1. 设计思想原则
2. 系统总体设计原则

为确保系统的建设成功与可持续发展，在系统的建设与技术方案设计时我们遵循如下的原则：1）、统一设计原则 统筹规划和统一设计系统结构。尤其是应用系统建设结构、数据模型结构、数据存储结构以及系统扩展规划等内容，均需从全局出发、从长远的角度考虑。

2）、先进性原则 系统构成必须采用成熟、具有国内先进水平，并符合国际发展趋势的技术、软件产品和设备。在设计过程中充分依照国际上的规范、标准，借鉴国内外目前成熟的主流网络和综合信息系统的体系结构，以保证系统具有较长的生命力和扩展能力。保证先进性的同时还要保证技术的稳定、安全性。

3）、高可靠/高安全性原则 系统设计和数据架构设计中充分考虑系统的安全和可靠。

4）、标准化原则 系统各项技术遵循国际标准、国家标准、行业和相关规范。

5）、成熟性原则 系统要采用国际主流、成熟的体系架构来构建，实现跨平台的应用。

6）、适用性原则 保护已有资源，急用先行，在满足应用需求的前提下，尽量降低建设成本。7）、可扩展性原则 信息系统设计要考虑到业务未来发展的需要，尽可能设计得简明，降低各功能模块耦合度，并充分考虑兼容性。系统能够支持对多种格式数据的存储。

1. 软件设计原则

1.开-闭原则(Open-Closed Principle, OCP)

2.里氏代换原则(Liskov Substitution Principle,常缩写为.LSP)

3.依赖倒置原则(Dependence Inversion Principle)

4.接口隔离原则(Interface Segregation Principle, ISP)

5.合成/聚合复用原则(Composite/Aggregate Reuse Principle,CARP)

6.迪米特法则(Law of Demeter LoD)又叫做最少知识原则(Least Knowledge Principle,LKP)

7.单一职责原则(Simple responsibility pinciple SRP)

1. 分布式系统设计原则

分布式系统需要达到如下目标：

1).数据存储的分区容错，冗余

2).应用的大访问、高性能要求

3).应用的高可用要求，故障转移

分布式系统需要遵循以下原则：

1).符合CAP原理

CAP三要素

* 一致性(Consistency)
* 可用性(Availability)
* 分区容忍性(Partition tolerance)

2). **最终一致性**

3).可扩展性

4).可监控性

5).安全性

1. 业务应用支撑平台设计原则

业务应用支撑平台的设计遵循了以下原则：

1、遵循相关规范或标准 遵循Java、XML、JSON、JDBC、HTTP、TCP/IP、SSL等业界主流标准

2、采用先进和成熟的技术 系统采用三层体系结构，使用JSON规范作为信息交互的标准，充分吸收国际厂商的先进经验，并且采用先进、成熟的软硬件支撑平台及相关标准作为系统的基础。

3、可灵活的与其他系统集成 系统采用基于工业标准的技术，方便与其他系统的集成。

4、快速开发/快速修改的原则 系统提供了灵活的二次开发手段，在面向组件的应用框架上，能够在不影响系统情况下快速开发新业务、增加新功能，同时提供方便地对业务进行修改和动态加载的支持，保障应用系统应能够方便支持集中的版本控制与升级管理。

5、具有良好的可扩展性 系统能够支持硬件、系统软件、应用软件多个层面的可扩展性，能够实现快速开发/重组、业务参数配置、业务功能二次开发等多个方面使得系统可以支持未来不断变化的特征。

6、平台无关性 系统能够适应多种主流主机平台、数据库平台、中间件平台，具有较强的跨系统平台的能力。

7、安全性和可靠性 系统能保证数据安全一致，高度可靠，应提供多种检查和处理手段，保证系统的准确性。针对主机、数据库、网络、应用等各层次制定相应的安全策略和可靠性策略保障系统的安全性和可靠性。

8、用户操作方便的原则 系统提供统一的界面风格，可为每个用户群，包括客户，提供一个一致的、个性化定制的和易于使用的操作界面。

9、应支持多CPU的SMP对称多处理结构

1. 系统架构设计

系统采用微服务架构，微服务与传统架构模式相比，具有语言无关性、独立进程通讯、高度解耦、任务边界固定、按需扩展等特点。

1. 微服务定义

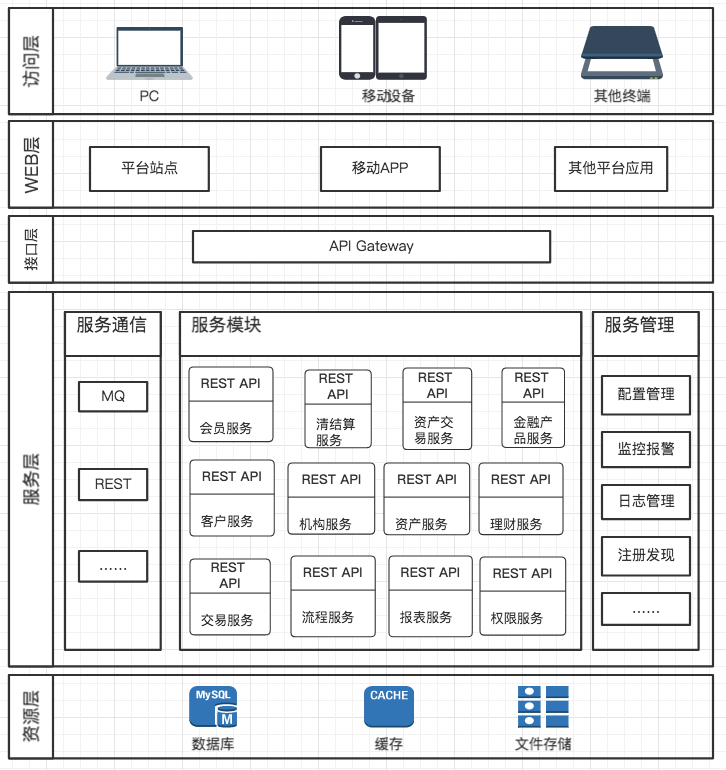
微服务是一种架构风格，一个大型复杂软件应用由一个或多个微服务组成。系统中的各个微服务可被独立部署，各个微服务之间是松耦合的。每个微服务仅关注于完成一件任务并很好地完成该任务。在所有情况下，每个任务代表着一个小的业务能力。

微服务的概念源于2014年3月Martin Fowler所写的一篇文章“Microservices”(<http://martinfowler.com/articles/microservices.html>)。

1. 微服务架构的一些通用特性

根据MartinFowler的分析，微服务架构有以下的一些通用特性，但并非所有微服务架构应用都必须具备所有这些特性：

1. 通过服务实现应用的组件化(Componentizationvia Services)：微服务架构中将组件定义为可被独立替换和升级的软件单元，在应用架构设计中通过将整体应用切分成可独立部署及升级的微服务方式进行组件化设计。
2. 围绕业务能力组织服务(Organizedaround Business Capabilities)：微服务架构采取以业务能力为出发点组织服务的策略，因此微服务团队的组织结构必须是跨功能的（如：既管应用，也管数据库）、强搭配的DevOps开发运维一体化团队，通常这些团队不会太大。
3. 产品而非项目模式(Productsnot Projects)：传统的应用模式是一个团队以项目模式开发完整的应用，开发完成后就交付给运维团队负责维护；微服务架构则倡导一个团队应该如开发产品般负责一个“微服务”完整的生命周期，倡导“谁开发，谁运营”的开发运维一体化方法。
4. 智能端点与管道扁平化(Smartendpoints and dumb pipes)：微服务架构主张将组件间通讯的相关业务逻辑/智能放在组件端点侧而非放在通讯组件中，通讯机制或组件应该尽量简单及松耦合。RESTful HTTP协议和仅提供消息路由功能的轻量级异步机制是微服务架构中最常用的通讯机制。
5. “去中心化”治理(DecentralizedGovernance)：整体式应用往往倾向于采用单一技术平台，微服务架构则鼓励使用合适的工具完成各自的任务，每个微服务可以考虑选用最佳工具完成(如不同的编程语言)。微服务的技术标准倾向于寻找其他开发者已成功验证解决类似问题的技术。
6. “去中心化”数据管理(DecentralizedData Management)：微服务架构倡导采用多样性持久化(PolyglotPersistence)的方法，让每个微服务管理其自有数据库，并允许不同微服务采用不同的数据持久化技术。
7. 基础设施自动化(InfrastructureAutomation)：云化及自动化部署等技术极大地降低了微服务构建、部署和运维的难度，通过应用持续集成和持续交付等方法有助于达到加速推出市场的目的。
8. 故障处理设计(Designfor failure)：微服务架构所带来的一个后果是必须考虑每个服务的失败容错机制。因此，微服务非常重视建立架构及业务相关指标的实时监控和日志机制。
9. 演进式的设计(EvolutionaryDesign)：微服务应用更注重快速更新，因此系统的计会随时间不断变化及演进。微服务的设计受业务功能的生命周期等因素影响。如某应用是整体式应用，但逐渐朝微应用架构方向演进，整体式应用仍是核心，但新功能将使用应用所提供的API构建。再如在某微服务应用中，可替代性模块化设计的基本原则，在实施后发现某两个微服务经常必须同时更新，则这很可能意味着应将其合并为一个微服务。
10. 系统分层图如下：



访问层：用户通过PC、手机登访问平台

WEB层：平台网站、移动端接口，提供给访问层使用

接口层：API Gateway完成授权鉴权、路由负载功能

服务层：

服务模块：各个业务模块，提供REST API接口调用，可以直接提供给用户使用，也可以给其他服务进行调用，也可提供业务功能的聚合，将各个服务组件加上业务操作提供给用户使用

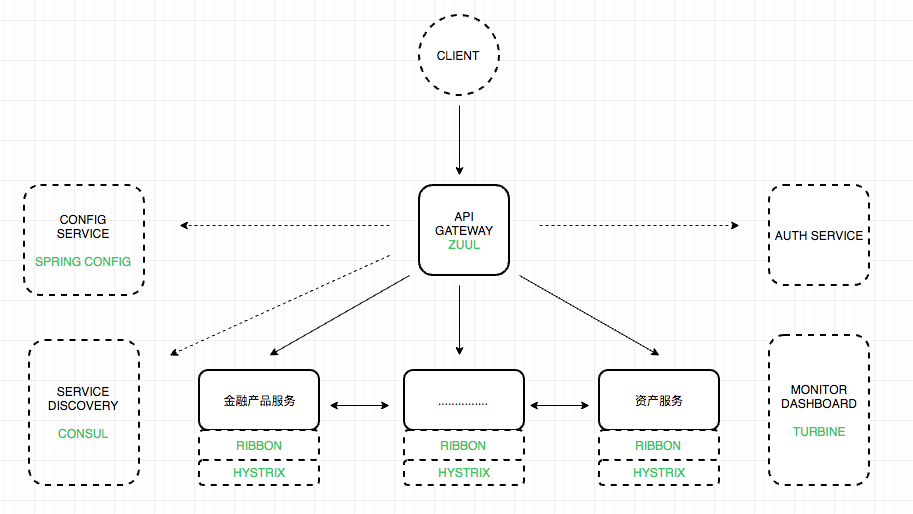
服务通信：使用MQ支持异步消息，降低系统耦合性；使用REST API进行同步通信

服务管理：当服务数量增加是对各个服务进行管理和监控，方便系统拍错和升级

资源层：提供数据存储和数据的缓存

1. 系统架构图

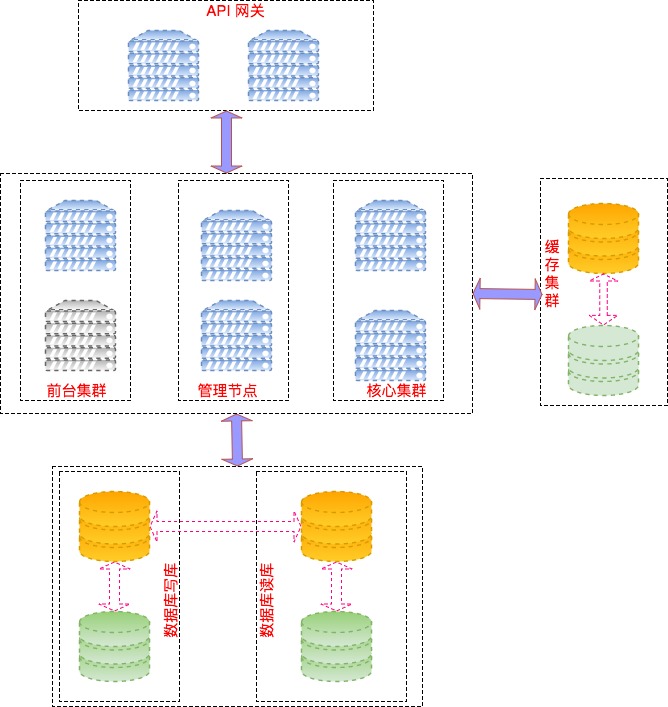
系统采用基于Java的Spring Boot 和Spring Cloud技术体系。系统架构图如下：



* 金融产品服务、资产服务等服务在启动时向服务注册中心（CONSUL）注册自己的信息，方便进行服务之间的查找
* 各个服务互相调用时需要向服务注册中心查询需要使用的服务，并通过Ribbon实现负载均衡，并在调用过程中增加了断路器（Hystrix）的功能；
* 由于服务发现后才能调用，因此金融产品服务、资产服务等服务通过注册中心（CONSUL）实现互相发现；
* API Gateway（Zuul）提供对外统一的服务网关，首先从注册中心（CONSUL）处获取相应服务，再根据服务调用各个服务的真实业务逻辑；
* 服务调用过程通过聚合器（Turbine）统一所有断路信息；
* 整个业务过程中所有服务的配置文件通过Spring Cloud Config来管理，即起什么端口、配置什么参数等；
* 认证机制通过Auth service实现，提供基本认证、鉴权服务。

1. 平台构成/系统结构设计

系统拓扑图

****

系统简单分为 API 网关、应用层、缓存层、数据层四层。

* API 网关主要负责授权以及相关路由负载
* 应用层主要分为前台系统、核心交易系统、管理系统三大核心模块。
* 前端系统
  + 会员管理
  + 理财频道
  + 资产交易大厅
* 核心系统
  + 计算服务
  + 后台系统
  + 金融产品管理
  + 资产管理
  + 清结算管理
  + 机构管理
  + 客户管理
  + 权限管理
  + 交易管理
  + 流程管理
* 管理系统
  + 服务发现
  + 配置管理
  + 报表管理
* 缓存层主要负责session共享以及降低数据库压力
* 数据层存储有效数据

1. 软件及硬件设备配置方案建议

参考以上三中系统拓扑图，提供早期系统对应服务器配置参数：

蓝色服务器： 8核16G+100G磁盘x7

灰色服务器： 4核8G+100G磁盘x1

黄色数据库：8核16G+100G 磁盘x3

绿色数据库： 4核8G+100G 磁盘x3

|  |  |  |
| --- | --- | --- |
| **层次** | **配置** | **数目** |
| API | 8核16G+100G | 2 |
| 前台 | 8核16G+100G | 1 |
| 前台 | 4核8G+100G | 1 |
| 核心系统 | 8核16G+100G | 2 |
| 管理节点 | 8核16G+100G | 2 |
| 缓存 | 8核16G+100G | 2 |
| 数据库 | 8核16G+100G | 2 |