# 小组成员

|  |  |
| --- | --- |
| 姓名 | 学号 |
| 肖瑞 | 131250002 |
| 张亚飞 | 131250023 |
| 赖铭津 | 131250032 |
| 郭玥 | 131250033 |
| 张怡 | 131250036 |
| 曾春晖（PM） | 131250049 |
| 许露佳 | 131250104 |
| 雷莳芳 | 131250107 |

# 模块分工

|  |  |
| --- | --- |
| 模块 | 设计者 |
| GoodsSearch | **许露佳** |
| GoodsSort | **许露佳** |
| Shield | **雷莳芳** |
| PriceParity | **雷莳芳** |
| SafeGuard | **肖瑞** |
| Parttake | **肖瑞** |
| Crawler | **张怡** |
| CrawlerDataUpdate | **张怡** |
| CrawlerDataAnalyser | **张亚飞** |
| CompatibilityHandler | **张亚飞** |
| PriceCompatibility | **张亚飞** |
| LanguageCompatibility | **张亚飞** |
| ScaleCompatibility | **张亚飞** |
| BrowerAdapter | **曾春晖** |
| CommentCheck | **曾春晖** |
| Pay | **赖铭津** |
| Advertisement | **赖铭津** |
| DataBase | **郭玥** |
| 数据库表设计 | **郭玥** |

# 1.GoodsSearch模块设计

## 1.1概述

### 1.1.1承担的需求（主要秘密）

根据用户输入的关键字与数据库中的商品信息进行快速匹配，返回搜索的结果

本模块需要对输入的关键字进行预处理，将一些无用的词汇删除，如英语的a、an、the，中文的了，啊，呢，吧，哦等等

本模块需要将输入的关键词在同义词表中找出相对应的同义词，并根据同义词进行搜索，得到检索列表

涉及需求：

R2 系统高可靠性 在用户输入有误的时候能够及时反馈，并且仍然能够正常运行

R11 浏览器兼容性 兼容Firefox版本号>=3.5;兼容Chrome版本号>=3;兼容IE版本号>=6;兼容Opera版本号>=10.5;兼容Safari版本号>=3.1

R25 高效性搜索 搜索响应时间不超过2s

R27 搜索结果排序 允许使用竞价规则定制排序策略

### 1.1.2可能会修改的实现（次要秘密）

同义词匹配算法

检索匹配算法

输入词汇预处理算法

### 1.1.3角色

检索功能

### 1.1.4对外接口

## 1.2类图

## 1.3类描述

（类方法（重要方法要给出前置与后置条件）与重要数据结构描述）

## 1.4重要协作

（顺序图与协作描述）

## 1.5使用的设计模式

（使用场景；所用的设计模式；要达到的效果）

# 2.GoodsSort模块设计

## 2.1概述

### 2.1.1承担的需求（主要秘密）

用户输入关键字并向系统提交查询请求后，系统在查询结果页面根据如下的排序规则对结果进行排序并展示给用户：

根据用户输入的关键词的相关度进行排序，关键字越吻合，相关度越大，排名越靠前

根据来源的商城的可靠度进行排序，商城知名度越大，关键字相关度一致的条件下排名越靠前

允许使用竞价排序规则调整排序

涉及需求：

R27 搜索结果排序 允许使用竞价规则定制排序策略

商品排序用例图：



用例描述：

1. 计算相关度

GoodsSort模块根据传入的关键词和商品列表计算商品列表的每一项与关键词的匹配程度

1. 计算可靠来源

GoodsSort模块根据每一项商品信息的来源计算其来源的可靠程度

1. 竞价规则

GoodsSort模块根据竞价规则对商品信息列表进行排序

### 2.1.2可能会修改的实现（次要秘密）

排序算法

竞价规则排序策略

### 2.1.3角色

计算相关度：根据所获得的关键词与关键词列表进行匹配，通过特定的算法计算获得的关键词与关键词列表中每一项的信息的相关度

计算来源可靠度：对与满足相关度要求的商品信息，根据网站来源等信息来计算可靠度

竞价排序：根据竞价规则对得分的各部分进行加权，获得最后的排序列表

### 2.1.4对外接口

与GoodsSearch模块之间的接口:

Public List<Goods> getSort(List<Goods> goodslist,String keyword);

与数据库之间的接口:

Public List fetchSourceList();

## 2.2类图



## 2.3类描述

### 2.3.1 SortFilter类

该类为GoodsSort模块和其他模块交互的主入口，调用各个排序规则以实现搜索结果的排序

类方法

|  |  |
| --- | --- |
| 类方法 | 描述 |
| Public List<Goods> getSort(List<Goods> goodslist,String keyword); | 概述：本方法为外界交互的接口，用于获取排序后的商品信息  前置条件： keyword合法,goodslist不为空  后置条件： 获得排序后的商品信息列表 |

数据结构描述

|  |  |
| --- | --- |
| 数据结构 | 描述 |
| List sortRules | SortRules集合 |

### 2.3.2 SortRules接口

本接口用于封装进行排序的计算规则，具体算法由子类实现，包括Relevancy，ReliableSource和Bidsort

### 2.3.3 Relevancy类

该类计算商品信息列表与关键字的匹配程度并进行排序

类方法

|  |  |
| --- | --- |
| 类方法 | 描述 |
| Public List<Goods> sort(List<Goods> goodslist,String keyword); | 概述：根据getRelevancy(Goods good) 计算出的相关度进行排序  前置条件： keyword合法，goodslist不为空  后置条件： 获得排序后的商品信息列表 |
| Private double getRelevancy(Goods god) | 将商品信息与关键字进行比较，计算出相关度 |

数据结构描述

|  |  |
| --- | --- |
| 数据结构 | 描述 |
| String keywords | 用户输入的搜索关键字 |

### 2.3.4 ReliableSource类

该类用于计算商品来源的可靠度

类方法

|  |  |
| --- | --- |
| 类方法 | 描述 |
| Public List<Goods> sort(List<Goods> goodslist,String keyword); | 概述：根据getSourceScore(String url)计算得到的来源可靠度对商品信息列表进行排序并返回  前置条件： goodslist不为空  后置条件： 获得排序后的商品信息列表 |
| Private double getSourceScore(String url) | 根据url的来源计算商品的可靠度 |
| Public List fetchSourceList() | 从数据库中获取sourceList |

数据结构描述

|  |  |
| --- | --- |
| 数据结构 | 描述 |
| List sourcetList | 来源网站及可靠度信息列表 |

### 2.3.5 BidSort类

该类用于根据竞价规则对商品列表进行排序

类方法

|  |  |
| --- | --- |
| 类方法 | 描述 |
| Public List<Goods> sort(List<Goods> goodslist,String keyword); | 概述：根据竞价规则bidRules对商品列表进行排序  前置条件： goodslist不为空  后置条件： 获得排序后的商品信息列表 |

数据结构描述

|  |  |
| --- | --- |
| 数据结构 | 描述 |
| BidRules bidRules | 所选择的竞价规则 |

### 2.3.6 BidRules接口

本接口用于封装竞价规则，具体算法由子类实现，目前有ConcreteRules

### 2.3.7 ConcreteRules类

该类为一个具体的竞价规则的实现类，根据具体的竞价规则对商品列表进行竞价排序

类方法

|  |  |
| --- | --- |
| 类方法 | 描述 |
| Public List<Goods> sortByBid(List<Goods> goodslist); | 概述：根据制定的竞价规则商品列表进行排序  前置条件： goodslist不为空  后置条件： 返回根据竞价规则排序后的商品信息列表 |

## 2.4重要协作



协作描述

商品搜索模块从数据库获取到商品信息列表后，将该列表传入本模块，有sortFilter对商品列表进行排序。SortFileter先通过Relevancy对商品和关键词进行相关度的匹配并根据相关度大小进行排序；再调用ReliableSource对商品信息的来源进行进行可靠性排序；后通过BidSort对商品信息列表进行根据竞价规则的排序。

## 2.5使用的设计模式

### 2.5.1策略模式

**使用场景**

1) 商品排序使用的策略可能发生变化

2) 商品的竞价规则可能发生变化

**在类图中的体现**

****

****

**要达到的效果**

策略模式可以在不修改原有排序策略和竞价规则的前提下增加新的策略（规则），并使得这些策略可以相互替换

# Shield模块

## 概述

### 承担的需求（主要秘密）

商品屏蔽模块的职责是屏蔽与搜索关键字匹配的商品列表中的负面商品，因为负面商品可以灵活的添加和修改，且屏蔽规则必须符合法律法规，故要求屏蔽规则具有高度的可修改性，可扩展性和灵活性。

**质量属性要求**：商品搜索要求具有很高的可修改性、可扩展性，而变更的部分主要是待屏蔽商品信息的变化。因此该模块的设计就要着重考虑词库的可扩展性和可修改性，可以灵活变更又不影响其他模块的调用。同时，由于搜索返回结果往往比较庞大，待屏蔽商品库也较庞大，因此商品过滤效率也直接影响到用户体验，需要尽量采用高效的过滤算法。用户总共进行商品查询的时间不能多于2s。

**商品屏蔽用例图**



**用例描述**

1. 负面商品信息增删改

系统管理员可以在后台管理界面灵活地增加、修改、删除负面商品。

1. 进行链接过滤

当一个商品被加入负面商品信息表中后，网页上指向该商品的连接必须失效，可以通过url内容比对结合访问控制来实现。

1. 进行内容过滤

当用户输入搜索关键字后，系统将检测搜索到的商品列表，并屏蔽其中的负面商品。

可能会修改的实现（次要秘密）

负面商品信息表

商品过滤规则

**角色**

**向外提供接口**

|  |  |
| --- | --- |
| **接口名** | **职责** |
| Public GoodsList search(Keywords keywords) | 根据输入的关键词，先调用底层模块找到符合关键词的商品列表，再进行过滤，返回符合规则的商品列表。 |
| Public URL filter(URL goodsURL) | 根据输入的URL判断是否指向被屏蔽的商品，若被屏蔽，则重定向到另一个URL，否则返回原URL。 |
| Public void addShieldGoods(Goods goods) | 增加一条负面商品信息 |
| Public void updateShieldGoods(Goods goods) | 修改一条负面商品信息 |
| Public void deleteShieldGoods(Goods goods) | 删除一条负面商品信息 |

## 类图

## 类描述

### XX类

|  |  |
| --- | --- |
| 方法名 |  |
| 前置条件 |  |
| 后置条件 |  |
| 作用 |  |

## 重要协作

## 设计模式

# PriceParity模块

## 概述

### 承担的需求（主要秘密）

商品比价模块的职责是对同一个商品，比较其在不同购物网站上的价格，并可以按照价格进行升序和降序排序。要求可以灵活地定义参与比价的网站，在未来可能会增加不同的价格计算规则，因此比价模块应该具有高度的可修改性，可扩展性和灵活性。

**商品比价用例图**



**用例描述**

1. 展示不同购物网站上的商品价格

用户查看某一商品详细信息时，系统应显示该商品在不同网站上的价格，且用户可以自定义参与比价的购物网站（从本系统提供的购物网站中选择）。

1. 根据价格排序

用户在查看某一商品时，可以点击“按价格排序”，系统将不同购物网站按商品价格升序或降序排序。

可能会修改的实现（次要秘密）

参与比价的购物网站。

价格计算规则，如是否包括运费，是否可以使用优惠券等等。

**角色**

**向外提供接口**

|  |  |
| --- | --- |
| **接口名** | **职责** |
| Public GoodsPriceList findNamesake(Goods goods) | 根据该商品查找不同购物网站上的同名商品，返回同名商品列表 |
| Public GoodsPriceList sort(GoodsPriceList goodsPriceList,Order order) | 根据Order为升序还是降序，将参数中的goodsList按价格排序 |
| Public void addShoppingSite (ShoppingSite site) | 对当前用户，增加一个参与比价的比价网站 |
| Public void deleteShoppingSite (ShoppingSite site) | 对当前用户，删除一个参与比价的比价网站 |

## 类图



## 类描述

### PriceParity接口

|  |  |
| --- | --- |
| 方法名 | Public GoodsList findNamesake(Goods goods) |
| 前置条件 | goods不为null,goods已经存在于数据库中，用户已经登陆 |
| 后置条件 | GoodsPriceList不为null |
| 作用 | 通过goods名字，在当前用户默认的购物网站中找到同名商品列表 |

|  |  |
| --- | --- |
| 方法名 | Public GoodsPriceList sort(GoodsList goodsList, Order order，PriceCalcuStrategy strategy) |
| 前置条件 | goodsList, order, strategy不为null, order的排序依据不为null，用户已经登陆 |
| 后置条件 | 返回的goodsPriceList已经按order的排序依据(asc或desc)排好序 |
| 作用 | 根据order的排序依据对goodsPriceList中的项排序 |

|  |  |
| --- | --- |
| 方法名 | Public void addShoppingSite (ShoppingSite site) |
| 前置条件 | Site不为null，site存在于数据库中，用户已经登陆 |
| 后置条件 | 系统将site添加到用户默认购物网站列表中 |
| 作用 | 给用户比价时增加一个参与比价的购物网站 |

|  |  |
| --- | --- |
| 方法名 | Public void deleteShoppingSite (ShoppingSite site) |
| 前置条件 | Site不为null，site存在于数据库中，用户已经登陆 |
| 后置条件 | 系统将site从用户默认购物网站列表中删除 |
| 作用 | 给用户比价时减少一个参与比价的购物网站 |

### PriceParityImpl类

|  |  |
| --- | --- |
| 方法名 | Public GoodsList findNamesake(Goods goods) |
| 前置条件 | goods不为null,goods已经存在于数据库中，用户已经登陆 |
| 后置条件 | GoodsPriceList不为null |
| 作用 | 通过goods名字，在当前用户默认的购物网站中找到同名商品列表 |

|  |  |
| --- | --- |
| 方法名 | Public GoodsPriceList sort(GoodsList goodsList, Order order，PriceCalcuStrategy strategy) |
| 前置条件 | goodsList, order, strategy不为null, order的排序依据不为null，用户已经登陆 |
| 后置条件 | 返回的goodsPriceList已经按order的排序依据(asc或desc)排好序 |
| 作用 | 根据order的排序依据对goodsPriceList中的项排序 |

|  |  |
| --- | --- |
| 方法名 | Public void addShoppingSite (ShoppingSite site) |
| 前置条件 | Site不为null，site存在于数据库中，用户已经登陆 |
| 后置条件 | 系统将site添加到用户默认购物网站列表中 |
| 作用 | 给用户比价时增加一个参与比价的购物网站 |

|  |  |
| --- | --- |
| 方法名 | Public void deleteShoppingSite (ShoppingSite site) |
| 前置条件 | Site不为null，site存在于数据库中，用户已经登陆 |
| 后置条件 | 系统将site从用户默认购物网站列表中删除 |
| 作用 | 给用户比价时减少一个参与比价的购物网站 |

### NamesakeFinder类

|  |  |
| --- | --- |
| 方法名 | Public GoodsList findNamesake(Goods goods,UserShoppingSiteList siteList) |
| 前置条件 | goods不为null，siteList不为null |
| 后置条件 | goodsPriceList中包含在siteList项对应的购物网站中与goods同名的商品，也包括同义词表中包含的商品 |
| 作用 | 给用户比价时减少一个参与比价的购物网站 |

### PriceSort类

|  |  |
| --- | --- |
| 方法名 | Public GoodsPriceList sort(GoodsPriceList goodsPriceList,Order order) |
| 前置条件 | goodsPriceList,order不为null,order的排序依据不为null |
| 后置条件 | 返回的goodsPriceList已经按order的排序依据排好序 |
| 作用 | 根据order的排序依据和顺序对goodsPriceList中的项排序 |

### PriceCalcuStrategy类

|  |  |
| --- | --- |
| 方法名 | Public GoodsPrice calcuPrice(Goods goods) |
| 前置条件 | goods不为null |
| 后置条件 | GoodsPrice不为null |
| 作用 | 计算商品价格，不同的PriceCalcuStrategy子类实现不同的价格计算策略 |

### UserShoppingSiteController类

|  |  |
| --- | --- |
| 方法名 | Public UserShoppingSiteList getShoppingSiteList(User user) |
| 前置条件 | user不为null |
| 后置条件 | UserShoppingSiteList不为null |
| 作用 | 根据用户返回用户设置的参与比价的购物网站的列表 |

|  |  |
| --- | --- |
| 方法名 | Public void addShoppingSite(ShoppingSite site,User user) |
| 前置条件 | user不为null, site 不为null |
| 后置条件 | 系统将site添加到用户默认购物网站列表中 |
| 作用 | 调用持久化模块，保存用户增加的购物网站到用户设置的参与比价的购物网站列表中 |

|  |  |
| --- | --- |
| 方法名 | Public void deleteShoppingSite(ShoppingSite site,User user) |
| 前置条件 | user不为null, site 不为null |
| 后置条件 | 系统将site从用户默认购物网站列表中删除 |
| 作用 | 调用持久化模块，将用户选择的购物网站从用户设置的参与比价的购物网站列表中删除 |

## 重要协作

### 寻找同名商品



### 计算价格



### 排序



### 增删购物网站

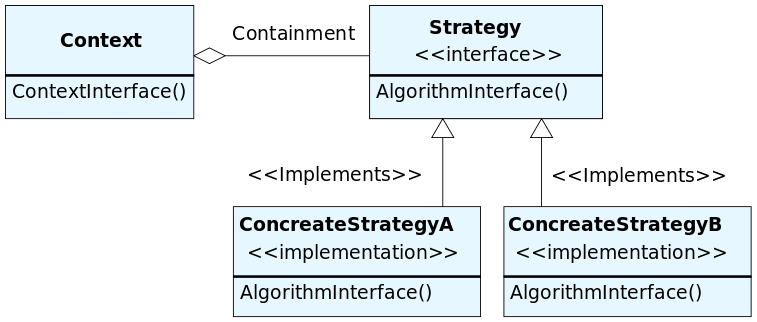


## 设计模式

### 策略模式

策略模式定义了一系列的算法，并将每一个算法封装起来，而且使它们之间可以相互替换。策略模式让算法独立于使用它的客户而独立变化。它提供了一种替代继承的方法，而且既保持了继承的优点(代码重用)，还比继承更灵活(算法独立，可以任意扩展)。同时避免程序中使用多重条件转移语句，使系统更灵活，并易于扩展。策略模式还遵守大部分GRASP 原则和常用设计原则，高内聚、低偶合。

如下图所示：



在价格计算模块中，因为要求考虑到不同计算模式下总价的差异，因此必须将各个价格计算算法封装起来，实现calcuPrice()接口后，就可以方便的修改，如果要新增算法，增加一个算法类，实现上述接口，并在声明的时候将引用指向新的算法类即可；如果要移除算法，移除对应的算法类并更新声明的引用即可；如果要修改算法，可在不修改接口的前提下修改算法类的内部即可，保证了高内聚和低耦合。

类图体现：



# SafeGuard模块详细设计

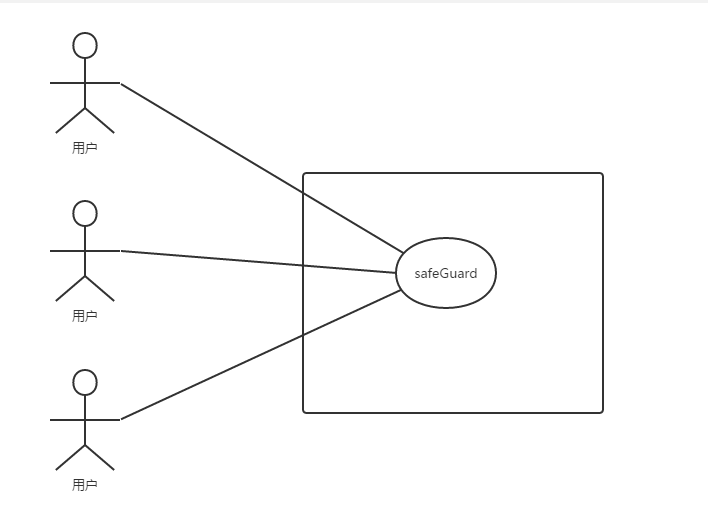
## 概述

模块需求：该模块负责保证服务器安全。用户访问服务器时需要先通过服务器检测模块，屏蔽特定IP以及访问过于频繁的IP（插件扫描攻击）。并且该模块支持可能的入侵检测规则扩展。

涉及需求：

R3 系统安全性 屏蔽99%以上的插件扫描攻击和特定的IP 促进C4

R4 入侵检测可拓展性 增加新的检测规则所需时间低于0.25人月



说明：用户的请求到达SearchProcessor模块前需要先通过SafeGuard模块进行处理

**可能会发生的修改实现**

所屏蔽的特定IP的列表

系统安全检测方法

**分配职责**

1. 该模块将用户访问中可能出现的不安全行为屏蔽，确保服务器的安全和效率

**对外接口：**

接口：Int doSafeCuard(HttpRequest request） ()

功能：当用户的请求到达系统时，首先调用该接口，该接口将返回对该IP访问的检查结果，如果检查通过，则调用模块继续向下调用，否则屏蔽该次访问

**参数说明：**

HttpRequest request：用户请求信息，含有用户的IP、访问时间、mac地址、访问目标等

**返回值说明：**从0至5，分别表示危险评级，屏蔽3级及以上危险评级的访问

**使用方法：**

每当一个用户向服务器请求数据的时候，调用该模块检查用户是否符合访问安全要求，

**前置条件：**

传入的request中包含用户IP、mac地址、访问时间、访问目标

**后置条件**

1、返回对该次访问的安全评级

2、对储存的数据进行修改，更改某些IP地址的访问记录以及安全评价

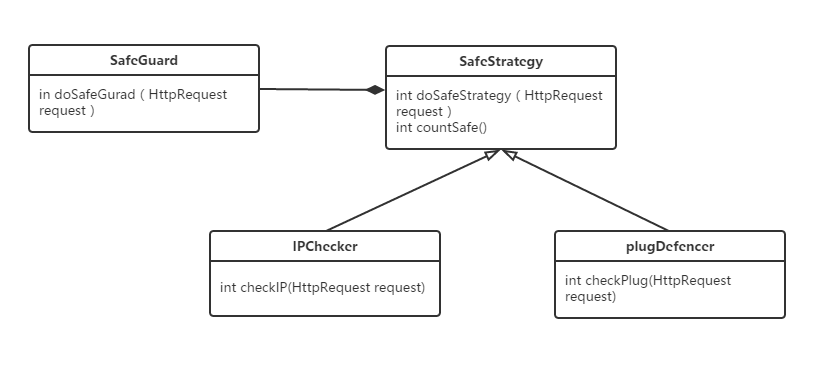
**设计方案分析**

用户的请求到达的时候，SafeGurad模块调用SafeStrategy模块进行访问分析，SafeStrategy模块则依此调用继承自SafeStrategy模块的安全策略模块，目前拥有的是IPCheck模块和PlugDefence模块

## 2类图

**静态结构**

**详细设计类图和类图说明：**



设计说明：该模块主要由两个部分组成：SafeGuard 和SafeStrategy

HttpRequest通过SafeGuard类传入SafeStrategy类，再通过SafeStrategy类调用其子类，最后通过子类返回值计算出该次request的安全评级并返回

## 3类描述

**类方法（重要方法要给出前置与后置条件）与重要数据结构描述**

|  |  |
| --- | --- |
| 类 | 说明 |
| SafeGuard | 传入HttpRequest对象，向下传递并获取返回结果 |
| SafeStrategy | 安全策略类，为所有策略的父类，调用所有策略，计算结果并返回 |
| IPChecker | 策略类的子类，检查IP是否在屏蔽列表内 |
| PlugDefencer | 策略类的子类，检查该次访问是否是插件扫描攻击 |

|  |  |
| --- | --- |
| SafeGuard | |
| DoSafeGuard | 获取HttpRequest向下传递并获取返回值 |

|  |  |
| --- | --- |
| SafeStrategy | |
| doSafeStrategy | 调用所有的检查策略，并返回该次访问的危险值 |
| CountSafe | 计算该访问的危险并返回 |

|  |  |
| --- | --- |
| IPChecker | |
| checkIP | 检查IP是否合法 |

|  |  |
| --- | --- |
| PlugDefencer | |
| checkPlug | 检查该次访问是否是插件扫描攻击 |

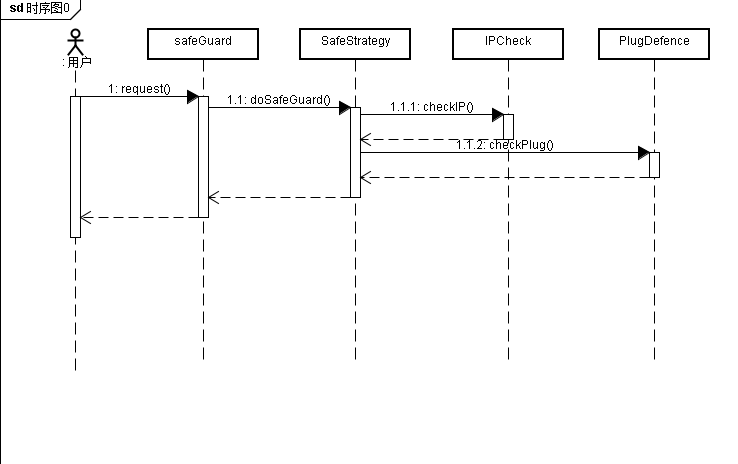
**重要的数据结构描述**

用户请求

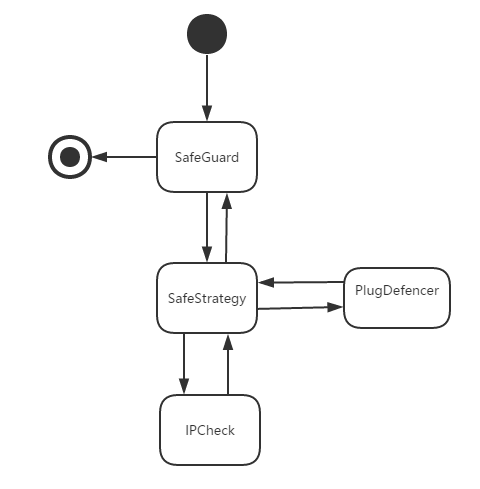
HttpRequest：其中有用户IP、mac地址、访问时间、访问目标

## 4重要协作

**顺序图**



**状态图**



## 5使用的设计模式

**策略模式：**模块含有一个策略的调用者SafeStrategy和若干个继承自SafeStrategy的安全策略类，SafeGuard调用SafeStrategy类，由safeStrategy类调用其记录的所有策略类。当安全检查策略发生修改或删除时，只需修改对应的策略类，当增加安全策略时，只需要增加安全策略类，并在SafeStrategy类中增加该类的调用

**场景：**检查用户的访问安全

**目的：**屏蔽不安全的请求

# Partake模块详细设计

## 概述

模块需求：该模块负责保证服务器扩展、服务器正常运行和服务器负载分担。用户访问服务器时需要先通过服务器负载均衡模块，负载均衡模块同时负责监测各个服务器正常运行。

涉及需求：

R1服务器高可靠性 C1 服务器故障应该在10min内侦测出来

C2 服务器故障时不能影响正常访问

R6 可维护性 C10 允许在系统运行状态下更新系统组件,系统信息

R7 系统运行容故障性 C11 系统存储的数据要每3小时保存一次

R23容故障性 8 C29 对系统存储的数据进行备份

C30 每隔30秒定时保存

说明：用户访问服务器时需要先通过服务器负载均衡模块

**可能会发生的修改实现**

增加服务器

减少服务器

**分配职责**

1. 该模块负责服务器的扩展、负载均衡和保证服务器故障时正常运行

**对外接口：**

接口：void doVisit(HttpRequest request);

功能：当用户的请求到达系统并且通过安全检查后，调用该接口，该接口将选择工作正常并且负载较低的服务器响应请求

**参数说明：**

HttpRequest request：用户请求信息，含有用户的IP、访问时间、mac地址、访问目标等

**返回值说明：**

无

**使用方法：**

每当一个用户向服务器请求数据的时候，调用该模块以选取服务器响应用户请求

**前置条件：**

负载均衡服务器工作正常

**后置条件**

无

**设计方案分析**

负载均衡模块使用单例模式以保证只有一个实例

负载均衡模块定时对所有服务器进行心跳测试以确保所有服务器正确运行

## 2类图

**静态结构**

**详细设计类图和类图说明：**

设计说明：该模块主要由两个部分组成：SafeGuard 和SafeStrategy

HttpRequest通过SafeGuard类传入SafeStrategy类，再通过SafeStrategy类调用其子类，最后通过子类返回值计算出该次request的安全评级并返回

## 3类描述

**类方法（重要方法要给出前置与后置条件）与重要数据结构描述**

|  |  |
| --- | --- |
| 类 | 说明 |
| Partake | 传入HttpRequest对象，调用负载最低的服务器响应请求 |
| CilentChecker | 检查各个服务器是否正常，并返回该服务器的负载 |
| CilentFeedback | 服务器用于反馈自己的负载 |
| CilentBackup | 策略类的子类，检查IP是否在屏蔽列表内 |

|  |  |
| --- | --- |
| Partake | |
| doVisit | 获取HttpRequest，调用负载最低的服务器响应请求 |

|  |  |
| --- | --- |
| CilentChecker | |
| CheckStart | 开始执行服务器监测 |
| GetFreeClient | 获取最空闲的服务器 |

|  |  |
| --- | --- |
| CilentFeedback | |
| Feedback | 返回自身的负载 |

|  |  |
| --- | --- |
| CilentBackup | |
| Backup | 保存服务器以及数据库数据到备份服务器 |
| Restore | 从备份服务器服务器以及数据库数据 |

**重要的数据结构描述**

用户请求

HttpRequest：其中有用户IP、mac地址、访问时间、访问目标

## 4重要协作

**顺序图**

**状态图**

## 5使用的设计模式

**单例模式：**partake类采用单例模式，将构造函数隐藏并提供公共的getInstance方法获取该类的对象，以保证partake类只有一个实例

**场景：**调用partake模块相关方法

**目的：**保证partake类不会拥有多个实例

# Crawler模块

1.1 概述

**承担的需求（主要秘密）：**

该模块主要承担的任务是从其他网站上抓（爬）取商品价格信息，抓（爬）取策略的选择和配置以及周期性抓取新数据以保证数据有效性。

模块需要实现自动抓（爬）取指定网页的功能，初步过滤掉无效信息，然后将商品相关的信息交给数据处理模块处理。对一个网页进行数据抓（爬）取时，将页面中的有效链接放入到一个URL抓（爬）取队列中，在当前网页抓（爬）取完毕后，从URL抓（爬）取队列中选取下一个URL作为抓（爬）取对象。以此操作，直到URL抓（爬）取队列为空。

模块需要实现抓（爬）取策略可变更的功能，能够根据爬虫管理员的选择来采取相应的网页抓（爬）取策略。抓（爬）取策略存储和配置存储在配置文件中。当系统初始化或者抓（爬）取策略变更时，爬虫类从配置文件中读取选定的抓（爬）取策略，然后开始网站抓（爬）取工作。

模块需要实现一个可配置的定时器来完成周期性抓（爬）取的功能。定时器的各项配置保存在配置文件中。爬虫管理员需要提前设定好定时器的周期。在开启爬虫程序后，一旦定时器达到指定周期，就会触发爬虫去抓（爬）取网站信息。在爬虫程序运行过程中，爬虫管理员可以随时改变定时器的周期，然后定时器的配置文件会做出相应的更改，然后在适当的时间（即爬虫程序空闲时）重启爬虫程序来使配置生效。

**可能会修改的实现（次要秘密）：**

增加新的网站抓（爬）取源时，需要增加新的网页内容抓（爬）取方式。

增加新的网站抓（爬）取策略时，需要增加新的抓（爬）取策略实现。

**角色：**

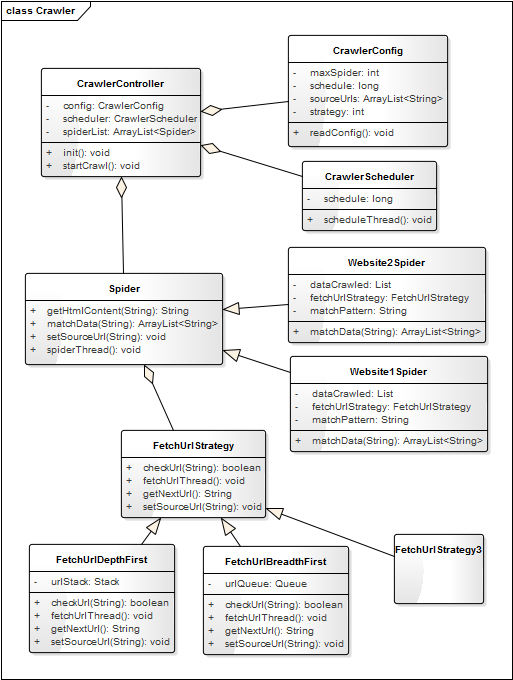
在整个系统中扮演商品数据信息获取的角色，获取的数据交给商品兼容处理模块。

**对外接口：**

1）void analyseData();（参数为从网站爬取的原始商品信息）

提供给商品数据兼容处理模块，目的是让其能够获取从网站上爬取的原始商品信息。

1.2 类图



* 1. 类描述

**CrawlerController类：**爬虫模块的集中控制类，包括爬虫配置文件读取、爬虫定时器的启动、初始化爬虫和开始爬虫程序。类中维护了一个爬虫列表。

**Init():void----**爬虫程序初始化方法，通过CrawlerConfig读取配置文件来获取一些运行时的基本参数。

**startCrawl():void----**爬虫程序的启动方法，里面根据config里的基本参数来新建爬虫对象，开启爬虫线程。

**CrawlerConfig类：**爬虫模块配置类，负责读取配置文件中的爬虫配置，从而获得爬虫程序的启动参数。具体参数包括爬取网站源、爬取周期、爬取策略、最大爬虫数量等。

**readConfig():void----**读取配置文件，获得基本的爬虫参数。

**CrawlerScheduler类：**爬虫模块的定时器类，负责周期性启动爬虫程序。

**scheduleThread():void----**开启定时器的计时线程，当一个周期到时，能够重新开启爬虫程序来爬取数据。

**Spider类：**爬虫模块的抓取数据类，负责根据给定的URL抓取网页的信息，然后从中提取需要的商品信息。它是具体网站信息抓取类Website1Spider、Website2Spider类的父类，包含了爬虫类的统一属性和方法。

**setSourceUrl(String):void----**设置待抓取网站的源URL，给URL爬取类使用。

**spiderThread():void----**启动爬虫页面爬取类的线程。

**getHtmlContent(String):String----**根据URL获取网页的html内容，作为返回值返回，供后面提取商品信息用。

**matchData(String):ArrayList<String>----**根据匹配规则筛选出抓取页面中的商品信息。

**WebSite1Spider类：**是适用于具体网站Website1的爬虫类。

**matchData(String):ArrayList<String>----**根据适用于Website1的信息匹配规则，提取出页面中的商品信息。

其他方法与父类中相同。

**Website2Spider类：**是适用于具体网站Website2的爬虫类。

**matchData(String):ArrayList<String>----**据适用于Website2的信息匹配规则，提取出页面中的商品信息。

其他方法与父类中相同。

**FetchUrlStrategy类：**爬虫模块的URL爬取类，负责爬取给定网站所包含的所有URL，并且对URL进行去重。爬取到的URL提供给爬虫类进行网页数据的抓取。

其中的方法为抽象方法，供具体策略类实现。

**FetchUrlDepthFirst类：**URL爬取类的具体实现，运用了深度优先遍历的URL爬取策略。

**setSourceUrl(String)----**设置待抓取网站的源URL，作为初始URL爬取对象。

**fetchUrlThread():void----**开始URL爬取线程，爬取的URL放入URL栈中。

**checkUrl(String):Boolean----**检查爬取的URL是否重复，不重复则加入URL栈中。

**getNextUrl():String----**爬虫类从URL栈中获取下一条可抓取信息的URL。

**FetchUrlBreadthFirst类：**URL爬取类的具体实现，运用了广度优先遍历的URL爬取策略。

**setSourceUrl(String)----** 设置待抓取网站的源URL，作为初始URL爬取对象。

**fetchUrlThread():void----**开始URL爬取线程，爬取的URL放入URL队列中。

**checkUrl(String):Boolean----**检查爬取的URL是否重复，不重复则加入URL队列中。

**getNextUrl():String----**爬虫类从URL队列中获取下一条可抓取信息的URL。

**FetchUrlStrategy3类：**URL爬取类的具体实现，运用了其他的URL爬取策略（Strategy3）。

* 1. 重要协作

**顺序图**

**协作描述**

* 1. 使用的设计模式

**使用场景**

**所用设计模式**

**要达到的效果**

# CrawlerDataUpdate模块

2.1 概述

**承担的需求（主要秘密）：**

该模块主要承担的任务是将规范化后的商品数据增量更新到数据库中。由于商品信息中不同的数据改变频率不同（比如商品描述变化少，商品价格、数量变化频繁），该模块会对不同的部分采取不同的更新策略，以此来提高数据更新的效率。

**可能会修改的实现（次要秘密）：**

可能会新增不同的数据更新方式和策略。

可能会修改数据库表结构的设计。

**角色：**

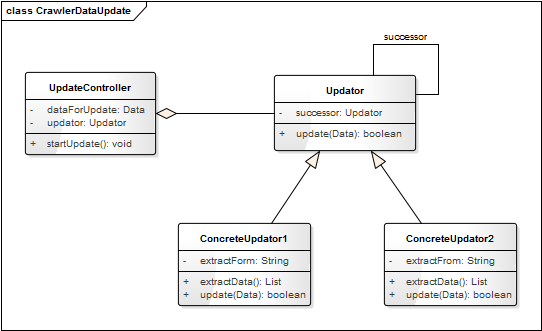
在爬虫部分中扮演数据更新的角色，负责将商品兼容模块处理后的数据更新进数据库。

**对外接口：**

1）void giveFormattedData();（参数为分析处理后的商品数据）

提供给商品数据兼容处理模块，目的是获取处理后的商品数据，从而更新到数据库中。

2.2 类图



* 1. 类描述

**UpdateController类：**爬虫数据更新模块的集中控制类，其中包括分析处理过后的规范化数据和一个更新器。

**startUpdate():void----**开始数据的更新。

**Updator类：**爬虫数据更新模块的更新器，负责对数据进行更新。里面维护了一个Updator对象，如果还有指向的更新器，则继续向下传递数据进行更新。

**update(Data):boolean**----数据更新方法。

**ConcreteUpdator1类：**更新器类的具体实现，从接收到的数据中提取自己负责更新的部分，然后按照给定策略进行更新。

**extractData():List**----按照extractForm里的规则，从接收到的数据中提取自身类负责更新的部分。

**update(Data):boolean----**重写Updator类里的方法，按照给定的策略进行更新。

**ConcreteUpdator2类：**同ConcreteUpdator1类，只是其负责更新的数据部分不同。即其extractForm里提取数据的规则不同，update(Data):boolean里的数据更新策略有不一样的实现。

* 1. 重要协作

**顺序图**

**协作描述**

* 1. 使用的设计模式

**使用场景**

**所用设计模式**

**要达到的效果**

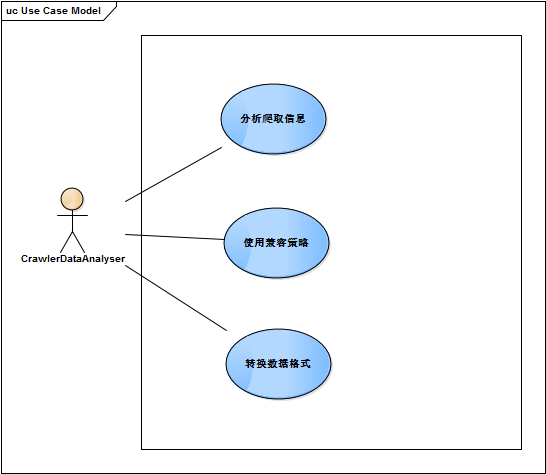
# CrawlerDataAnalyser模块

## 概述

**承担的需求（主要秘密）**

本模块是由Crawler模块的analyseData()方法调用，目的是为了对crawler爬虫模块爬取到的信息进行分析处理，使用CompatibilityHandler模块提供的兼容策略，并为调用CrawlerDataUpdater模块存储爬取信息做准备。

用例图如下：



用例图解释：

1. 分析爬取信息：由于爬取信息来源的网站不同，爬取到的信息内容也会因网站的不同而产生差异，因此CrawlerDataAnalyser模块需要对不同信息的来源进行不同的分析，将其转换为可供兼容性操作的数据模型
2. 使用兼容策略：通过调用CompatibilityHandler模块提供的接口对不同网站来源的同一商品信息进行兼容
3. 转换数据格式：将经过兼容处理的同一商品信息转换为存储在数据库中的数据模型，完成后调用CrawlerDataUpdater模块进行存储

**可能会修改的实现（次要秘密）**

1. 可能对于新增的爬取信息来源网站需要增加新的分析程序
2. 可能需要修改数据库中的数据对象模型

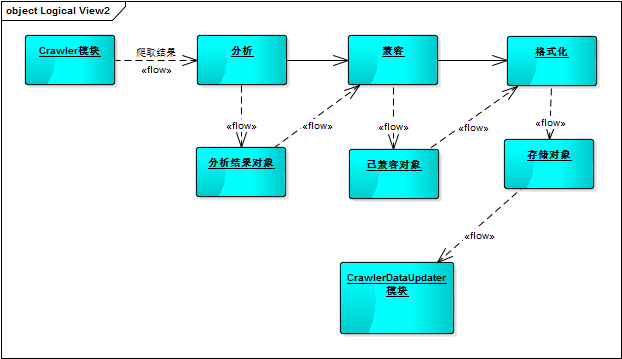
**角色**

1) 分析功能：接收Crawler模块爬取的信息，并根据来源网站的不同进行分析，建立新的数据对象类型，保存爬取信息

2) 兼容功能：调用CompatibleHandler模块提供的接口，对于同一商品的信息进行兼容处理

3) 格式化功能：将经过兼容处理的内容格式化，并存放到新的数据模型中，准备进行存储

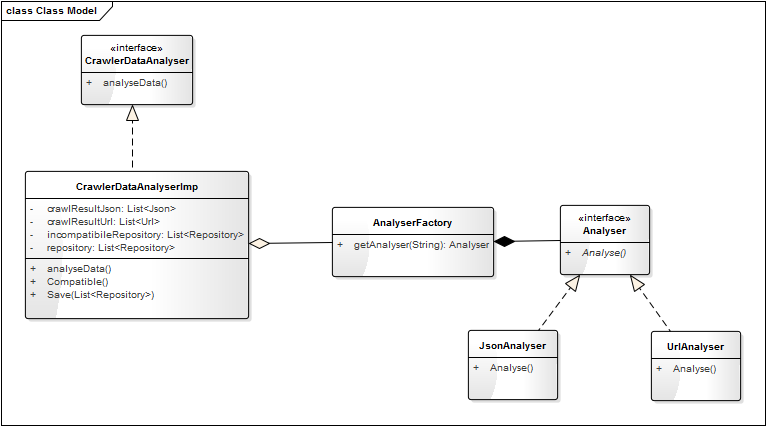
4）角色流程图：



**对外提供接口**

向Crawler提供的接口：void analyseData();

## 1.2 类图



## 1.3 类描述

### 1.3.1 CrawlerDataAnalyser类

**类方法：**

Void analyseData()：

此抽象方法提供调用CrawlerDataAnalyser的主逻辑程序的接口，开启分析过程，由继承CrawlerDataAnalyser的子类实现该方法

### 1.3.2 CrawlerDataAnalyserImp类

该类保存了各阶段程序执行后的产物，方便各阶段的测试与修改

**类方法：**

Void analyseData()：

此方法实现了父类中的抽象方法，实例化AnalyserFactory对象，获取Analyser分析器，分析处理爬取到的信息，并将结果保存到incompatibleRepository中，并调用Compatibile方法开始兼容处理

前置条件：Crawler模块爬取到了信息调用CrawlerDataAnalyser模块

后置条件：将结果保存到incompatibleRepository中

Void Compatibile()：

调用CompatibilityHandler提供的接口，将不兼容的商品信息处理转换为兼容的商品信息，存放在repository中，并调用save()方法保存数据

前置条件：经过分析处理的信息保存到incompatibleRepository

后置条件：将同一商品的信息保存到一条Repository记录中

Void Save(List<Repository>)：

调用CrawlerDataUpdater模块提供的接口，向数据库中保存信息

List<Json> getResultJson()：

获取CrawlerDataAnalyserImp中保存的爬取到的json数据，进行分析

List<Url> getResultUrl()：

获取CrawlerDataAnalyserImp中保存的爬取到的url数据，进行分析

### 1.3.3 AnalyserFactory类

**类方法：**

Analyser getAnalyser(String type)：

此方法根据参数返回对应类型的Analyser对象，例如A网站爬取的内容是json格式，就是用”json”参数，生成对应分析json数据的Analyser，对数据进行处理

### 1.3.4 Analyser接口

**类方法：**

Void Analyse()：

此方法用于分析处理爬取到的信息，由继承该接口的类实现

### 1.3.5 JsonAnalyser类

**类方法：**

Void Analyse()：

此方法实现Analyser接口定义的方法，专用于分析Json数据，新建数据模型保存json格式中的数据

### 1.3.6 UrlAnalyser类

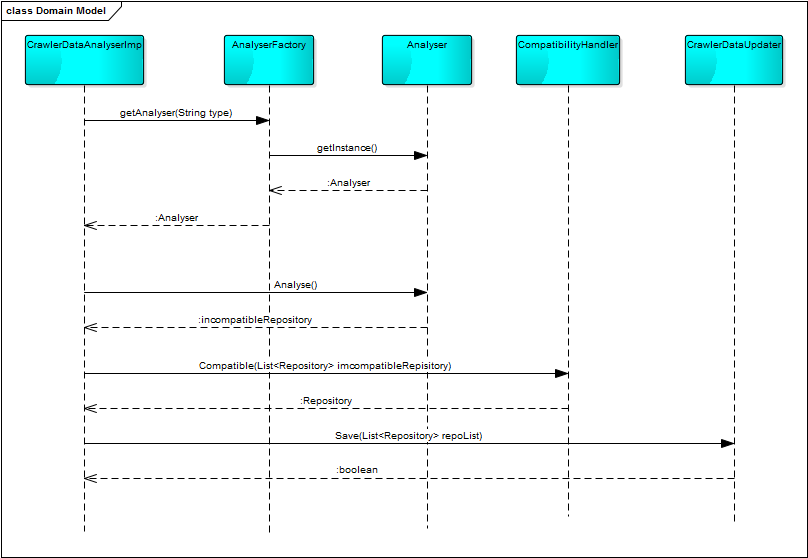
**类方法：**

Void Analyse()：

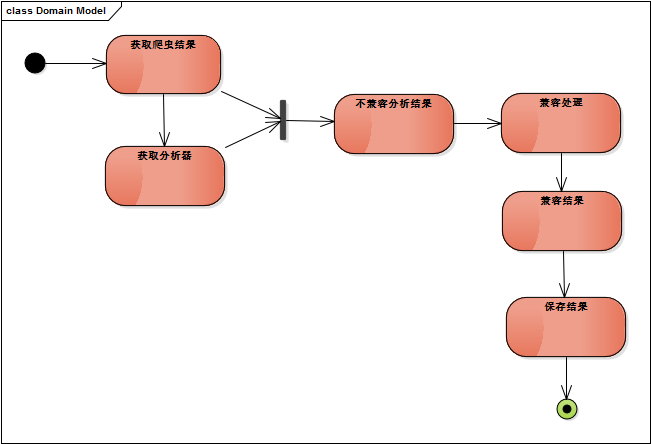
此方法实现Analyser接口定义的方法，专用于分析Url数据，新建数据模型保存url格式中的数据

## 1.4 重要协作

顺序图：



状态图：



## 1.5 设计模式

# CompatibilityHandler模块

# PriceCompatibility模块

# LanguageCompatibility模块

# ScaleCompatibility模块

# BrowerAdapt模块

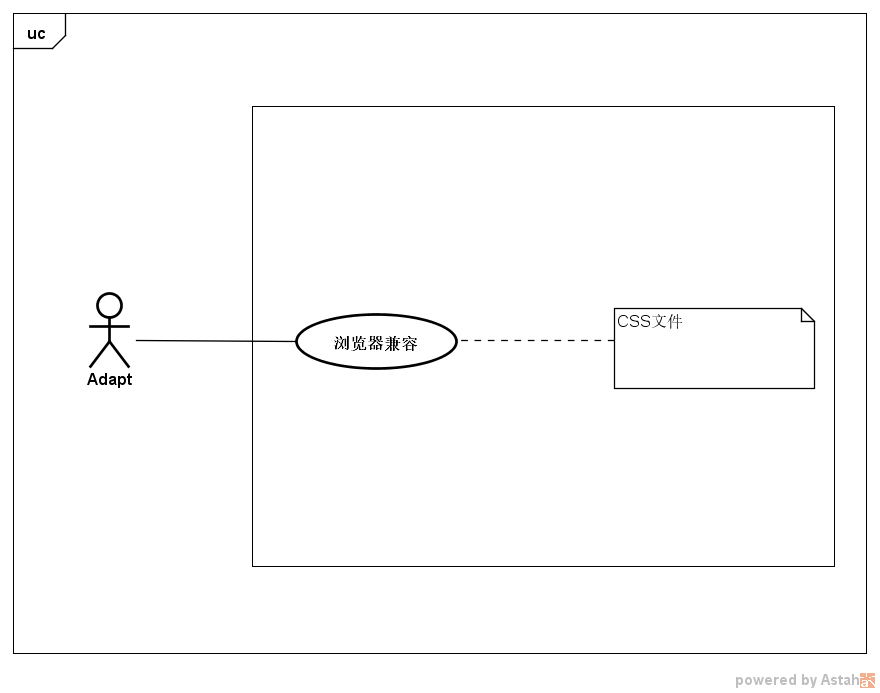
## 概述

### 承担的需求（主要秘密）

浏览器兼容模块的职责是根据用户浏览器类型及版本的不同，自动采取兼容措施，保证网页在不同浏览器上仍保持一致的显示效果。

**质量属性要求**：网页显示要求具有较高的兼容性，而变更的部分主要是用户浏览器类型及版本的变化。因此该模块的设计就要着重考虑网页CSS对不同浏览器的兼容性，可以灵活变更又不影响其他模块的调用，同时，为避免CSS进行兼容变更之后文件过大影响网站加载速度，采取按兼容版本不同编写相应的同名CSS文件，并由浏览器兼容模块统一调度。

**浏览器兼容用例图**



**用例描述**

1. 浏览器兼容

当用户访问网站时，系统自动获取用户的浏览器版本，并返回相应的CSS文件，

以达到兼容浏览器的目的。

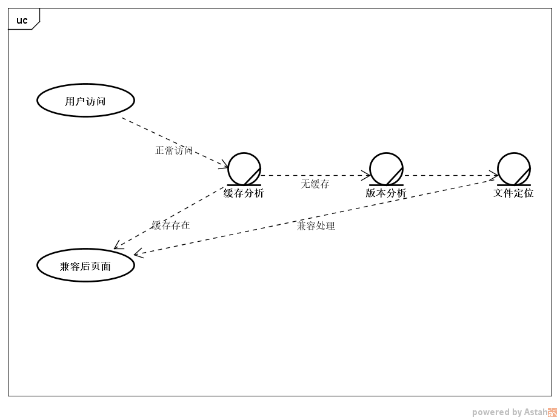
**可能会修改的实现（次要秘密）**

扩展兼容版本

**角色**

1. CSS缓存文件分析
2. 浏览器版本分析
3. CSS文件重定位

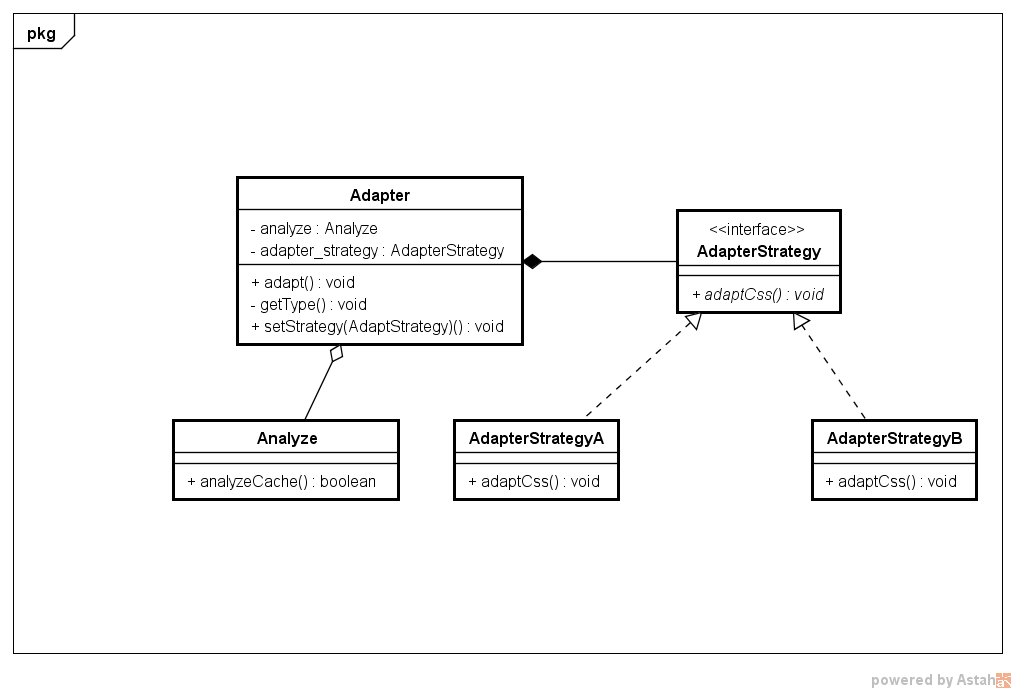
**角色流程图**

****

**向外提供接口**

|  |  |
| --- | --- |
| **接口名** | **职责** |
| Public void adapt() | 根据浏览器类型，选择合适的CSS文件 |

## 类图



## 类描述

### Analyze类

该类的职责是分析用户浏览器是否缓存了相应的CSS，以便

Adapter类判断是否需要进行兼容操作

**类方法：**

Boolean analyzeCache();

检查是否有网站缓存，并返回结果

前置条件：用户正常访问

后置条件：无

### Adapter类

该类的职责是根据Analyze类的结果判断是否需要进行兼容处理，如果需要，则根据兼容策略进行兼容处理

**类方法：**

Void setStrategy(AdaptStrategy)

组合模型表明在Adapter类的构造函数中其实就已经指定了兼容算法(AdaptStrategy),但为了更好地动态指定不同的兼容算法，我们可以使用setStrategy方法来重设Adapter对象的adapt\_strategy成员变量。

前置条件:Adapter对象已经被初始化；

后置条件: Adapter对象的adapt\_strategy成员变量被重新设置为新的算法对象

Void adapt()

Adapter类的主要方法，负责根据缓存分析结果和兼容策略，对用户浏览器进行兼容

BrowerType getType()

该方法获取用户浏览器类型版本等信息并返回

### AdapterStrategy接口

为了使得不同的兼容算法能够更好地复用，以及避免将来算法的变更对现有行为类

造成影响，选择策略模式将搜索算法抽象成接口，在接口中申明adaptCss方法，并由

具体的AdapterStrategy类实现

### AdapterStrategyA类、AdapterStrategyB类

这些类，是AdapterStrategy接口的具体实现类，实现了接口中申明的方法，并且数量在未来可能增加

**类方法：**

Void adaptCss(BrowerType type)

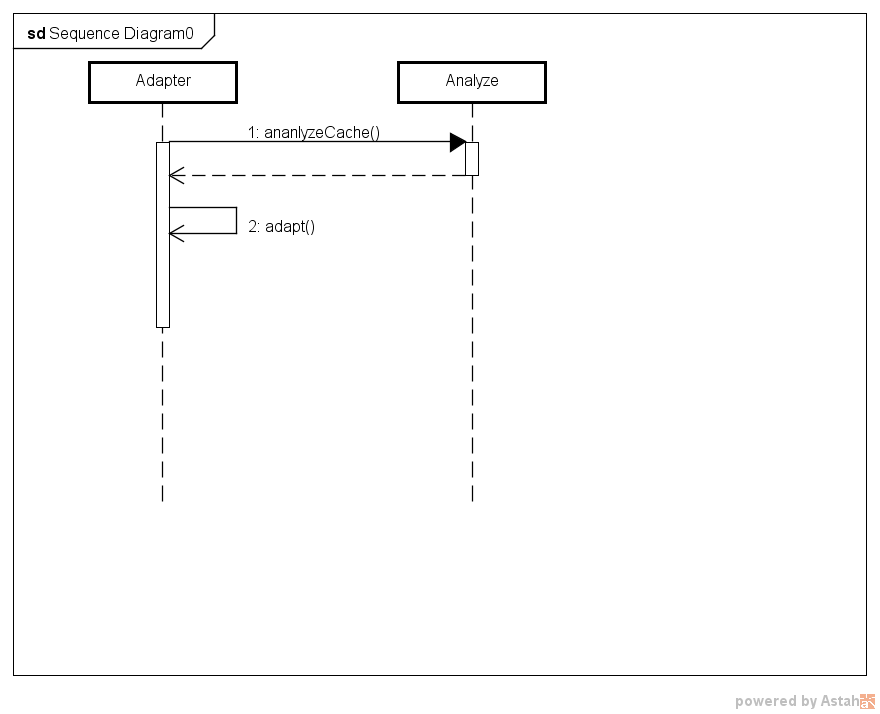
该方法是实现了接口中定义的adaptCss方法，参数type为用户浏览器类型信息

前置条件：getType方法以被执行

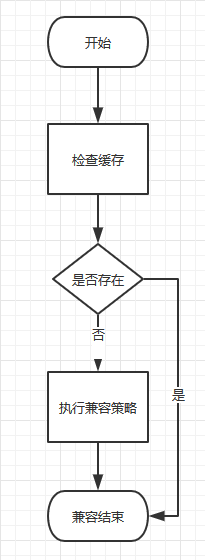
后置条件：无

## 重要协作

### 顺序图



### 流程图



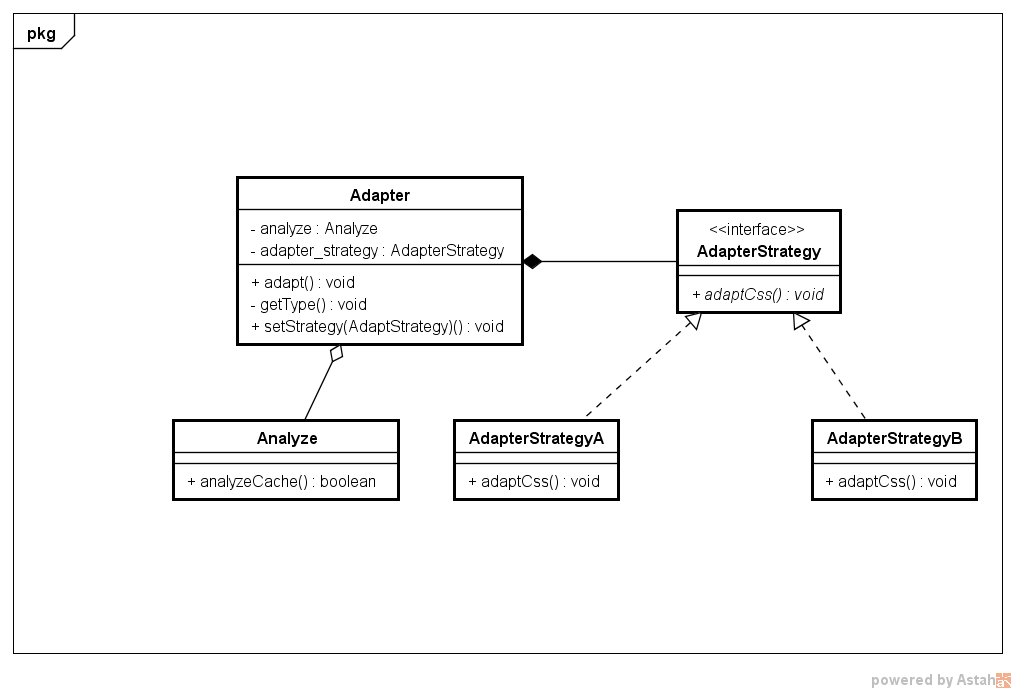
## 设计模式

### 策略模式

策略模式定义了一系列的算法，并将每一个算法封装起来，而且使它们之间可以相互替换。策略模式让算法独立于使用它的客户而独立变化。它提供了一种替代继承的方法，而且既保持了继承的优点(代码重用)，还比继承更灵活(算法独立，可以任意扩展)。同时避免程序中使用多重条件转移语句，使系统更灵活，并易于扩展。策略模式还遵守大部分GRASP原则和常用设计原则，高内聚、低偶合。

在浏览器兼容模块中，由于浏览器兼容算法以后可能变更，因此把该算法封装起来，实现接口adaptCss（），以便将来可以方便地修改。以便以后可以方便地修改。如果要新增算法，增加一个算法类，实现上述接口，并在声明的时候将引用指向新的算法类即可；如果要移除算法，移除对应的算法类并更新声明的引用即可；如果要修改算法，可在不修改接口的前提下修改算法类的内部即可，保证了高内聚和低耦合。

**类图体现：**

****

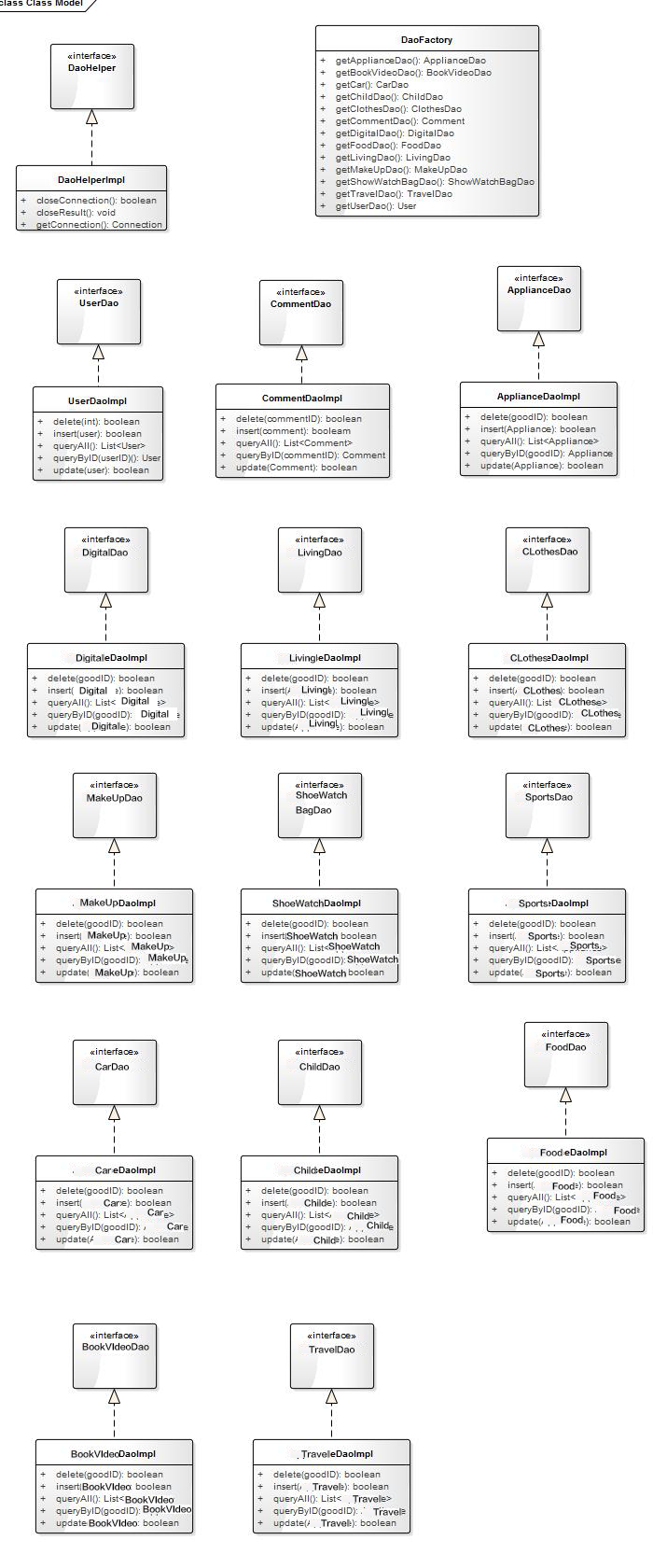
# 数据库模块

1.1概述

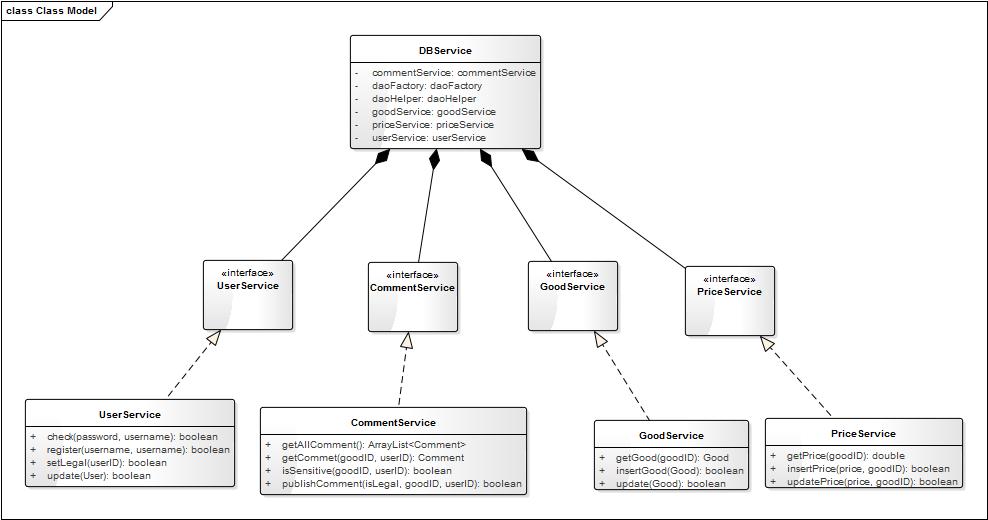
此模块旨在向爬虫模块和用户模块提供必要的数据库接口，是的服务器可以从分布式数据库服务器中取得商品、用户、评论等信息

1.2类图

DAO类图



DBService类图



1.3类描述

DAO类描述

**DaoFactory类：**Dao工厂类，实例化各Dao对象

getUserDao():实例化并返回UserDao对象

getCommentDao():实例化并返回CommentDao对象

getApplianceDao():实例化并返回ApplianceDao对象

getDigitalDao():实例化并返回DigitalDao对象

getLivingDao():实例化并返回LivingDao对象

getClothesDao():实例化并返回ClothesDao对象

getMakeupDao():实例化并返回MakeupDao对象

getShoeWatchBagDao():实例化并返回MakeupDao对象

getSportsDao():实例化并返回MakeupDao对象

getCarDao():实例化并返回MakeupDao对象

getChildDao():实例化并返回ChildDao对象

getFoodDao():实例化并返回FoodDao对象

getBookVideoDao():实例化并返回MakeupDao对象

getTravelDao():实例化并返回TravelDao对象

**DaoHelper类：**数据库连接类，连接数据库获取连接对象、关闭数据库

closeConnection(con):关闭数据库连接对象

closePreparedStatement(stmt):关闭预声明对象

closeResult(result):关闭结果集

getConnection():取得数据库连接对象

**UserDaoImpl类：**定义了User对象的增删查操作

delete(userID):根据userID,删除User

insert(user):增加user

queryAll():查询返回User对象列表

queryByID(userID):根据userID查询返回User对象

update(user):更新User对象

**CommentDaoImpl类：**定义了Comment对象的增删查操作

delete(commentID):根据commentID,删除Comment

insert(comment):增加comment

queryAll():查询返回User对象列表

queryByID(commentID):根据commentID查询返回Comment对象

update(comment):更新Comment对象

**ApplianceDaoImpl类：**定义了Appliance对象的增删查操作

delete(goodsID):根据goodsID,删除Appliance

insert(appliance):增加appliance

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(appliance):更新Clothes对象

**DitigalDaoImpl类：**定义了Digital对象的增删查操作

delete(goodsID):根据goodsID,删除Digital

insert(digital):增加digital

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(digital):更新Digital对象

**LivingDaoImpl类：**定义了Living对象的增删查操作

delete(goodsID):根据goodsID,删除Living

insert(living):增加living

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(living):更新Living对象

**ClothesDaoImpl类：**定义了Clothes对象的增删查操作

delete(goodsID):根据goodsID,删除Clothes

insert(clothes):增加clothes

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(clothes):更新Clothes对象

**MakeupDaoImpl类：**定义了Makeup对象的增删查操作

delete(goodsID):根据goodsID,删除Makeup

insert(makeup):增加makeup

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(makeup):更新Makeup对象

**ShoeWatchBagsDaoImpl类：**定义了**ShoeWatchBag**s对象的增删查操作

delete(goodsID):根据goodsID,删除ShoeWatchBags

insert(ShoeWatchBags):增加ShoeWatchBags

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(ShoeWatchBags):更新ShoeWatchBags对象

**SportsDaoImpl类：**定义了sports对象的增删查操作

delete(goodsID):根据goodsID,删除Sports

insert(sports):增加sports

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(sports):更新Sports对象

**CarDaoImpl类：**定义了Car对象的增删查操作

delete(goodsID):根据goodsID,删除Car

insert(Car):增加Car

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(Car):更新Car对象

**ChildDaoImpl类：**定义了Child对象的增删查操作

delete(goodsID):根据goodsID,删除Child

insert(child):增加child

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(child):更新Child对象

**FoodDaoImpl类：**定义了Food对象的增删查操作

delete(goodsID):根据goodsID,删除Food

insert(food):增加food

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(food):更新Food对象

**BookVideoDaoImpl类：**定义了BookVideo对象的增删查操作

delete(goodsID):根据goodsID,删除BookVideo

insert(BookVideo):增加BookVideo

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(BookVideo):更新BookVideo对象

**TravelDaoImpl类：**定义了Travel对象的增删查操作

delete(goodsID):根据goodsID,删除Appliance

insert(travel):增加travel

queryAll():查询返回Goods对象列表

queryByID(goodsID):根据goodsID查询返回Goods对象

update(travel):更新Travel对象

### DBService类图

**DbService类:**数据库服务对象，使得其他模块可以通过此对象操作数据库

属性：

commentService:评论服务对象

daoFactory:dao工厂，实例化并返回dao对象

daoHelper:数据库连接对象

goodsService:商品服务对象

priceService:价格服务对象

userService:用户服务对象

**UserService类:**用户操作接口

isVerified(email,password):判断用户email,password是否正确

register(email,username,password):用户注册

setLegal(userID,legal):设置用户是否为水军

updatePassword(email,username,password):更新密码

verifiedRegister(email,password):判断用户是否已注册

**CommentService类:**评论操作接口

getAllComment():得到所有用户评论

getComment(userID):得到用户评论

isSensitive(userID):判断用户是否发表过敏感言论

publishComment(userID,connent,isLegal):发表评论

**GoodsService类:**商品操作接口

getGoods(goodsID):得到商品

insertGoods(goods):插入商品

isGoodsNew(goods):判断商品是否已存在

updateGoods(goods):更新商品

**PriceService类:**商品价格操作接口

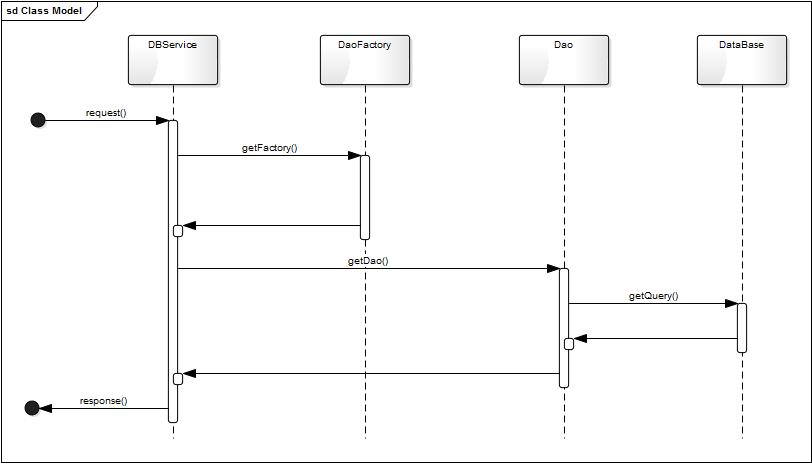
getPrice():取得价格

insertPrice():插入价格

updatePrice():更新价格

1.4重要写作

顺序图



数据库模块接收其他模块的查询请求

DbService取得DaoFactory对象

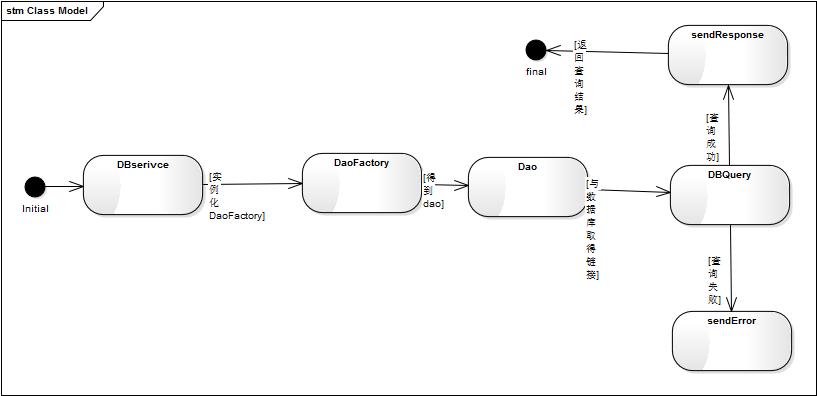
DaoFactory对象实例化一个对应的Dao对象

Dao对象执行查询数据库

Dao对象将结果返回给DbService对象

DbService对象将结果返回给请求模块

状态图



## 1.5设计模式

### Dao模式

通过Dao设计模式包括数据库连接类DaoHelper、Dao接口、Dao实现类、Dao工厂类以及数据库对象模型VO类，增强了程序的可移植性和可维护性。

### Factory模式

通过DbFactory实例化各Dao对象，将创建dao对象与使用dao对象查询数据库分开，有利于后期维护Dao对象

# 1数据库设计概述

## 1.1 数据库部分模块图

## 1.2 数据库大存储量设计说明

1）为了满足系统最大要能够存储1千万个商品的相关信息的要求，故采取分布式数据库进行存储。

2）为了便于进行商品搜索，将不同的商品大类分别存储于不同的子服务器

3）为了能满足大访问量的要求，用户信息和商品信息分开在不同服务器

4）由于要满足系统大存储量并且查询速度的要求，将存储用户信息的子服务器事务隔离机制设置为最低，当进行更新时再更改事务隔离机制

## 1.3 数据库更新设计说明

1）为了满足“数据库实时性”的系统需求，每天对数据库进行更新，同时对数据库采取备份。当数据库更新都是发生在每晚爬虫更新商品信息阶段，故在此阶段使Monitor将当前云次能够数据库切换到备份数据库，然后爬虫对主数据库进行更新，更新过程会有日志存储，更新完毕后，再由Monitor将当前运行数据库切换到主数据库，最后利用日志对备份数据库进行更新。

## 1.4 数据库可靠性设计说明

1）为了满足“满足数据存储出现故障时能够继续工作并快速恢复”的系统需求，数据库采用热备份机制，并且利用Monitor对数据库进行检测，当发现数据库故障时，切换运行数据库。

# 2表结构设计

## 2.1 概述

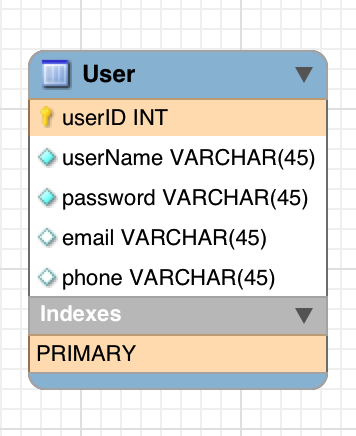
1）由于需要满足大容量存储以及并发性要求，数据库采取分布式存储，其中商品按照大类存储在不同的子服务器上，用户信息则单独存储在数据库的分服务器上，商品的分类如下：家用电器、手机数码、家具家装、服装首饰、个人护理、鞋表箱包、运动户外、汽车用品、母婴玩具、食品生鲜、营养保健、图书音像、机票旅游

2）采用全文索引方式以便加快搜索速度

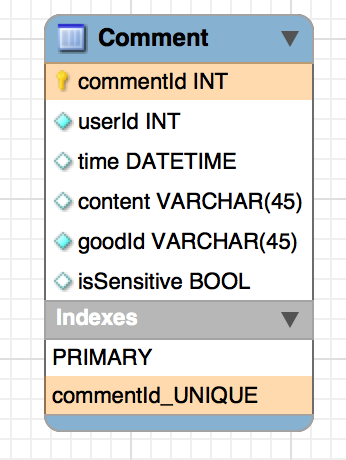
3）采用日志机制便于进行数据库热备份，以及数据库出现故障时能快速恢复

## 2.2 表结构

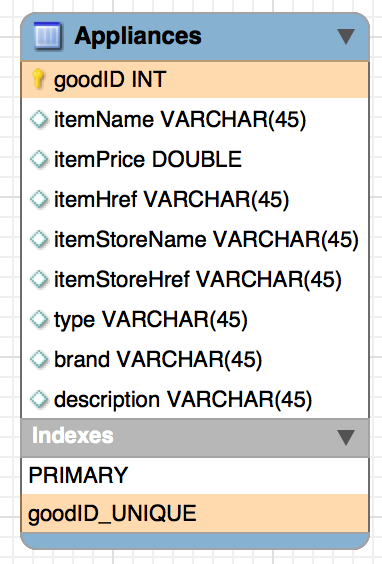
### 2.2.1用户信息表



### 2.2.2评论信息表



### 2.2.3家用电器表



说明：description中形式为：item[value]

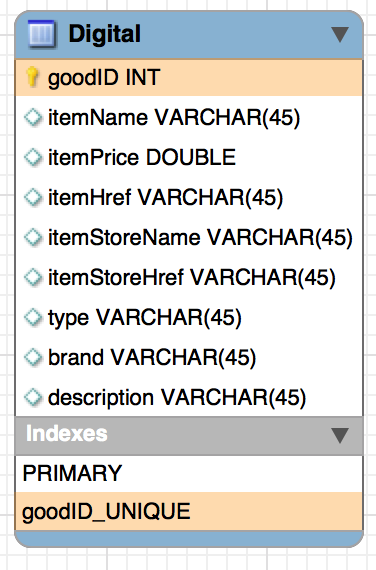
若type为空调，则形式为：商品匹数[]；能效等级[]；空调冷暖方式[]；空调技术[]；适合面积[];其他[]

若type为电视，则形式为：观看距离[]；屏幕尺寸[]；分辨率[]；3D类型[]；面板类型[];背光灯类型[];其他[]

若type为冰箱，则形式为：最大容积[]；是否无霜[]；能效等级[]；家庭组成[]；控温方式[];箱门结构[];其他[]

若type为洗衣机，则形式为：洗涤公斤量[]；家庭组成[]；能效等级[]；使用方式[]；其他[]

### 2.2.4手机数码表



说明：description中形式为：item[value]

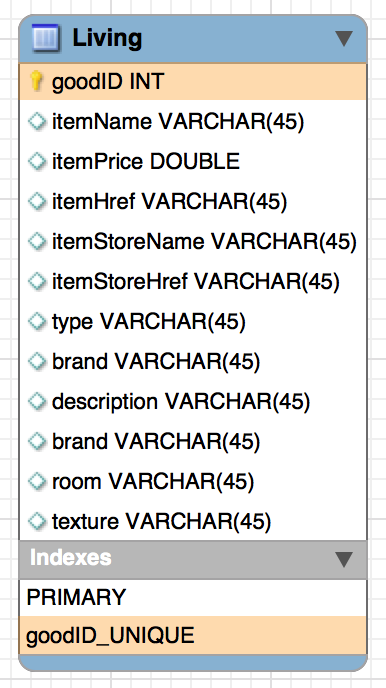
若type为手机，则形式为：网络类型[]；像素[]；核心数[]；分辨率[]；运行内存RAM[];其他[]

若type为数码相机，则形式为：传感器尺寸[]；光学变焦倍数[]；防抖性能[]；是否支持机身防抖[]；屏幕尺寸[];单反画幅[];像素[];其他[]

若type为耳机，则形式为：耳机插头类型[]；耳机输出音源[]；佩戴方式[]；线材[]； 其他[]

若type为电脑，则形式为：显卡类型[]；CPU型号[]；硬盘容量[]；内存容量[]；屏幕比例[]； 其他[]

### 2.2.5家具家装表



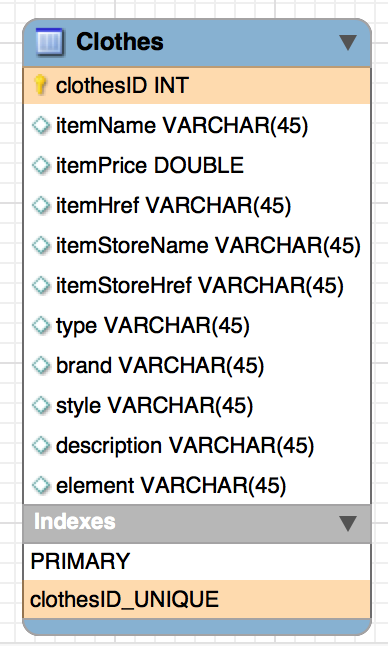
说明：description中形式为：item[value]

若type为餐具，则形式为：盘类型[]；适用人数[]；材质[]；形状[]； 其他[]

若type为床上用品，则形式为：适用床尺寸[]；款式[]；材质[]；面料支数[]； 其他[]

若type为台灯，则形式为：光源类型[]；开关类型 []；电压[]；功率[]；适用空间[]; 其他[]

### 2.2.6服装内衣表



说明：description中形式为：item[value]

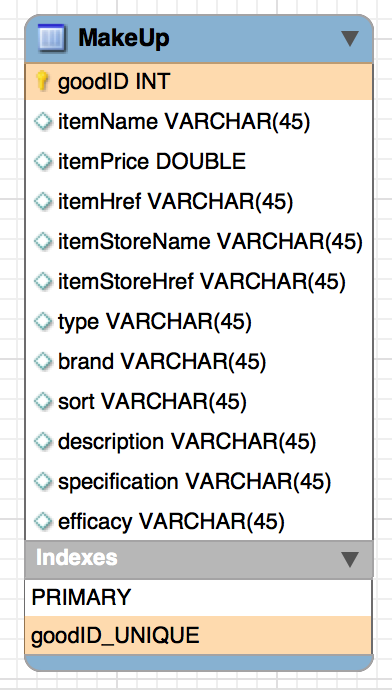
若type为上装，则形式为：衣长[]；尺寸[]；适合年龄[]；风格[]； 其他[]

若type为裤子，则形式为：裤长[]；厚薄[]；适合年龄[]；适用性别[]； 款式[];其他[]

若type为裙子，则形式为：裙长[]；厚薄[]；适合年龄[]；腰型[]；裙型[]; 风格[];其他[]

若type为内衣，则形式为：尺寸[]；厚薄[]；适合年龄[]；适用性别[]；其他[]

### 2.2.7个人护理表

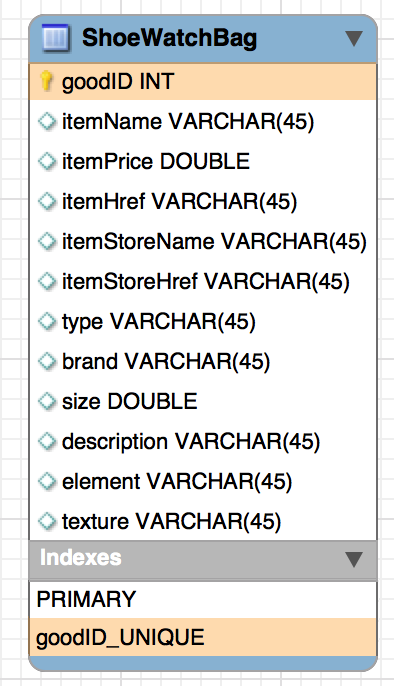


说明：description中形式为：item[value]

若type为护肤，则形式为：产地[]；适合肤质[]；适合年龄[]；化妆品品类[]； 其他[]

若type为美发，则形式为：美发器类型[]；导热体直径[]；功能[]；发型[]； 其他[]

### 2.2.8鞋表箱包表



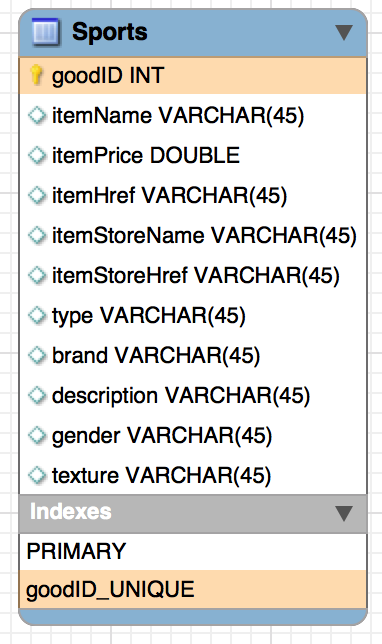
说明：description中形式为：item[value]

若type为女鞋，则形式为：鞋头款式[]；跟高[]；跟形[]；材质 []； 其他[]

若type为表，则形式为：机芯类型[]；表带材质[]；形状[]；防水深度 []； 其他[]

若type为箱包，则形式为：流行款式[]；风格[]；大小[]；材质[]； 其他[]

### 2.2.9运动户外表

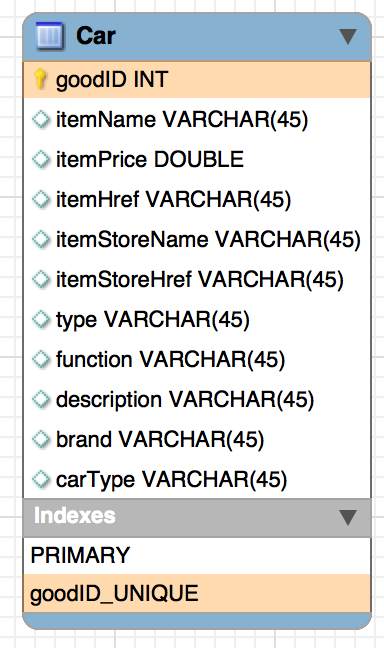


说明：description中形式为：item[value]

若type为跑步鞋，则形式为：适合人群[]；系带[]；功能科技[]；价格区间 []； 适合路面[];尺码[];上市时间[];其他[]

若type为冲锋衣，则形式为：适合性别[]；适用季节[]；户外运动用品[]；户外运动项目 []； 项目[];尺码[];其他[]

2.2.10汽车用品表

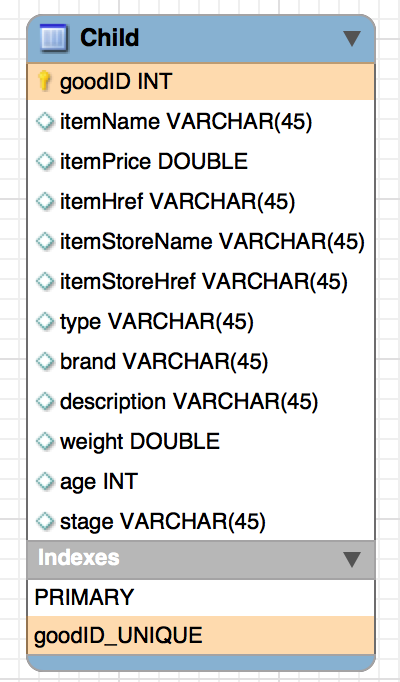


说明：description中形式为：item[value]

若type为导航仪，则形式为：地图类型[]；分辨率[]； 屏幕类型[]；附加功能[]；其他[]

若type为倒车雷达，则形式为：显示器类型[]；探头个数[]； 测试距离[]；附加功能[]；其他[]

### 2.2.11母婴玩具表



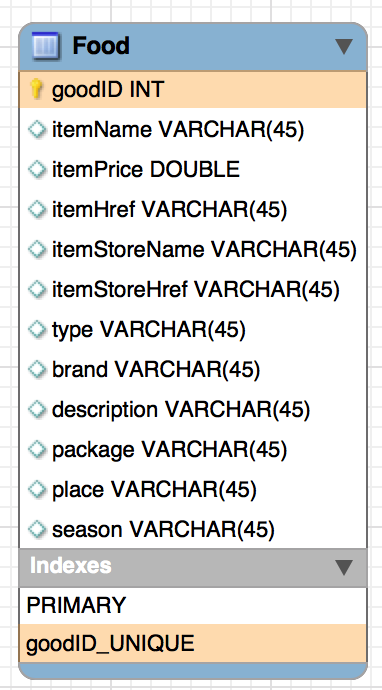
说明：description中形式为：item[value]

若type为奶粉，则形式为：使用阶段[]；产地[]；包装种类[]；奶粉种类[]；其他[]

若type为婴儿床，则形式为：附加功能[]；材质[]；适用年龄[]；尺寸[]；其他[]

若type为孕妇装，则形式为：尺码[]；材质[]；裙长[]；使用季节[]；袖长[];其他[]

### 2.2.12食品生鲜表



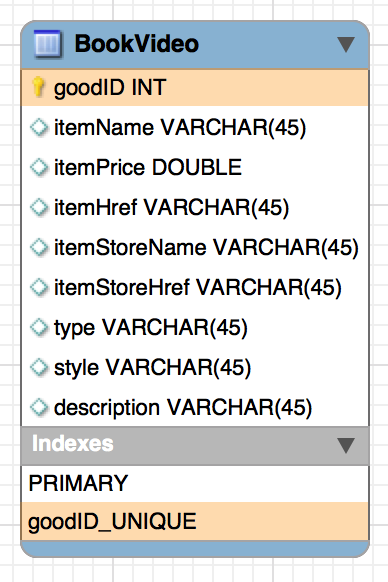
说明：description中形式为：item[value]

若type为零食，则形式为：口味[]；产地[]；包装种类[]；鱼制品种类[]；其他[]

若type为牛奶，则形式为：口味[]；产地[]；包装种类[]；适用对象[]；其他[]

若type为营养品，则形式为：计价单位[]；产地[]；产品剂型[]；适用对象[]；主要原料[];其他[]

### 2.2.13图书音像表



说明：description中形式为：item[value]

若type为图书，则形式为：适合年龄[]；作者地图[]；是否是套装 []；图书种类[]；其他[]

若type为音像，则形式为：适合年龄[]；歌手类型[]；音乐类型 []；种类[]；其他[]

### 2.2.14机票旅游表



# 1.Advertising模块设计

## 1.1概述

### 1.1.1承担的需求（主要秘密）

本模块旨在根据不同合作商户的要求，动态地在网页添加广告

涉及需求：

R3 系统安全性 屏蔽99%以上的插件扫描攻击和特定的IP

R4入侵检测可拓展性 增加新的检测规则所需时间低于0.25人月

R11 浏览器兼容性 兼容Firefox版本号>=3.5;兼容Chrome版本号>=3;兼容IE版本号>=6;兼容Opera版本号>=10.5;兼容Safari版本号>=3.1

R31 植入广告 广告展示的形式多样化

R34 广告可扩展性 广告的位置和数量随时可以发生变化

### 1.1.2可能会修改的实现（次要秘密）

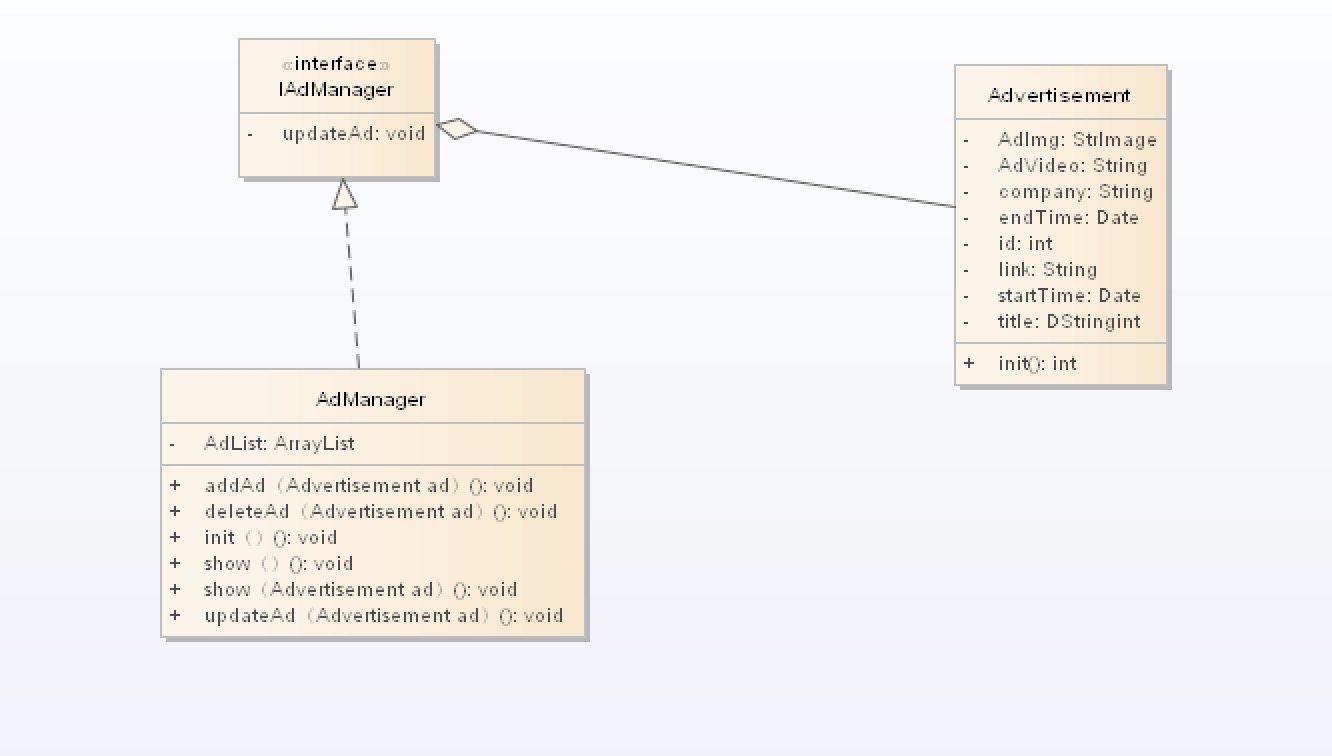
**广告的类型**

### 1.1.3角色

### 1.1.4对外接口

|  |  |
| --- | --- |
| **接口名** | **职责** |
| Public void updateAdvertisementInfo(Advertisement info) | 更新广告内容信息 |
| Public void showAdvertisement (Advertisement info) | 在当前页面显示广告信息 |

## 1.2类图



## 1.3类描述

1.3.1 IAdManager接口

该接口封装了广告管理的方法，根据各商家的出价进行竞价排序，更新网页上的广告信息

1.3.2 AdManager类

该类的职责为对广告的增加，删除，更新，显示进行管理

|  |  |
| --- | --- |
| 方法名 | Public void addAd(Advertisement ad) |
| 前置条件 | ad不为null,ad不存在于数据库中，管理员已经登陆 |
| 后置条件 | 无 |
| 作用 | 在数据库中新增一个广告 |

|  |  |
| --- | --- |
| 方法名 | Public void deleteAd(Advertisement ad) |
| 前置条件 | ad不为null，管理员已经登陆 |
| 后置条件 | 数据库中没有该广告 |
| 作用 | 在数据库中删除一个广告 |

|  |  |
| --- | --- |
| 方法名 | Public void updateAd(Advertisement ad) |
| 前置条件 | ad不为null,数据库中能找到对应id的ad，管理员已经登陆 |
| 后置条件 | 在数据库中更新该广告信息 |
| 作用 | 在数据库中更新一个广告 |

|  |  |
| --- | --- |
| 方法名 | Public void show(Advertisement ad) |
| 前置条件 | ad不为null,ad存在于数据库中，管理员已经登陆 |
| 后置条件 | 无 |
| 作用 | 在页面上显示一个广告 |

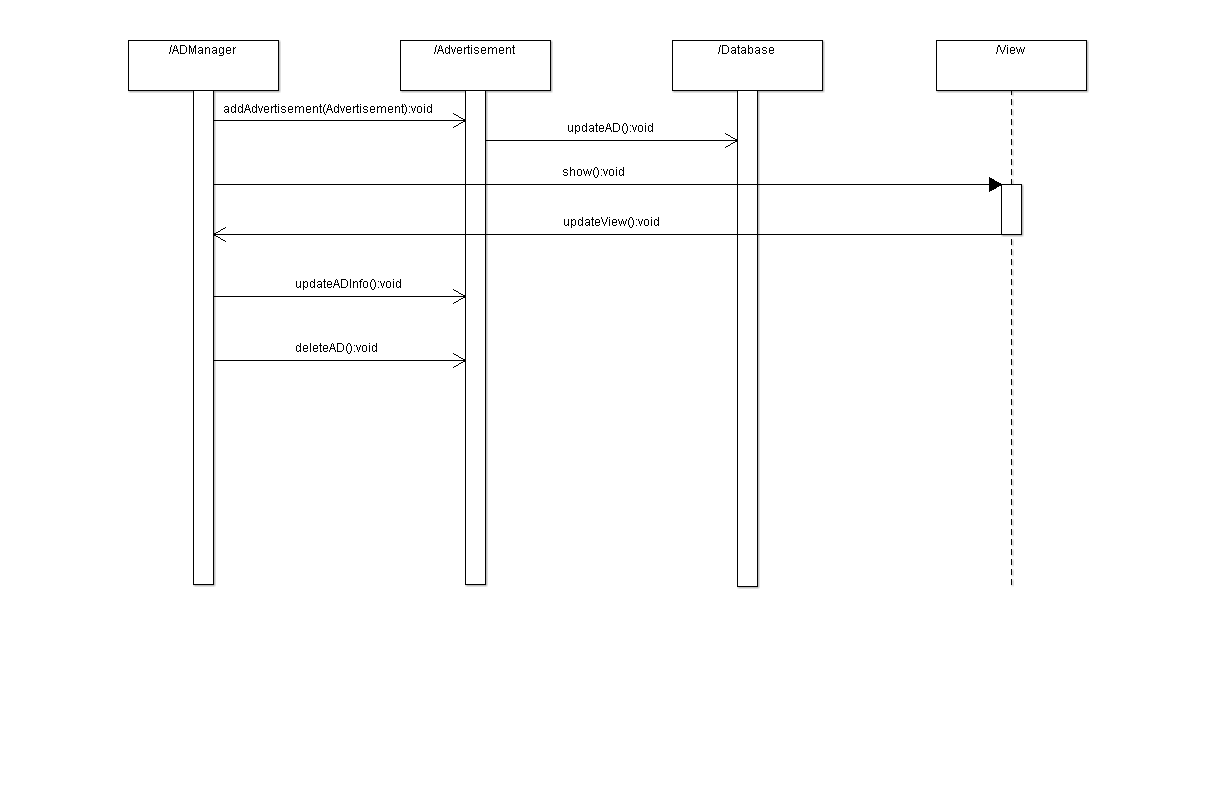
|  |  |
| --- | --- |
| 方法名 | Public void show() |
| 前置条件 | 数据库中广告不为空，管理员已经登陆 |
| 后置条件 | 无 |
| 作用 | 显示所有广告 |

|  |  |
| --- | --- |
| 方法名 | Public void init() |
| 前置条件 | 管理员已经登陆 |
| 后置条件 | 数据库中广告都进行初始化 |
| 作用 | 在数据库中初始化广告 |

数据结构描述：

|  |  |
| --- | --- |
| 数据结构 | 描述 |
| ArrayList AdList | 广告集合 |

## 1.4重要协作



协作描述：

广告管理模块通过实现IAdManager接口的AdManager类来进行广告的增加，删除，更新，初始化到数据库，然后通过界面显示出来

## 1.5使用的设计模式

## 1.5.1 MVC模式

该模块使用MVC模式构建，将网页，逻辑算法和数据库中的数据分离开

# 2.BuyGoods模块设计

## 2.1概述

### 2.1.1承担的需求（主要秘密）

该模块旨在满足用户的购买商品需求，用户在进行搜索得到合适的商品后，可以将商品加入购物车，并进行网银付款。

涉及需求

R30 购买商品 提供指向购物网站商家的链接

### 2.1.2可能会修改的实现（次要秘密）

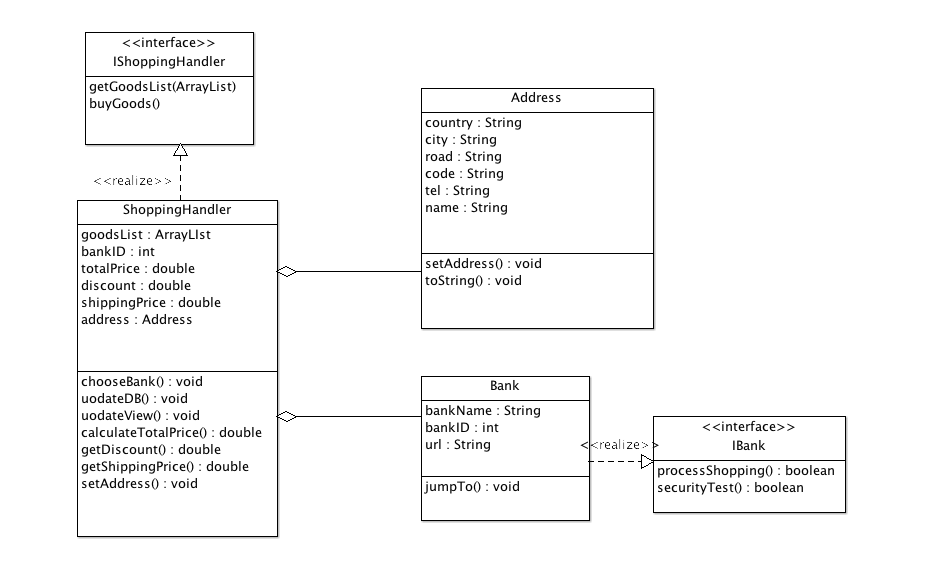
用户进行网银付款时候系统所支持的银行类别

### 2.1.3角色

### 2.1.4对外接口

|  |  |
| --- | --- |
| **接口名** | **职责** |
| Public void updateShoppingInfo (Shopping info) | 在数据库中更新支付内容信息 |
| Public void buyGoods(Shopping info) | 购买商品 |

## 2.2类图



## 2.3类描述

### 2.3.1 IshoppingHandler接口

该接口实现购买商品，使用策略模式将ShoppingHandler的方法抽象出来，便于以后进行改进

### 2.3.2 ShoppingHandler类

该类的职责是获得用户所选取的所有商品，计算总价、运费、要求用户填写收获地址以及联系方式，待用户确定购买之后，提示用户选择相应网上银行，用户付款之后进行数据库的更新，并更新网页信息，告知用户购买结果。

类方法：

|  |  |
| --- | --- |
| 方法名 | Public void getGoodsList(ArrayList list) |
| 前置条件 | 用户已经登陆，list不为nulll |
| 后置条件 | 显示用户选择的所有商品 |
| 作用 | 从用户输入模块获得用户所选择的所有商品 |

|  |  |
| --- | --- |
| 方法名 | Public void buyGoods() |
| 前置条件 | 用户已经登陆，用户点击提交按钮 |
| 后置条件 | 跳转到相应的购买页面 |
| 作用 | 用户点击提交按钮以后进行商品的购买 |

|  |  |
| --- | --- |
| 方法名 | Public void chooseBank(Bank bank) |
| 前置条件 | 用户已经登陆，选择银行种类 |
| 后置条件 | 跳转到下一信息填写页面 |
| 作用 | 供用户选择所要使用的银行种类 |

|  |  |
| --- | --- |
| 方法名 | Public void updateDB() |
| 前置条件 | 用户已经登陆，用户成功购买商品 |
| 后置条件 | 数据库信息进行更新 |
| 作用 | 用户购买成功以后，进行数据库的更新操作 |

|  |  |
| --- | --- |
| 方法名 | Public void updateView() |
| 前置条件 | 用户已经登陆，用户成功购买商品 |
| 后置条件 | 跳转到购买成功后页面 |
| 作用 | 用户购买成功之后，进行网页的更新 |

|  |  |
| --- | --- |
| 方法名 | Public void calculateTotalPrice() |
| 前置条件 | 用户已经登陆，用户选定需要购买的商品 |
| 后置条件 | 显示选定商品的总价格 |
| 作用 | 计算所有商品的总额 |

|  |  |
| --- | --- |
| 方法名 | Public ArrayList<String> getDiscount() |
| 前置条件 | 用户已经登陆 |
| 后置条件 | 显示所有打折活动信息 |
| 作用 | 检查是否有打折活动 |

|  |  |
| --- | --- |
| 方法名 | Public void getShippingPrice() |
| 前置条件 | 用户已经登陆,用户选定地址 |
| 后置条件 | 显示数据库中该地址对应的运费 |
| 作用 | 查询数据库，根据用户的所在地获得运费 |

|  |  |
| --- | --- |
| 方法名 | Public void setAddress(String address) |
| 前置条件 | 用户已经登陆，用户填写好地址 |
| 后置条件 | 在数据库中更新好地址信息 |
| 作用 | 存储用户填写的地址 |

|  |  |
| --- | --- |
| 方法名 | Public void chooseAddress() |
| 前置条件 | 用户已经登陆 |
| 后置条件 | 显示数据库中存储的用户常用地址 |
| 作用 | 根据用户以前填写的地址状况，供用户选择常用地址 |

### 2.3.3 Address类

该类的职责是存储用户的地址，以便用户在购买时填写

类方法

|  |  |
| --- | --- |
| 方法名 | Public void setAddress(Addressaddress) |
| 前置条件 | 用户已经登陆，已经填写好地址信息 |
| 后置条件 | 在数据库中更新地址信息 |
| 作用 | 设置用户的地址 |

|  |  |
| --- | --- |
| 方法名 | Public String toString(Addressaddress) |
| 前置条件 | Address不为null |
| 后置条件 | 返回String格式的地址 |
| 作用 | 将地址变成String的格式，提供给外部模块 |

### 2.3.4 IBank接口

改接口将不同银行共有的方法抽象出来，方便以后添加新的功能

### 2.3.5 Bank类

该类实现了IBank接口，职责是在用户选择银行的时候，查询数据库，将用户引导到正确的网银付款窗口

类方法

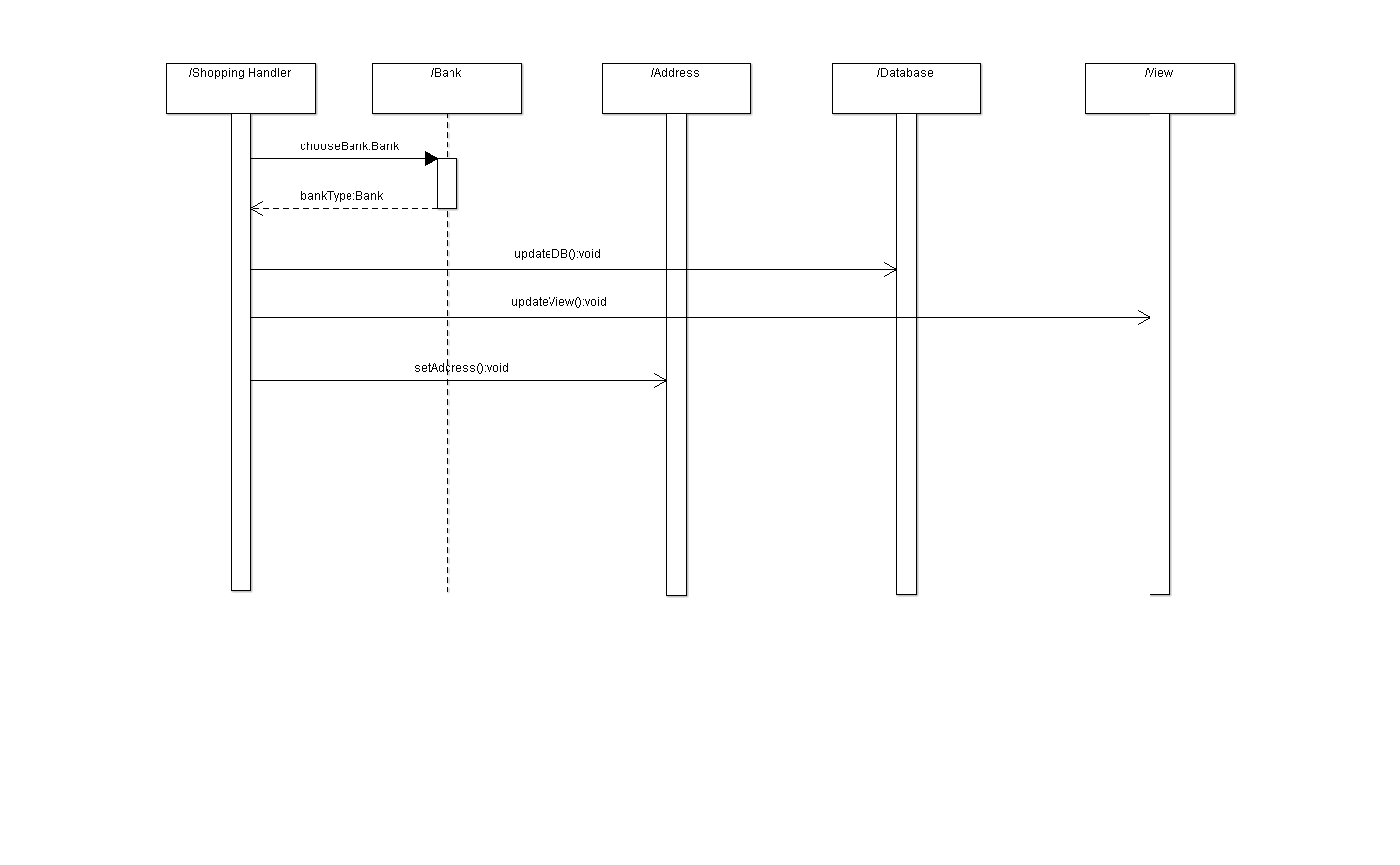
|  |  |
| --- | --- |
| 方法名 | Public void jumpTo(String link) |
| 前置条件 | 用户已经登陆，用户已经选择好支付银行 |
| 后置条件 | 网页跳转到正确的付款窗口 |
| 作用 | 根据用户选择的银行，查询数据库，将网页跳转到正确的付款窗口 |

|  |  |
| --- | --- |
| 方法名 | Public Boolean processShopping(Shopping info) |
| 前置条件 | 用户已经登陆，用户已经选定商品并选择好支付银行 |
| 后置条件 | 显示执行结果是否成功 |
| 作用 | 执行付款操作，根据银行的不同种类，该方法有不同的实现方式。该方法返回执行结果的布尔值 |

|  |  |
| --- | --- |
| 方法名 | Public Boolean securityTest() |
| 前置条件 | 用户已经登陆，用户已经填写好支付信息 |
| 后置条件 | 显示检测的结果是否满足付款要求 |
| 作用 | 检测当前网页安全环境是否满足付款要求，返回检测结果的布尔值。根据不同的银行种类，该方法有不同的实现。 |

数据结构描述

## 2.4重要协作



协作描述

商品购买模块从选定商品进行购买后，设置支付的银行，地址等信息，然后将信息更新到数据库中，并且在界面上显示出来。

## 2.5使用的设计模式

### 2.5.1策略模式

将银行进行付款的算法封装出来，便于以后修改。保证了高内聚和低耦合

### 2.5.2 MVC模式

该模块使用MVC模式构建，将网页、逻辑算法和数据库的数据分开