# 详细设计

1. 概述
   1. 系统简述

本区块链系统采用前后端分离的设计思想，用于打造一个银行、企业、物流和保险公司之间的区块链系统。

* 1. 软件设计目标

本软件的设计目标是最终实现一个具有良好用户交互体验和基本功能的web软件。具体包括：不同身份用户的注册与登陆、不同身份用户的基本操作等。本软件主要用于研究性学习而非实用，对安全性的要求不严格。在性能方面，应保证正常实用状态下无明显卡顿，并将信息传递与展示的时延控制在合理范围内。

* 1. 参考资料

《Applying UML and Patterns》

《Software Architecture in Practice》

1. 术语表

（对本文档中所使用的各种术语进行说明。如果一些术语在需求规格说明书中已经说明过了，此处不用再重复，可以指引读者参考需求说明。）

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

1. 设计概述
   1. 系统的复用计划

暂无复用计划

* 1. 系统接口设计
     1. 界面设计
        1. 登录界面：

登陆界面提供账号密码，用户输入账号密码选择登陆类型即可。



* + - 1. 注册界面

用户可以在注册界面输入注册账号的相关信息，例如用户名、密码、需要绑定的邮箱、公司名称和公司类型等。系统在执行成功注册后返回登陆界面



* + - 1. 主页面

银行主页面：



然后银行可以进入授信管理、抵押品管理、Token管理三个页面。

企业主页面：



企业可以进入合同管理、库存管理、物流管理、token管理、保单管理等页面。

物流公司主页面：



然后物流公司可以进入 物流管理、保单管理、Token管理等页面。

保险公司主页面：



保险公司可以保险管理、保险标管理、Token管理等页面。

* + 1. 系统外部接口设计：与外部系统的交互设计

系统与外部系统暂无交互

* + 1. 系统内部接口设计：各子系统、各模块间的接口设计
  1. 对象模型设计

后台数据库设计很简单，就只保存了用户的账号密码等信息，主要信息都还是存在区块链上。

struct Account { //8

string accNo; //账户号

string accType; //公司类型：银行、企业、物流、保险

string name; //公司名

//address privateKey; //私钥

//address publicKey; //公钥

//address walletAddress; //钱包地址

uint bal; //余额

string status; //用户状态(V:valid I:invalid)

//uint[] conId; //存储与自己有关的合同id

//uint[] traId; //存储与自己有关的物流id

//uint[] insId; //存储与自己有关的保险id

//uint[] invId; //存储与自己有关的库存id

//uint[] transferId; //存储与自己有关的转账id

//uint[] appId; //存储与自己有关的申请兑付id

//uint[] creId; //存储与自己有关的信用id(银行)

}

struct Contracts{ //8

uint conNo; //合同编号

string initiator; //发起方

string respondent; //回复方

string title; //合同名称

string text; //合同内容

uint money; //合同金额

string status; //合同状态

//string conTime; //签署合同时间

string updateTime; //更新合同状态时间

}

struct Transport{ //8

uint traNo; //物流订单编号

string initiator; //发起方

string receiver; //收货方

string respondent; //物流公司

string text; //订单内容

uint money; //订单金额

string status; //订单状态

//string conTime; //签署订单时间

string updateTime; //更新订单状态时间

}

struct Insurance{ //8+2?

uint insNo; //保险订单编号

string initiator; //发起人企业

string respondent; //保险公司

string text; //保单内容

string insType; //保险类型

uint money; //保单金额

string status; //保单状态

//string conTime; //签署保单时间

//string beginTime; //保单生效时间

//string endTime; //保单失效时间

string updateTime; //更新保单状态时间

}

struct Inventory{ //8

uint invNo; //库存编号

string holder; //持有人

string supplier; //供应商

string transport; //物流公司

uint traNo; //物流订单编号

string text; //入库内容

uint money; //交易金额

string updateTime; //入库时间

}

struct TransferInfo{ //5

uint traNo; //转账编号

string payer; //付款方

string payee; //收款方

uint money; //转账金额

string updateTime; //转账时间

}

struct PaymentApplication{ //6

uint appNo; //兑付申请编号

string initiator; //发起人企业/保险/物流

string respondent; //承兑方银行

uint money; //兑付金额

string status; //申请状态

//string appTime; //申请时间

string updateTime; //更新状态时间

}

struct Credit{ //7

uint creNo; //信用编号

string enterprise; //企业

string bank; //授信的银行

uint creditScore; //信用分

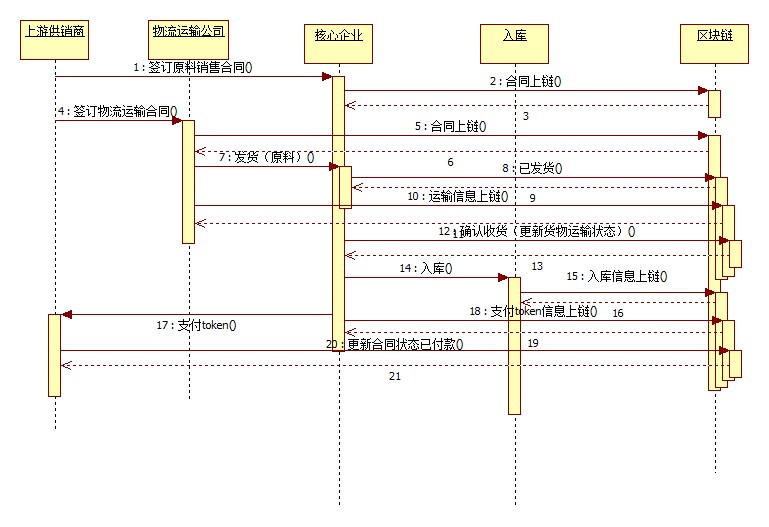
string updateStatus; //状态更新内容

string updateTime; //更新状态时间

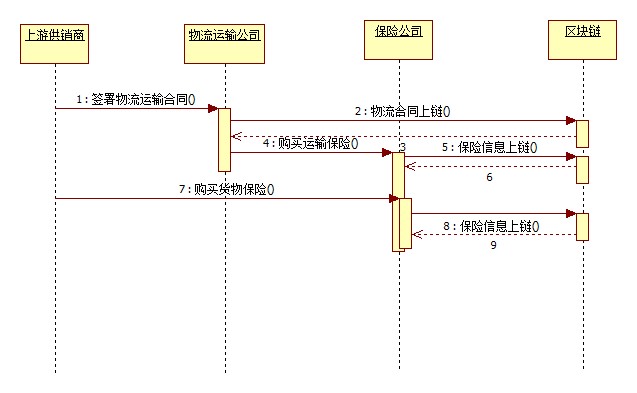
}

* 1. 系统用例实现详细设计

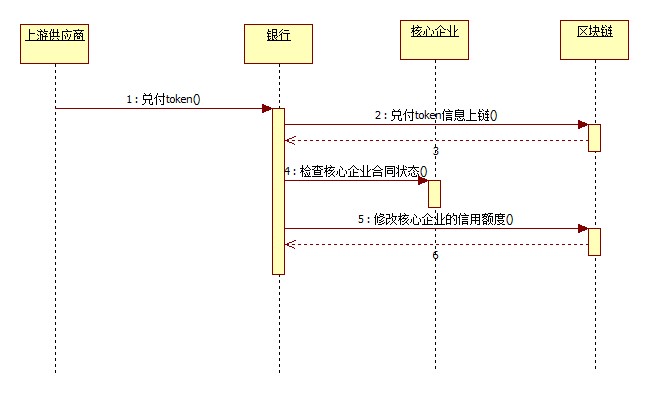
上游企业与核心企业签署订单并且交给物流公司运输货物顺序图：



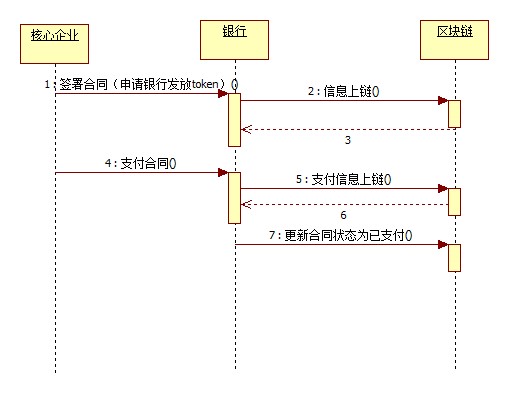
上游企业与物流签署运输合同并且购买保险顺序图：



供应商找银行兑付token顺序图：



银行授信核心企业并且修改核心企业信用额度顺序图：



* 1. 系统非功能设计
     1. 可用性
     2. 可靠性
        1. 安全性

需求：系统应该保障用户账号密码在网络传输中的安全性。

设计：对密码进行加密处理。

* + - 1. 容灾能力

需求：系统应该具备基本的容灾策略来保证服务的正常提供。

设计：如有必要，计划使用高可用性的物理部署方案

* + - 1. 性能

1. 实时性

需求：系统需要保证时间敏感类信息的实时转发。

设计：通过较为稳定的TCP连接（WebSocket连接）转发信息。

1. 数据传输格式

需求：系统的数据传输格式应该便于团队开发、调试，同时对系统运营有帮助。

设计：在开发前期，为了方便调试，使用简单的文本协议、JSON等数据格式，后期生产部署，出于流量等考虑，可转用Protobuf等体积更小的协议