则说明 hardcopy id 输入正确，系统则会循环遍历这个文档数组，如果在 bankruptcy 数据库查询到了某个文档，则返回“Existing Document Found”，不能创建新的 Case，处理结束。

（3） 如果没有任何文档已存在 bankruptcy 系统中，系统则创建新的 Case, 返回新建 Case 的 ID。

（4） 如果 User ID 存在，则根据从数据库取出的正则表达式与 user ID 进行匹配。如果匹配正确，则获得该 userID 的所有标识符。

（5） 如果有标识符存在，则得到的标识符数组中做循环在数据库中根据标识符类型和标识符查找是否已经存在，如果已经存在，则处理结束。

（6） 如果不存在，注册一个 BK 账户，即向数据库增加一条账户记录，返回新账户的 ID，将这个账户与新建的 Case 关联起来，并且将其设为主账户，将账户的名字和地址设为这个 case 的 member name 和 member address(通过 addFilledFacet 方法)并存到数据库。

（7） 将所有该 user Id 的标识符与 BK 账户关联起来，并更新到数据中。

（8） 成功创建 Case,导入过程结束，系统自动跳转到新 Case 页面。

- 从 Hardcopy Queue 导入 Case

即将 Hardcopy Queue 中每个可用的 Hardcopy 都导入到一个新的 Case，即创建一个新的 Case，每个 Hardcopy 导入的过程跟上述所说的从 Hardcopy 导入 Case 一样。

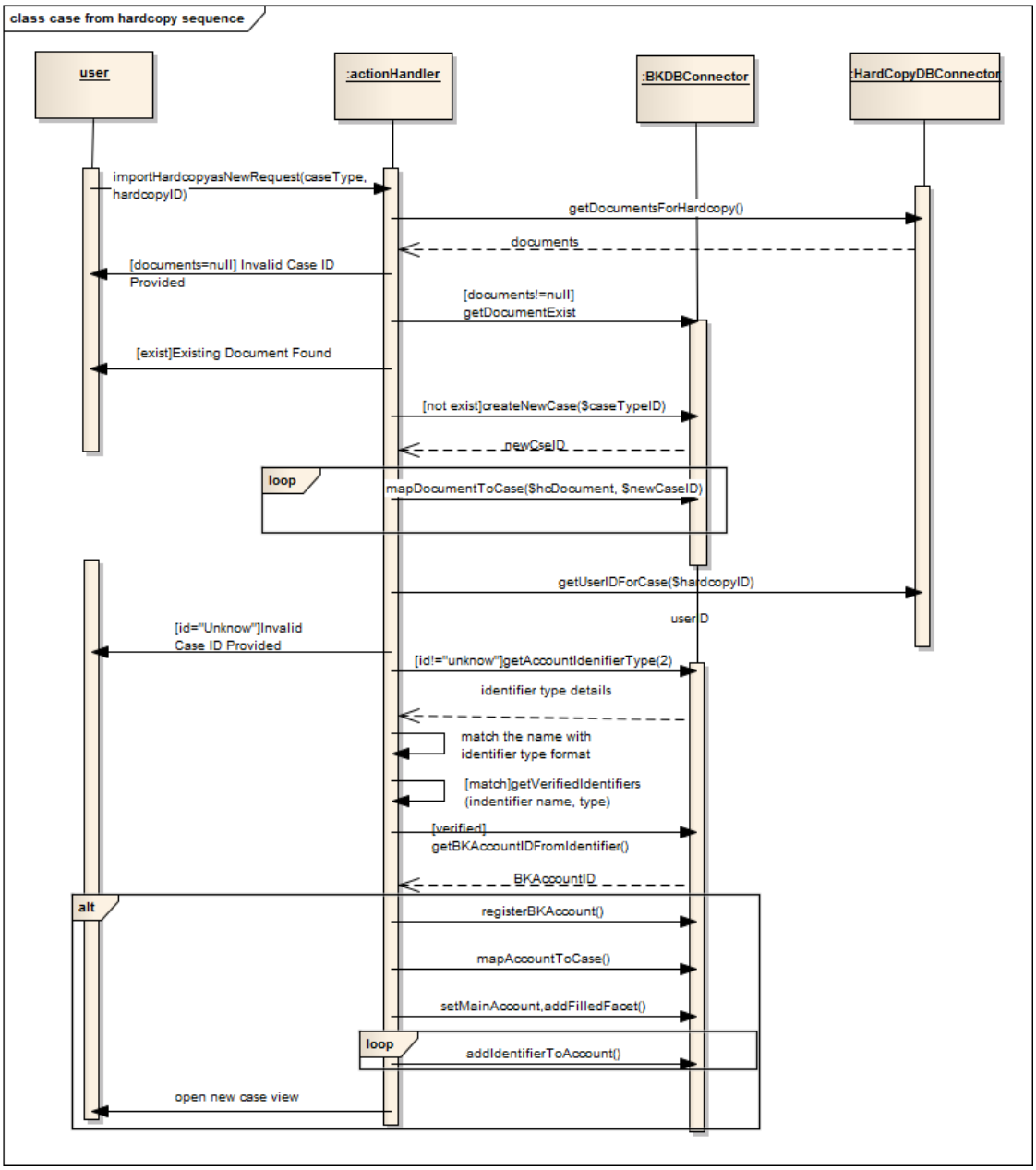

图 5- 4 从 Hardcopy 导入 Case 顺序图

5.2 Case 视图的实现

图 5-5 为某个 Case 视图，根据设计显示了 case 的详细信息（当前登录的用户为 Case 的 Owner，所有可以对信息进行修改）。其中上半部分的最左边一栏为每个 Case 都需要显示的内容（即 Internal ID, Region, Situation, Local Status, Case Owner）, Internal ID 为 Case 在数据库 cases 表中的 ID，直接从数据库读取，不可修改。

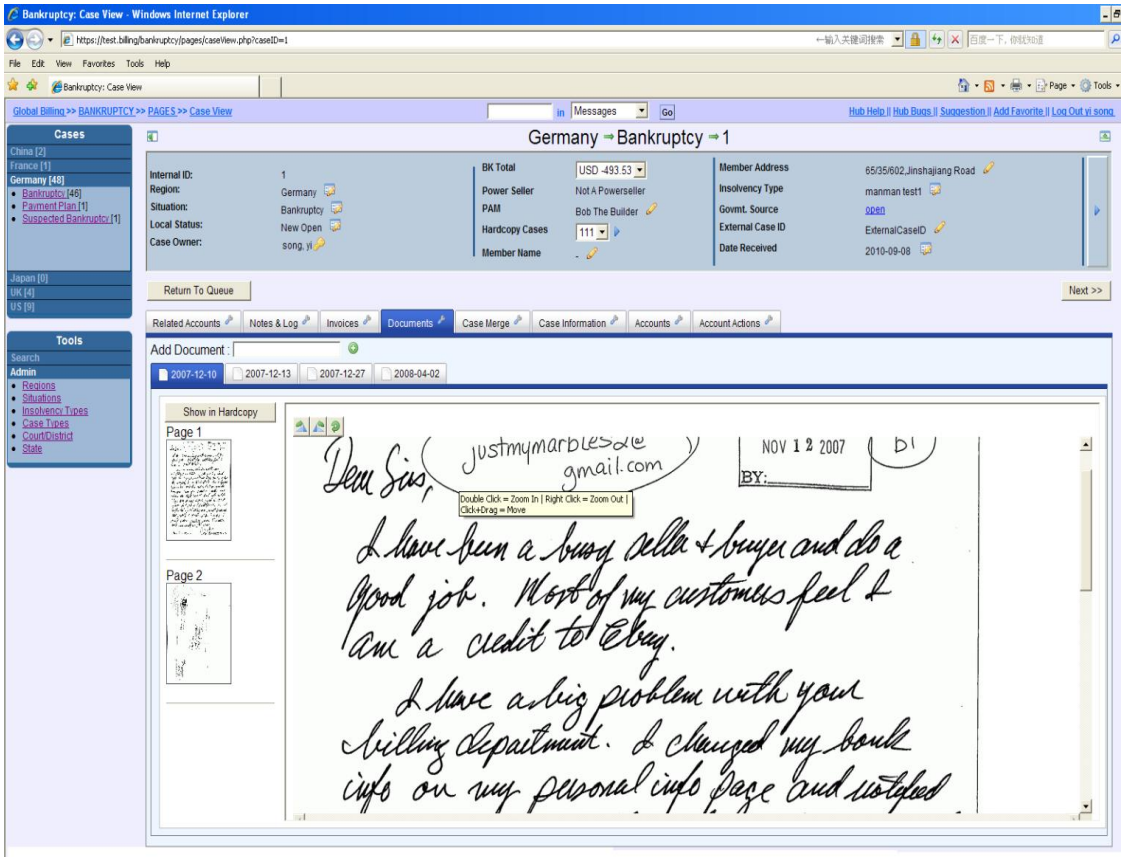

图 5- 5 Case 视图

页面上可修改的部分的按钮都是使用的封装好的类，修改时需要用户输入信息的部分均使用的 InlineInputEditor 类，比如 PAM，External ID 等信息。用户修改时需要从已有的数据的下拉框中选择数据的部分均使用的 InlineDropdownEditor 实现，比如 Local status, region, insolvency type， Situation。但是 Region 和 Situation 的修改与其他不同，当用户从任何一个下拉框中选取某数据提交修改时，系统都会先判断该 Case 类型是否存在，如果存在才能修改成功， 否则显示修改前的数据，并且返回”the selected combination of Region and Situation is not configured”的信息给用户。

除了每个 Case 都需要显示的信息数据外，其他的每一种数据信息都被封装在一个 Facet 类中，在显示的时候，用户需要从数据库读出该 Case 所属类型所需要显示的信息种类，然后显示在上面部分的信息调用类中的 getDetailInterface 的方法，然后通过 StackCarousel 类组合显示出来。具体代码如下：

代码中第一行调用类中 getCaseStaticDetailsView 将每个 Case 都必须显示的信息加到页面中,代码的第四行和第五行获取从数据库获取 Case 的详细信息和 Case 所有的 filledFacet 内容。通过从数据库取出的 filledFacet 的类的路径和类的名字创建相应的 facet 对象。接着判断该 filledFacets 是否为 CaseFacet 类的子类，

```
$staticTable = $this->getCaseStaticDetailsView($staticDiv, $caseID, $readonly);
$facetsCell = $detailsTable->getCell(0, 1);
$caseDetails = $this->dbConn->getCase($caseID);
//从数据库获得显示在 details 部分的 Facet
$detailFacets = $this->dbConn->getDetailFacetsForCase($caseID);
if (is_array($detailFacets)) {
                $detailFacetNodes = array();
                foreach ($detailFacets as $detailFacet) {
                    if (!empty($detailFacet->class_path)) {
                        include_once($detailFacet->class_path);
                        $facetClass = new $detailFacet->class_name($this->dbConn);
                    } else {
                        $facetClass = null;
                    }
                    if ($facetClass instanceof CaseFacet) {
//调用相应 facet 中的 getDetailInterface 的方法
                        $facetNode = $facetClass->getDetailInterface($caseID,
$detailFacet->facet_type_id,$detailFacet->filled_facet_id,$readonly);
                    } else {
                        $facetNode = new Table(null, 1, 2);
                        $facetNodeLabelCell = $facetNode->getCell(0, 0);
                        $facetNodeLabelCell->contentText = $detailFacet->detail_label;
                        $facetNodeDataCell = $facetNode->getCell(0, 1);
                        $facetNodeDataCell->contentText = $detailFacet->facet_data;
                    }
                    $detailFacetNodes[] = $facetNode;
                }
            }
            if (count($detailFacetNodes) > 0) {
          $facetDetailCarousel = new StackCarousel($facetsCell, $detailFacetNodes, 2, 5, $sizes);
            }
```

图 5- 6 case details view 实现

如果是，则调用 getDetailInterface 的方法，返回值赋值给$facetNode，如果不为 CaseFacet 类的子类，则将该 filledFacet 的 label 和 data 显示出来，赋值给$facetNode，第二十五行代码将$facetNode 放到$detailFacetNodes 数组中。第二十九行代码通过调用 StackCarousel 类将信息显示在页面的上半部分。

页面中下半部分选项卡的显示跟上半部分的实现类似，即获得从数据库获得所有的该 Case 的显示在选项卡上的信息，新建 Tabset 的类，然后逐个将通过 getTabInterface 获得的加到 tabset 上去，代码如下：

```
//从数据库读取需要显示在 tab 上的 facet
$tabbedFacets = $this->dbConn->getTabbedFacetsForCase($caseID);
$facetTabset = new TabSet($parent, "facetTabset" . $caseID);
$readonly = (($caseDetails->owner != $_COOKIE['BTSSN']) || ($caseDetails->merged == 1));
        if (is_array($tabbedFacets)) {
            foreach ($tabbedFacets as $tabbedFacet) {
                if (!empty($tabbedFacet->class_path)) {
                    include_once($tabbedFacet->class_path);
                    $facetClass = new $tabbedFacet->class_name($this->dbConn);
                } else {    $facetClass = null;     }
                if ($facetClass instanceof CaseFacet) {
                    $contentNode = $facetClass->getTabInterface($caseID,
$tabbedFacet->facet_type_id, $tabbedFacet->filled_facet_id, $readonly);
                } else {
                    $contentNode = new PageElement(null, "div", $tabbedFacet->facet_data);
                    $tabLabelContainer = new PageElement(null, "div");
                    $tabLabelSpan = new PageElement($tabLabelContainer, "span",
htmlentities($tabbedFacet->tab_label));
                    $facetTabset->addTab($tabLabelContainer, $contentNode, $firstTab,
"facetTab" . $tabbedFacet->facet_type_id);
                }
            }
        }
```

图 5- 7 case tab view 实现

5.3 Case 视图配置的实现

图 5-6 中为配置某 Case 属性以及视图的页面，配置页面的左边为配置该 Case 类型所关联的 Hardcopy 队列，SLA，以及状态，右边为配置 Case 视图中上半部分显示的 detailed Facets,通过选择 Availble Facets，然后点击右移按钮，点击保存即可，选择的顺序即为显示的顺序。对于显示在下半部分选项卡中的内容也如此（Tabbed Facets）。

具体实现如下：

首先从数据库获得所有的 Facets，放到 availableDetailFacetTypes 的数组里去，再从数据库获得当前 case 类型所当前需要显示的 Facets 放到 selectedDetailFacetTypes，并在 availableDetailFacetTypes 数据中出现的删除掉。创建一个 Form 类，通过 availableDetailFacetTypes 和 selectedDetailFacetTypes 创建 TwoListSwapper 类显示页面中的效果。

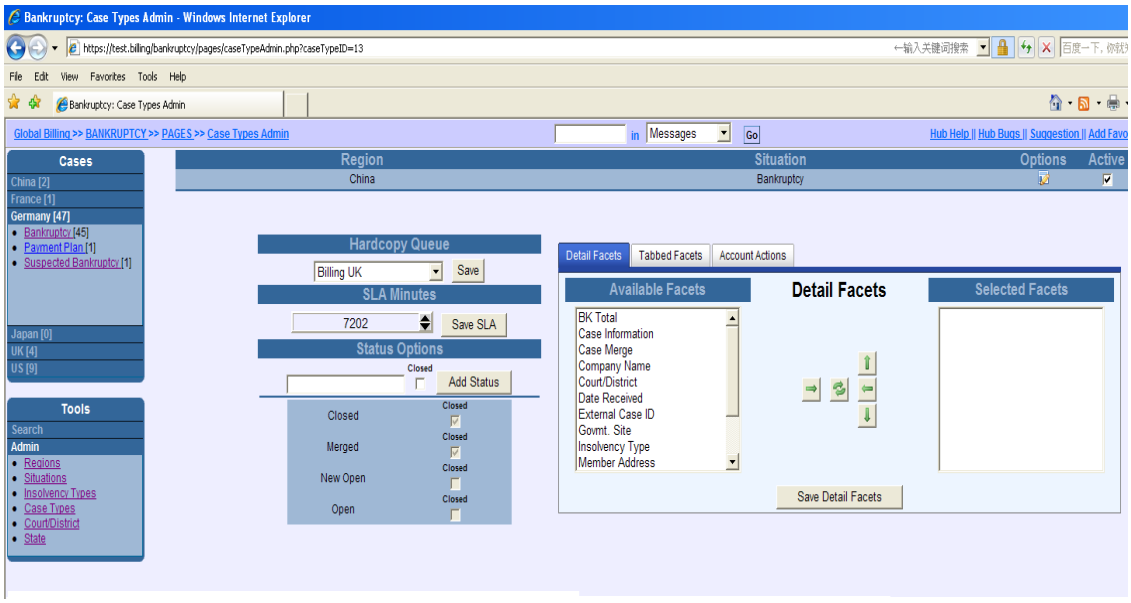

图 5- 8 配置 Case 视图

```
$facetTypes = $this->dbConn->getFacetTypes();
$availableDetailFacetTypes = array();
if (is_array($facetTypes)) {
        foreach ($facetTypes as $facetType) {
                if (!empty($facetType->detail_label)) {
                        $availableDetailFacetTypes[$facetType->facet_type_id]=
$facetType->detail_label;
                }
        }
        asort($availableDetailFacetTypes);//排序
}
$detailFacetTypes = $this->dbConn->getDetailFacetsForCaseType($caseTypeID);
$selectedDetailFacetTypes = array();
if (is_array($detailFacetTypes)) {
        foreach ($detailFacetTypes as $facetType) {
                unset($availableDetailFacetTypes[$facetType->facet_type_id]);
                $selectedDetailFacetTypes[$facetType->facet_type_id] = $facetType->detail_label;
        }
}
$detailFacetSwapper = new TwoListSwapper($detailFacetForm, $availableDetailFacetTypes,
$selectedDetailFacetTypes, "availableDetailFacets", "selectedDetailFacets");
//将不需要的 button 移除
$detailFacetSwapper->leftMoveUpButton->parent->removeChildNode($detailFacetSwapper->l
eftMoveUpButton);
$deailFacetSaveButton = new PageElement($detailFacetsDiv, "button", "Save Detail Facets");
```

图 5- 9 case view 配置实现

TwoListSwapper 类的实现代码：

```
class TwoListSwapper extends NodeContainer {
public $leftList;    // The left had select element
public $rightList; // The right hand select element
public $moveToLeftButton; // The button that moves things from the right table to left table
public $moveToBothButton;
public $rightMoveUpButton; // The button that moves selected items up in the right list
public $moveToRightButton; //The button that moves things from the left table to right table
function __construct($parent, $leftOptions = null, $rightOptions = null, $leftInputName =
"leftList", $rightInputName = "rightList", $onChange = ""){
$this->leftList = new Select($leftListCell, $leftOptions);
$this->leftList->addAttribute("size", "10");
$this->leftList->addAttribute("multiple", "true");
$this->moveToRightButton = new PageElement($controlTable->getCell(1, 0), "button");
$this->moveToRightButton->addAttribute("title", "Move the selected items on left to the right.");
$this->moveToRightButton->addAttribute("onclick",
                                "moveListValues(document.getElementById('$leftListID'),
                                document.getElementById('$rightListID'));
                                updateListInput(document.getElementById('$leftListID'),
                                document.getElementById('$leftInputID'));
                                updateListInput(document.getElementById('$rightListID'),
                                document.getElementById('$rightInputID')); $onChange");
}
function updateListInput(list, input){
        var values = new Array();
        for(i = 0; i < list.options.length; i++){
            values.push(list.options[i].value);
        }
        input.value = values.join(",");
    }
function moveListValues(listFrom, listTo){
        for(i = (listFrom.options.length - 1); i >= 0; i--){
            if(listFrom.options[i].selected){
                var optionToMove = document.createElement('option');
                optionToMove.text = listFrom.options[i].text;
                optionToMove.value = listFrom.options[i].value;
                listTo.add(optionToMove);
                listFrom.remove(i);
            }
        }
    }
```

图 5- 10 TwoListSwapper 实现

在本页面中的 TwoListSwapper 页面上，只用到三个按钮，左移，右移和对换按钮，页面中的元素都作为该类的属性，并且在构造函数中进行初始化设置，上述代码中列出来了左边列表的初始化代码（16-20 行代码）以及右移按钮的初始化过程（21-24 行代码）。当点击左移或者右移按钮时候，调用 JavaScript 的 moveListValues 函数，对 listForm 里面的 option 进行循环如果选中的则放到 listTo 中，然后调用 updateListInput 方法对左右两个 list 进行数据的更新。

5.4 合并 Case 的实现

Case 合并的条件是 Case 没有合并过或者 Case 合并之后是父 Case，即数据库表 cases 中的 MERGED 属性一栏为 0。

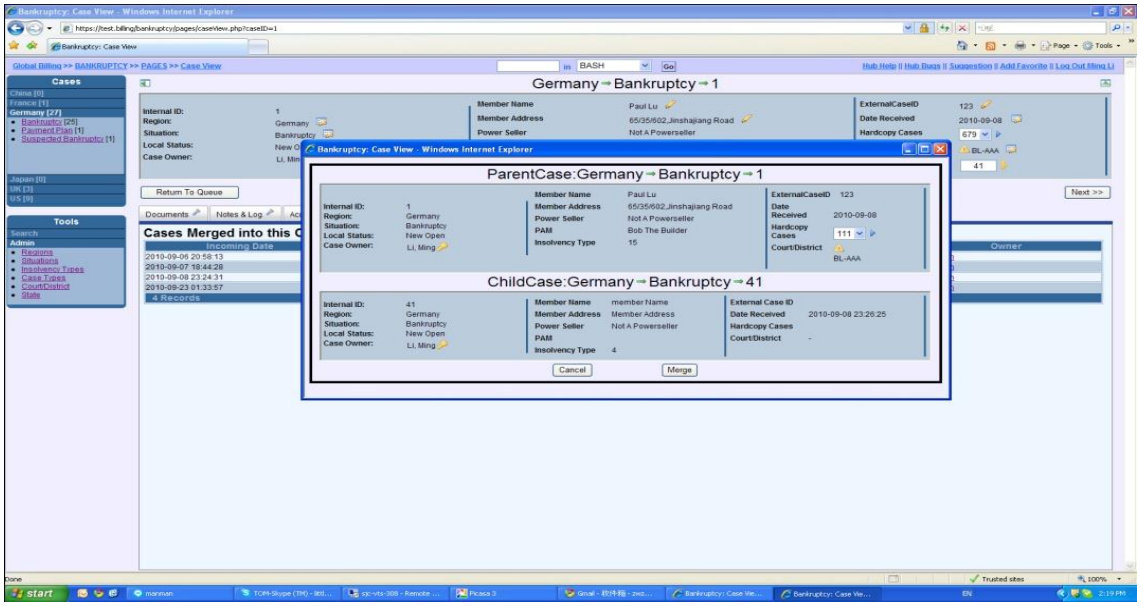

图 5- 11 Case 合并视图

如图 5-7 所示满足条件的 case 的 caseDetails 中，合并一栏会出现输入框，输入需要合并的另一个 Case 的 ID。如果另一个 case 也符合上面提到过的条件，会弹出一个确认窗口如上图，显示两个案件分别的 caseDetails。合并之后的 case 视图的 case merge 选项卡会显示其子案件的信息。

代码如下：

```
function confirmMerge($post) {
    if (!empty($post['mergeCaseID'])) {
        $mergeCaseID = $post['mergeCaseID'];
        $mergeCaseDetails = $this->dbConn->getCase($mergeCaseID);
        $caseID = $post['caseID'];
        $caseDetails = $this->dbConn->getCase($caseID);
        $caseOwner = $post['caseOwner'];
        if (($caseOwner == $mergeCaseDetails->owner) && ($caseID != $mergeCaseID)
&& ($mergeCaseDetails->case_type_id == $caseDetails->case_type_id)) {
            $mergeCaseTime = strtotime($mergeCaseDetails->create_stamp);
            $caseTime = strtotime($caseDetails->create_stamp);
            if ($mergeCaseTime < $caseTime) {
                $parentCaseID = $mergeCaseID;
                $childCaseID = $caseID;
            } else {
                $parentCaseID = $caseID;
                $childCaseID = $mergeCaseID
            }
$this->printPopupOpen("/bankruptcy/pages/mergeCaseView.php?childCaseID=$childCaseID
&parentCaseID=$parentCaseID", "Merge Case $childCaseID To Case $parentCaseID");
        } else if ($caseOwner != $mergeCaseDetails->owner) {
            $this->printIFrameFailure("You must be the owner of both cases to merge
them"); } else if ($mergeCaseDetails->case_type_id != $caseDetails->case_type_id) {
            $this->printIFrameFailure("Please enter the cases with the same case type");
        } else {
            $this->printIFrameFailure("Wrong caseID input");
        }
    }
}
```

图 5- 12 确认 case 合并实现

当在 merge 的输入框中输入想要 merged 的 case id 并提交之后，会先判断当前用户是否为 case 的 owner，所输入的 case id 是否和当前 case id 相同，需要合并的 case 是否是一样的类型。然后比较数据库中两个 case 的创建时间，创建时间早的 case 为父 case，然后通过 printPopupOpen 方法弹出一个新的页面。页面中有两个 case 的 details 信息。当用户确定 Case 合并之后，就会执行 actionhandler 中的 mergeCase 函数。mergeCase 函数会调用实现自接口 casefacet 的类中的 mergeCase 函数。

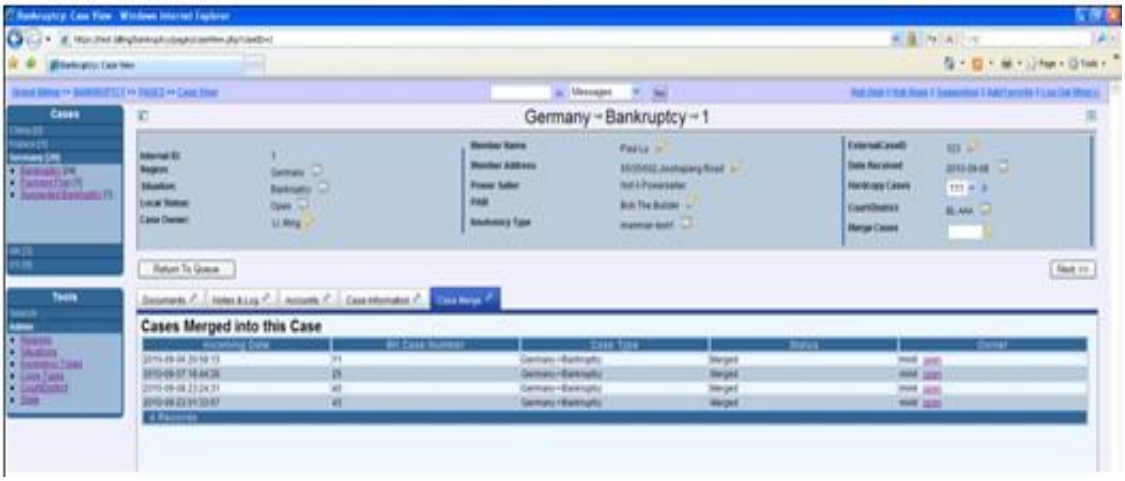

图 5- 13 父 Case 视图

最后转向到父案件页面如图 5-8 所示在合并中为父 Case 的 tab 页面会显示其子 Case 的信息，而子 Case 页面的 detail 里也会提供往父 Case 的链接。代码如下：

```
function mergeCase($post) {
    $parentCaseID = $post['parentCaseID'];
    $childCaseID = $post['childCaseID'];
    $oldUrl = $post['url'];
    if (!empty($parentCaseID) && !empty($childCaseID)) {
        $facetDetails = $this->dbConn->getFacetTypes();
        foreach ($facetDetails as $facetDetail) {
            if (!empty($facetDetail->class_path)) {
                include_once($facetDetail->class_path);
                $facetClass = new $facetDetail->class_name($this->dbConn);
            } else {
                $facetClass = null;
            }
            if ($facetClass instanceof CaseFacet) {
                $facetClass->mergeCaseFacets($childCaseID,$parentCaseID,
$facetDetail->facet_type_id);//合并信息
            }
        }
        $this->dbConn->mergeCase($childCaseID, $parentCaseID);
        $url = "/bankruptcy/pages/caseView.php?caseID=$parentCaseID";
        $this->printParent($url, $oldUrl);//
    }
}
```

图 5- 14 case 合并实现

5.5 案件备注的实现

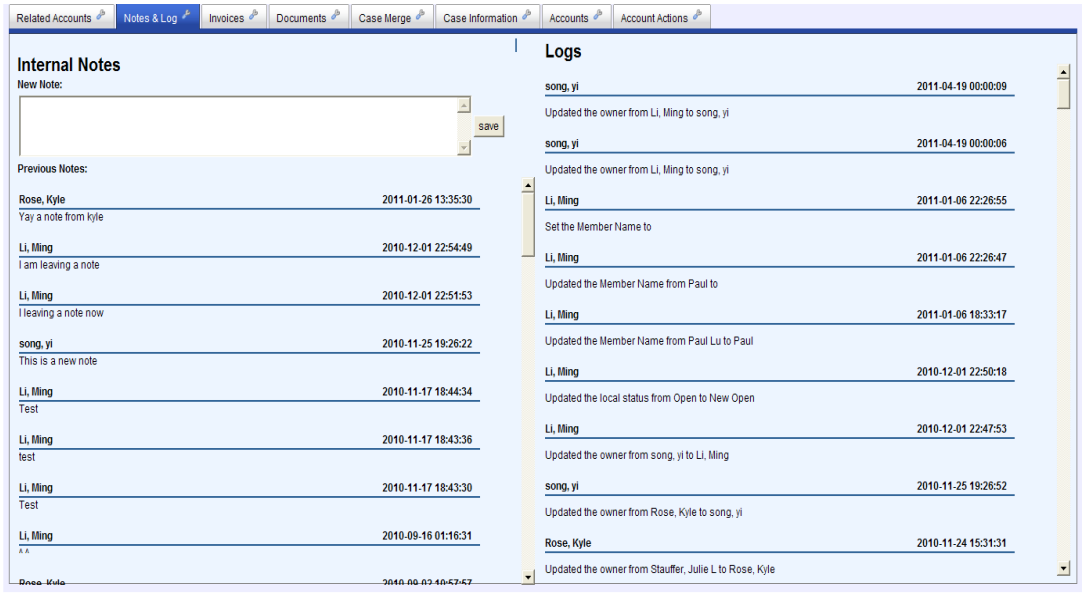

图 5- 15 案件备注视图

如图 5-9 所示在案件备注标签的左边，用户可以在一个 HTML textarea 元素中为案件做内部注释，并提交，所做的注释会实时显示在输入框下方的列表中。显示注释的 div 被放入了一个 AjaxNode[13]。请求将会被发送到文件 NotesAndLogFacet.class.php，并返回下图所示 previous notes 的 HTML 代码[12]，这样可以使用户在不用刷整个页面的情况下，看到先前写下的案件注释。

```
$notesHolder = new AjaxNode($noteDisplay, "/bankruptcy/classes/facets/NotesAndLogFacet.class.php?ajaxRequest=displayNotes&caseID=$caseID", true);
```

系统总共记录了三种种不同的操作记录，包括 filled facet 更改记录，case 状态更改记录及 Case 的 owner 更改记录会被显示在页面上。这些操作记录是用户在处理 Case 的过程中，修改了某些 Case 的信息而被同时加入到数据库中的。操作记录会显示修改时间，修改人及修改前后的值，既可以避免了误操作也可以提高系统的安全性。

我们首先用数据库方法 getLoggedChanges($caseID)获得这三种不同记录合并而成的数组，并按记录的时间先后排序。根据记录类型的不同，在页面上显示更改记录的方式也不同。有些类型的信息可能需要处理过再输出，有些则不需要。在显示 filled_facet 更改的记录时，会根据 facet 类型的不同读取其相对应的类中的 getChangeLogDisplay 方法来显示更改记录。

5.6 系统的优化

由于 case view 等页面中的信息量不断增大，慢慢地，各种 facets 为了取得当前值执行数据库中的函数的次数越来越多，从数据库中读取数据的速度需求也会越来越高。所以我们加入了缓存技术，使得当需要取得数据时，就可以直接读取先前数据库语句执行后的结果，从而加快读取速度。

在数据库类中，我们使用了 Memcache 和 全局变量 作为二级缓存。当要从数据库中读取一个数据时，首先从所实例化的数据库对象对应的全局变量中查找，如果找到就立即读取并返回；如果没有找到，就从 Memcache 中读取，同时把这个数据赋值给对应的全局变量；如果 Memcache 中也没有，就执行数据库语句，并把执行得到的数据存储在 Memcache 的服务器缓存中，同时也赋值给对应变量。

Memcache 类表示连接到一个服务器组的连接。 Memcache 模块提供了于 memcached 方便的面向过程及面向对象的接口，memcached 是为了降低动态 web 应用 从数据库加载数据而产生的一种常驻进程缓存产品。

在 dbconnector 类构造函数中，实例化了一个 Memcache 对象。Memcache::connect()建立一个到 memcached 服务端的连接，memcached 服务端监听主机地址为 127.0.0.1 也就是本机，监听端口为 11211。使用方法 Memcache::connect()打开的连接在脚本执行结束后会自动关闭。

```
function __construct() {
$this->cacheConn = new Memcache;
$this->cacheConn->connect("127.0.0.1", 11211);
$this->dbConn = new DBConnector("aging");
$this->dbConn->setErrorTrapping(true);
}
```

图 5- 16 memcached 初始化实现

将纯粹使用数据库查询的代码加上 memcached 支持是很简单的，假设这是原来的代码[14]：

```
function getRegions() {
      $sql = "SELECT * FROM bankruptcy.regions ORDER BY `NAME`";
      return $this->dbConn->getArrayOfRows($sql);
}
```

使用 Memcache 对象方法，加上 memcached 的高速缓存机制和依靠全局变量作为缓存的机制[15]后：

函数首先判断 regions 变量中是否储存了数组，如果存在，则直接返回。反之，判断服务端之前有以"BKDBConnector.getRegions 作为 key 存储的元素，

Memcache::get()方法此时返回之前存储的值。当想要获得的数据再两级缓存中都没有找到时，就执行 SELECT 语句，并把获得的数组存入缓存中。

```
function getRegions() {
    if (empty($this->regions)) {
        $regionsCache = $this->cacheConn->get("BKDBConnector.getRegions");
        if ($regionsCache === false) {
            $sql = "SELECT * FROM bankruptcy.regions ORDER BY `NAME`";
            $this->regions = $this->dbConn->getArrayOfRows($sql);
            $this->cacheConn->set("BKDBConnector.getRegions", $this->regions, 0, 600);
        } else { $this->regions = $regionsCache; }
    }
    return $this->regions;
}
```

图 5- 17 memcached 设置实现

Memcache::set()向 key "BKDBConnector.getRegions"存储全局变量即第一级缓存$this->regions 数组的元素值，并设定失效时间为 600 秒，设置合理的失效时间可以节约出更多的缓存使用空间。

在 memcached 内已经有高速缓存信息时将数据库的数据更新后，上述的程序会抓到旧的数据。其中一种解决的方法是在更新数据库时，删除 memcached 内的信息。比如当对 regions 表做了 add 操作的时候：

```
function addRegion($newRegionName) {
$newRegionName = array(trim($newRegionName));
$sql = "INSERT IGNORE INTO bankruptcy.regions (`NAME`) VALUES ( ? )";
$this->cacheConn->delete("BKDBConnector.getRegions");
return $this->dbConn->executeSingle($sql, $newRegionName, true, 'regions', 'region_id');
}
```

图 5- 18 memcached 修改实现

Memcache::delete()函数会通过 key，即"BKDBConnector.getRegions"，删除之前储存的 regions 数组元素，来保持 cache 和数据库中数据的一致性。

另一种解决的方法是在更新数据库时，同时更新 memcached 内的信息，这时可以用到。Memcache::replace()方法:Memcache::replace()通过 key 来查找元素并替换其值。当 key 对应的元素不存在时，Memcache::replace()返回 FALSE。

5.7 本章小结

这一章主要对 Bankruptcy 系统的实现细节进行了介绍，从各个页面以及相应的处理流程和代码实现进行了详细的阐述。首先详细介绍了系统导入新 Case 的流程，然后从 Case 视图，配置 Case 视图，合并 Case, note&log 选项卡依次介绍了系统功能的实现，最后，介绍了从使用缓存技术来优化系统。

第六章 总结与展望

6.1 本文总结

本文首先介绍了项目背景和 Bankruptcy 系统的设计目的，他主要是为了帮助 eBay 工作人员跟踪管理 Case。

然后本文介绍了系统的设计和实现中过程中所涉及到的相关技术,包括 PHP, AJAX 技术，为了加快页面的访问速度所采用的 Memcached 缓存技术以及设计时采用的 MVC 设计模式。

随后根据需求从用例，功能方面对系统进行了详细的分析并提出了系统的架构设计，系统采用 MVC 架构思想。接着本文从数据库的实体关系，表结构和接口设计，页面设计详细阐述了系统的设计。

最后，本文阐述了整个系统的实现以及优化，具体分析了 Case 创建的流程，和主要页面的实现，以及如何加快页面的访问速度。

6.2 项目展望

在 bankruptcy 系统的后续版本中，系统的业务需求还会继续扩大，如设置了 Case 类型的 SLA 时间,Case 如果超过设定的时间会自动关闭，在快到设定时间时候，系统会自动提醒工作人员，还有 Account Action 的实现。由于本系统采用 MVC 的架构设计，以及在 Case 视图部分使用了 Facet 接口的设计，所以在系统上增加新的功能很方便。

另外，随着数据量的加大，需要提高系统的性能，加快页面的响应速度，以提供更好的用户体验。所以在后续的版本中将会优化数据库，提高数据读取的效率，优化页面。

参考文献

[1] Peter Moulding. PHP BLACK BOOK.1st ed. USA:CORIOLIS, 2003.

[2] Various Licenses and Comments about Them. Free Software Foundation，http://www.gnu.org/licenses/index_html#GPLIncompatibleLicenses

[3] Rick Strahl. An Introduction to AJAX Technology [J]. FoxPro Advisor, 2007, 15 (1) _6 .

[4] 游丽贞,郭宇春,李纯喜等. Ajax 引擎的原理和应用 [J]. 微计算机信息, 2006, 22 (6) _3 .

[5] Rebecca Riordan. Head first Ajax.1st ed.USA: O'Reilly,2008.

[6] REUVEN M.LERNER. Memcached [J]. Linux Journal, 2008, 0 (176) _4 .

[7] Brad Fitzpatrick. Distributed caching with memcached[J]. 2004,124:5.

[8]IIS, http://www.microsoft.com/china/windowsserver2003/techinfo/overview/iis.mspx

[9] 刘艳锋. MVC 设计模式的分析与应用 [J]. 科技传播, 2010, (22) .

[10] 刘亮,霍剑青,郭玉刚等. 基于 MVC 的通用型模式的设计与实现 [J]. 中国科学技术大学学报, 2010, 40 (6) . DOI:10.3969/j.issn.0253-2778.2010.06.016.

[11] Chris Fone. From API 682 1st Edition to the ISO 1049 standard [J]. Sealing Technology, 2004, 0 (11) _4 . DOI:10.1016/S1350-4789(04)00411-8.

[12] Bogdan Brinzarea,Cristian Darie. AJAX and PHP: Building Modern Web Applications.2nd ed.USA： Packt Publishing，2010.

[13] Smith, K.. Simplifying Ajax-style Web development [J]. Computer, 2006, 39 (5) _4 . DOI:10.1109/MC.2006.177.

[14] Larry Ullman.PHP 6 and MySQL 5 for Dynamic Web Sites[M].USA：Peachpit Press，2008.

[15] 周淦淼,谭石强. 基于 Memcached 的 MySQL 查询优化 [J]. 现代计算机(专业版), 2009, (5) _4 .

致谢

在论文完成之际，我要向所有帮助，支持过我的人表示最诚挚的感谢。

首先，我要感谢我的指导老师刘海涛老师的热情关怀和悉心指导。感谢刘老师在实习期间给予了我很多的关怀。在我撰写论文过程中，刘老师倾注了很多的心血和汗水，从选题到提纲，材料，修改以及到最后的定稿，刘老师都给予了我很多的教诲和帮助。在这期间，老师严谨的的治学精神和一丝不苟的的工作作风使我终生受益。

我要感谢软件学院给我提供的良好的学习环境和先进的设备，感谢学院的每个老师在学习和生活上对我的关心，老师的知识的教授让我在本科期间打下扎实的专业知识基础。感谢所有的同学给我在学习和生活上的关怀和帮助，从他们身上学习了很多。

我要感谢 eBay 上海研发中心以及美国总部所有的同事和领导，他们在工作，学习上给予我的无私的帮助和支持。不管是在工作还是论文上，他们都很热情地帮助我，无论遇到什么困难，他们都很耐心的给我解答。总之，在实习期间，我学习到了很多知识，受益匪浅。

我要感谢我的家人，是他们在背后默默的支持和付出，给足我勇气从大一走到大四，克服每一个困难，他们是我坚强的后盾，正是他们无私的付出才使我有时间和精力完成学业。

总之，我要衷心的感谢所有帮助过我，关心过我的人，千言万语汇成一句话谢谢你们，衷心祝愿你们健康幸福！同时，向评阅本论文以及答辩委员会的各位老师致以深深的感谢，感谢你们所付出的辛勤的工作。