

# 软件架构重点归纳总结

## Useful architectural structures 主要架构结构

在书上第15页。

TABLE 1.1 Useful Architectural Structures					
	Software Structure	Element Types	Relations	Useful For	Quality Attributes Affected
Module Structures	Decomposition	Module	Is a submodule of	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control	Modifiability
	Uses	Module	Uses (i.e., requires the correct presence of)	Engineering subsets, engineering extensions	"Subsetability," extensibility
	Layers	Layer	Requires the correct presence of, uses the services of, provides abstraction to	Incremental development, implementing systems on top of "virtual machines"	Portability
	Class	Class, object	Is an instance of, shares access methods of	In object-oriented design systems, factoring out commonality; planning extensions of functionality	Modifiability, extensibility
C&C Structures	Data model	Data entity	(one, many)-to-(one, many), generalizes, specializes	Engineering global data structures for consistency and performance	Modifiability, performance
	Service	Service, ESB, registry, others	Runs concurrently with, may run concurrently with, excludes, precedes, etc.	Scheduling analysis, performance analysis	Interoperability, modifiability
	Concurrency	Processes, threads	Can run in parallel	Identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed	Performance, availability
Allocation Structures	Deployment	Components, hardware elements	Allocated to, migrates to	Performance, availability, security analysis	Performance, security, availability
	Implementation	Modules, file structure	Stored in	Configuration control, integration, test activities	Development efficiency
	Work assignment	Modules, organizational units	Assigned to	Project management, best use of expertise and available resources, management of commonality	Development efficiency

SHOT ON REDMI K20  
AI TRIPLE CAMERA

主要分三类：Module 模块 / C&C 组件和连接件 / Allocation 分配

## 为什么软件架构很重要

在书上第25页。

1. Influence quality attributes 影响质量属性
2. Help reason about and manage change as the system evolves 系统的发展有助于改变和管理变化的原因
3. Early prediction of a system's qualities 一个系统其质量的早期预测
4. Enhances communication among stakeholders 加强涉众沟通
5. Capture the earliest and hence most fundamental, hardest-to-change design decisions 捕捉最早最根本最难以改变的设计决策
6. Defines a set of constraints on subsequent implementation 定义了一组在随后执行上的约束
7. Dictates the structure of an organization, or vice versa 决定一个组织的结构，反之亦然
8. Provide the basis for evolutionary prototyping 为进化原型的设计提供依据
9. Allows the architect and project manager to reason about cost and schedule 让架构师和项目经理对成本和进度追根溯源
10. As a transferable, reusable model that from the heart of a product line 作为一种可转让可重用的模型是产品线的核心
11. Architecture-based development focuses attention on the assembly of components, rather than simply on their creation 以架构为基础的发展重点关注组件的部件，而不是他们的创建过程

12. Reducing design and system complexity 减少设计和系统的复杂度
13. Be the foundation for training a new team member 为培训新的团队成员奠定基础

## 创建软件架构的7个主要活动

在书上第45页。这7个活动与选择的架构和life-cycle model毫无关系，是必须经历的过程。

- Making a business case for the system 为系统制定业务案例
- Understanding the architecturally significant requirements 了解架构重要要求
- Creating or selecting the architecture 创建或选择架构
- Documenting and communicating the architecture 交流和记录架构
- Analyzing or evaluating the architecture 分析或评估架构
- Implementing and testing the system based on the architecture 基于架构实现并测试系统
- Ensuring that the implementation conforms to the architecture 确保实现符合架构

## 架构设计决策7个类别

在书上第73页。

1. Allocation of responsibilities 分配职责
2. Coordination model 协调模式
3. Data model 数据模型
4. Management of resources 资源管理
5. Mapping among architectural elements 架构元素之间的映射
6. Binding time decisions 绑定时间思想
7. Choice of technology 技术选择

## Availability 可用性

书p83的failure tree要知道怎么搞出来的

战术分三类：检测错误、从错误中恢复、防止错误

## Interoperability 互操作性

SOAP(simple object access protocol)和ReST(representation state transfer)是这章的重点。

### SOAP与REST

如果想让web应用互操作，有两种现成的技术可以使用：SOAP和REST。

SOAP是针对基于XML的信息的协议。分布式应用程序可以使用它来交换信息，从而实现互操作。它常伴随着一组SOA中间件互操作标准和他们的合理实现，称为web服务。SOAP定义了如下标准：

- 组合服务的基础架构。
- 传输标准。
- 服务发现。使用UDDI语言来发布、监听服务。
- 可靠性。SOAP本身不保证信息可靠传输。需要信息可靠传输的应用程序必须使用符合SOAP可靠性标准的服务，比如WS-Reliability。

SOAP非常通用，它的根源是交互应用程序的远程调用过程模型。SOAP具有简单的类型系统。SOAP依靠HTTP和RPC进行信息传输，但从理论上讲可以在任何传输协议上实现。SOAP并不要求服务的方法名称、寻址模型或过程约定，因此使用SOAP几乎不会在应用程序之间带来实际的互操作性，这只是一套信息交换标准。交互的应用程序需要就如何处理有效负载达成共识，这就是获得语义互操作性的地方。

REST基于客户端-服务器的体系结构，围绕一小组“创建、读取、更新、删除”(CRUD)操作(或者叫post, get, put, delete)和单地址寻址方案进行构造。REST对体系结构没有什么限制。与它相比，SOAP提供了完整性，而REST提供了简单性。

REST是关于状态和状态转移的，它将web服务视为一个巨大的信息网络，可通过单地址URI寻址方案进行访问。REST中没有类型的概念，因此也没有类型检查，这取决于应用程序是否正确获取交互语义。

由于REST接口简单通用，因此任何HTTP客户端都可以使用REST操作与任何HTTP服务器进行通信，而无需进一步配置，这带来了语法上的互操作性。但是，对于这些程序的实际作用以及它们交换的信息，必须在组织级别达成一致，也就是说，如果仅两个服务都有REST接口，就不能保证服务之间的语义互操作性。

在HTTP之上的REST旨在进行自我描述，并且在最佳情况下是无状态协议。

在SOAP和REST之间进行选择的一个方面是，是否要接受SOAP+WSDL的复杂性和限制以获得更标准化的互操作性，或者是否想通过使用REST来避免开销，也许会受益于标准化程度的降低。

REST中的信息交换比SOAP中的信息交换具有更少的字符。因此，在REST和SOAP之间进行选择的权衡之一就是各个信息的大小。对于大量交换信息的系统，另一个折衷是性能(选REST)和结构化信息(选SOAP)之间的权衡。

选择SOAP或REST将取决于以下方面。例如所需的服务质量，SOAP对安全性、可用性以及功能类型有更大的支持。REST由于其简单性，更适合于典型的只读功能，比如mashup，这种情况对服务的质量要求较低。

战术只分两类而且很少：定位服务、管理接口(编排接口、定制界面)

## Modifiability 可修改性

讲的东西不是很多

战术分四类：减少模块大小、提高内聚性(cohesion)、降低耦合度(coupling)、延迟绑定(deferred binding)

抓住高内聚低耦合就完事

## Performance 性能

书p132提到了并发，这是解决性能问题的好方法。总的来说，就是解决资源需求和有限提供的矛盾而已。

战术分两类：控制资源需求、管理资源(存储要合理)

就是能服务多少人就服务多少人，控制并发量

## Security 安全性

表征安全性有三个特征：机密性、完整性和可用性。机密性是保护数据或服务免遭未经授权访问的属性。完整性是数据或服务不受未经授权操作的属性。可用性是系统可用于合法使用的属性。除此之外还有认证性、不可否认性和授权性三个属性。

战术分四类：检测攻击、抵抗攻击、对攻击作出响应、从攻击中恢复。这四个跟可用性很像

## Testability 可测试性

讲的东西不是很多

战术分两类：观察并控制系统状态(废话)、限制复杂度。很容易想到

# Usability 易用性

设计清单(书p182)要看一下

战术分两类：支持用户主动操作、支持系统主动操作(很废话，就是两边都能操作)

## 其他质量属性

包括variability(可变性), portability(可移植性), development distributability(开发分布性), scalability(可伸缩性), deployability(可部署性), mobility(移动性), monitorability(可监视性), safety(安全性)

## 架构战术与模式(书Chap.13，超级重点)

架构模式是在实践中反复出现的一整套设计决策，它具有已知的属性，可以重复使用，且描述了一类架构。架构不是“发明”出来的，而是“发现”出来的。所以永远不会有完整的架构模式列表，一旦某些条件发生变化就会产生新的架构模式。经验丰富的架构师通常将创建架构视为选择、定制和组合架构模式的过程。与架构模式相比，战术作用范围更小，但更精细。可以说，战术是原子，而架构模式是分子。所以说：模式对战术进行了打包(整合)。

架构模式与以下成员建立联系：

- 语境(上下文)。产生问题的环境。
- 问题。架构模式概述了问题及其实体，并描述了任何互补或相反的作用力。问题的概述通常包含会遇到的质量属性。
- 解决方案。解决方案是成功解决问题并进行适当抽象的架构，描述了解决问题的体系结构，包括如何平衡工作中的多种力量，描述元素之间的职责和静态关系(使用模块结构)，或者描述元素的运行时行为以及元素之间的交互(C2C结构或分配结构)。解决方案由以下方式确定和描述：
  - 一系列元素类型。比如数据储存库，流程和对象。
  - 一系列交互机制和连接器。比如方法调用，事件和信息巴士。
  - 关于组件的拓扑布局。
  - 一组语义约束。它包含拓扑，元素行为和交互机制。

解决方案应解释清楚元素的静态关系和运行时行为提供了何种质量属性。像下面这样。

- Overview: ...
- Elements: ...
- Relations: ...
- Constraints: ...
- Weaknesses: ...

复杂的系统展现出多种模式。基于web的系统可能会采用三层C/S架构模式，在此架构模式下可能还会使用复制、代理、缓存、防火墙、MVC等，每一种架构模式都可能会采用更多的模式和策略。

## Module Patterns 模块架构模式

### Layered Pattern 层次模式

上下文：所有复杂的系统都需要独立开发和演化系统的各个部分。由于这个原因，系统的开发人员需要明确且有据可查的关注点分离，以便可以独立开发和维护系统的模块。

问题：需要对软件分段，使得模块可以单独开发，而部件之间几乎没有交互，支持可移植性，可修改性和重用。

解决方案：将软件以层为单位划分。每个层为一组模块，提供一组连贯的服务。使用方向必须为单向(称为allowed-to-use关系，其实就是depends-on关系的专业表述)。每层通过公共接口公开出去。

约束：

- 软件的每一段必须归属于某一层
- 至少要有两层
- allowed-to-use关系不能成环

弱点：

- 增加层会增加系统的前期成本和复杂性
- 每一层对性能都有负面影响

## Component-and-Connector Patterns 组件-连接件架构模式

### Broker Pattern 代理(经纪人)模式

上下文：许多系统被一系列服务构件，并部署在多台服务器上。

问题：如何构建分布式软件以便服务用户，满足不需要知道服务提供商的性质和位置就可以容易地动态更改用户和提供商之间的绑定？

解决方案：代理模式通过插入一个称为代理的中介，将服务的用户(客户端)与服务的提供者(服务器)分离。当客户端需要服务时，它对代理提供的服务接口发送请求。代理把该请求转发到服务器。

关系：附属关系将客户端和服务器与代理相关联。

约束：客户端和服务端都只能与代理交互。

弱点：

- 代理在客户端和服务器之间添加了一个间接层，因此增加了延迟，并且该层可能是通信瓶颈。
- 代理可能出现单点故障。
- 代理增加了前期的复杂性。
- 代理可能是安全攻击的目标。
- 代理可能很难测试。

### Model-View-Controller Pattern MVC架构模式

上下文：把视图与系统的其余部分分开。

问题：如何将用户界面功能与应用程序功能分开，但仍然响应用户输入或低层应用程序数据的更改？当底层应用程序数据发生变化时，如何创建、维护和协调用户界面的多个视图？

解决方案：使用MVC模式将应用程序功能分为三种类型的组件：

- Model，一个模型，存放应用程序数据。
- View，一个视图，展示部分数据并与用户交互。
- Controller，一个控制器，在模型和视图之间进行中介，并管理状态更改的通知。

MVC不是对所有情况都适用。

关系：通知关系。连接模型、视图和控制器的实例，并通知相关状态更改元素。

约束：MVC至少各有一个实例。

弱点：

- 对于简单用户接口，复杂度可能会很高
- MVC抽象形式可能不能与一些用户接口工具包良好匹配。

### Pipe-and-Filter Patter 管道和过滤器架构模式

上下文：许多系统要求从输入到输出传输离散数据流，许多类型的传输手段重复发生。

问题：系统需要被分成可重复使用的松散耦合的组件，这些组件具有简单的通用交互机制。

解决方案：使用管道-过滤器模式，其交互模式的特征在于数据流的连续变换。数据到达过滤器的输入端口，进行转换，然后通过管道的输出端口通过管道传递到下一个过滤器。一个信号过滤器可以从一个或多个端口消耗数据或向一个或多个端口产生数据。

关系：附件关系将过滤器的输出与管道的输入关联，反之亦然。

约束：

- 管道将过滤器输出端口连接到过滤器输入端口。
- 连接的过滤器必须就通过连接管道传递的数据类型达成共识。
- 模式的特殊化可能会将组件的关联限制为无环图或线性序列，有时也称为管线。
- 其他特殊化可能规定组件具有某些命名的端口，例如UNIX过滤器的stdin，stdout和stderr端口。

弱点：

- 管道和过滤器模式通常不是交互式系统的理想选择。
- 拥有大量独立的过滤器会增加大量的计算开销。
- 管道和过滤器系统可能不适用于长时间运行的计算。

### **Client-Server Pattern C/S架构模式**

上下文：有大量分布式客户端希望访问的共享资源和服务。

问题：我们希望通过集中控制这些资源和服务来提高可伸缩性和可用性，同事将资源本身分布在多个物理服务器上。

解决方案：客户端通过请求服务器的服务进行交互，服务器提供一系列服务。

关系：客户端和服务端是附属关系。

约束：

- 客户端和服务器通过请求/响应连接件连接
- 服务端可以充当其他服务端的客户端
- 专业化时可能添加约束条件
  - 给定端口的附件数量
  - 服务端允许的关系
- 组件可以按层排列，这些层是相关功能或将共享主机计算环境的功能的逻辑分组。

弱点：

- 服务端可能是性能瓶颈
- 服务端可能出现单点故障
- 一个系统建立后，决定每部分的功能通常是复杂的且花费很大。

### **Peer-to-Peer Pattern P2P架构模式**

上下文：分布式计算实体被认为是对等的。

问题：一组对等的分布式计算实体如何通过公共协议相互连接，以便它们能够以高可用性和可扩展性组织和共享其服务？

解决方案：在P2P模式中，组件直接作为对等体进行交互。所有对等体都是“相等的”，并且没有对等体或对等体组对于系统的健康是至关重要的。

关系：附属关系将对等方与其连接器相关联。附属关系可能会在运行时更改。

约束：

- 任何给定对等方允许的依赖数
- 用于搜索对等体的跳数
- 哪些对等体知道哪些对等体

有些P2P网络构造成星形，每个节点只能知道其父节点。

弱点：

- 管理安全性，数据一致性，数据(服务)可用性，备份和恢复都很复杂
- 小的P2P系统可能不能满足性能和可用性这两个质量属性

### Service-Oriented Architecture Pattern 面向服务的(SOA)架构模式

上下文：许多服务互操作，但是对他们的实现没有任何详细的了解。

问题：如何支持在不同平台上运行并以不同实现语言编写的、由不同组织提供并分布在因特网上的分布式组件的互操作性？

解决方案：面向服务的体系结构(SOA)模式描述了提供和/或使用服务的分布式组件的集合。在SOA中，提供和服务使用者组件的服务提供者组件可以使用不同的实现语言和平台。服务在很大程度上是独立的：服务提供商和服务使用者通常是独立部署的，并且通常属于不同的系统甚至不同的组织。组件具有描述他们从其他组件请求的服务以及它们提供的服务的接口。

SOA连接件的基本类型包括SOAP、REST、asynchronous messaging(异步消息传递)。

关系：将各种类型的组件连接到各个连接器。

约束：服务使用者连接到服务提供商，但是可以使用中间组件。

弱点：

- 基于SOA的系统通常构建起来很复杂。
- 无法控制独立服务的发展。
- 与中间件相关联的性能开销很大，服务可能是性能瓶颈，通常不提供性能保证。

### Publish-Subscribe Pattern 发布-订阅架构模式

上下文：数据生产者和消费者的确切数量和性质不是预先确定的或固定的，也不是它们共享的数据。

问题：如何创建支持在生产者和消费者之间传输消息的能力的集成机制，使他们互不知道对方的身份甚至存在？

解决方案：在发布-订阅模式中，组件通过已发布的消息或事件进行交互。

关系：附属关系通过规定哪些组件宣布事件以及哪些组件已注册以接收事件，从而将组件与发布-订阅连接器相关联。

约束：

- 所有组件都连接到事件分配器，该事件分配器可以看作是总线连接器或组件。附加了发布端口以声明角色，并附加了订阅端口以监听角色。约束可能会限制哪些组件可以侦听哪些事件，某个组件是否可以侦听其自己的事件以及系统中可以存在多少个发布-订阅连接器。
- 同时具有两种类型的端口，组件可以既是发布者又是订阅者。

弱点：

- 通常会增加等待时间，并对消息传递时间的可伸缩性和可预测性产生负面影响。
- 对消息顺序的控制较少，不能保证消息的传递。

### Shared-Data Pattern 共享数据模式

上下文：各种计算组件需要共享和操作大量的数据。

问题：系统如何存储和操作由多个独立组件访问的持久数据？

解决方案：在共享数据模式中，交互主要由多个数据访问器和至少一个共享数据存储之间的持久数据交换所主导。

关系：附属关系确定哪些数据访问器连接到哪些数据存储。

约束：数据访问器与数据存储交互。

弱点：

- 共享数据存储可能成为性能瓶颈。
- 共享数据存储可能发生单点故障。
- 数据生产者和消费者可能紧密成对。

## Allocation Patterns 分配架构模式

### Map-Reduce Pattern 映射-减少模式

上下文：业务部门迫切需要以PB级快速分析其生成或访问的大量数据。示例包括社交网站中的交互日志，大量文档或数据存储库以及搜索引擎的成对的<source, target> Web链接。用于分析此数据的程序应易于编写，高效运行，并且在硬件故障方面具有弹性。

问题：有效地执行大型数据集的分布式并行排序，并为程序员指定要执行的分析提供了一种简单的方法。

解决方案：map-reduce模式需要三个部分：一个负责根据需要分配数据的专用基础设施；一个用于过滤数据以检索项目的map；一个结合了映射结果的reduce。

关系：“部署于”是映射-减少的实例与其所安装的处理器之间的关系。实例化，监视和控制是基础结构与映射-减少实例之间的关系。

约束：

- 被分析的数据必须以文件的形式保存
- 映射函数必须是状态无关的且不能互相沟通
- 映射实例与削减实例的唯一沟通是从映射实例发出的<key,value>键值对。

弱点：

- 如果没有大数据集，就没有理由使用映射-减少模式
- 如果没有把大数据集分割成若干个子集，并行的优势将会丧失
- 多重削减的操作将难以实现

### Multi-tier Pattern 多级模式

上下文：在分布式部署中，通常需要将系统的基础结构分布到不同的子集中。这可能是出于运营或业务原因。

问题：我们如何将系统拆分为由某些通信介质连接的许多计算独立的执行结构？

解决方案：许多系统的执行结构被组织为一系列组件的逻辑分组。每个分组称为级。

关系：xx是xx的一部分，xx与xx沟通，xx分配给xx

约束：一个软件组件恰好属于一个层

弱点：大量的前期成本和复杂性

## 架构模式和战术的关系

模式包括战术，也可以使用战术在增强模式

## 代理模式的缺点(书上特地提到)

- 可用性。服务器，代理甚至客户端的活动都需要进行监控，并且必须提供修复机制。
- 性能。客户端和服务器之间的间接级别增加了开销，因此增加了延迟。同样，如果不需要客户端和服务器之间的直接通信，则代理是潜在的性能瓶颈。
- 可测试性。代理用于复杂的多进程和多处理器系统中。这样的系统通常是高度动态的，请求和响应通常是异步的，这些使测试和调试此类系统极为困难。但代理模式的描述不提供测试功能，例如测试接口，状态或活动捕获和回放功能，等等。
- 安全性。因为代理模式主要在系统跨越进程和处理器边界（例如在基于Web的系统上）时使用，所以安全是一个合理的问题。但是，所提供的代理模式不提供任何身份验证或授权客户端或服务器的方法，也不提供保护客户端与服务器之间的通信的方法。

使用什么战术来改进这一模式呢？

- 增加可用资源性能策略将导致多个代理，以帮助提高性能和可用性。
- 维护多副本策略将允许这些代理中的每个共享状态，以确保它们对客户端请求的响应相同。
- 负载平衡将确保一个代理不过载，而另一个代理处于空闲状态。
- 心跳，异常检测或ping / echo将为复制的代理提供一种通知客户端并在其中一个客户端停止服务时互相通知的方法，以作为检测故障的一种方法。

## 质量属性建模和分析

林连南曾经有道题(可以看到本文最后图片，第7题)，让大家建立一个模型来分析MVC性能。这其实是书上p252~254的内容。待续。

## 敏捷项目中的架构

主要是书p275，敏捷和体系结构的关系：从来不是二选一。虽然不能很好地共存，但二者皆不可缺少。

书上p279的An analytic perspective on up-front work vs. agility。主要使用了the constructive cost model II (COCOMO II)来对多个项目进行分析。书p280的图反映了项目整体时间(y)和架构和风险的解决时间(x)的关系(都是百分比)。Sweet point就是让项目总时间最少的前期工作时间比。可以看到，KSLOC(thousands of source lines of code)越大的项目，前期工作做得越差，项目总时间越多，越不太可能敏捷。

早熟的问题可以通过重构解决。

## 架构和要求(书Chap. 16，重点)

架构上的重要要求(architecturally significant requirement，ASR)是对架构有深远影响的要求。

获取ASR的方法：

1. Gathering ASRs from requirements documents 从需求文档中获取。
2. Gathering ASRs by interviewing stakeholders 采访涉众，这里谈到了Quality Attribute Workshop(QAW，质量属性研讨会)。QAW专注于系统级的关注，特别是软件在系统中的作用。
3. Gathering ASRs by understanding the business goals 理解业务目标。这里谈到了the Pedigreed Attribute eLicitation Method(PALM，属性激励方法)。其7个步骤为：
  1. PALM overview presentation PALM概述演示
  2. Business drivers presentation 业务驱动力介绍
  3. Architecture drivers presentation 架构驱动程序演示
  4. Business goals elicitation 业务目标启发
  5. Identification of potential quality attributes from business goals 从业务目标中识别潜在的质量属性

6. Assignment of pedigree to existing quality attribute drivers 将谱系分配给现有的质量属性动因
  7. Exercise conclusion 结论
4. Capturing ASRs in a utility tree 在效用树中捕获。

ASR必须具有以下特征：

- 对架构的深刻影响。
- 较高的业务或使命价值。

书p307给了个很全的效用树，写成了表格的形式，不过是把根节点隐去而已。

## 设计架构

主要是书p316的质量驱动设计方法(The attribute-driven design method, ADD)，它可以做到：

- 选择系统的一部分进行设计
- 整理该部分在架构上的所有重要要求
- 创建并测试该部分的设计

在设计架构之前，架构的功能、质量和约束都应该知道，但这种事情显然是不可能做到的。ADD可以在一组在结构上很重要的要求是已知的情况下使用。ADD的输出是一组架构视图的草图，这些视图将共同标识架构元素及其关系或交互的集合。

ADD的步骤为：

1. 选择要设计的系统元素。
2. 确定所选元素的ASR。
3. 为所选元素生成设计解决方案。
4. 盘存剩余需求，并为下一次迭代选择输入。
5. 重复以上四步直到所有ASR都被满足。

## 架构的实现和测试

书p364。架构侵蚀：例如有时实现者采用了一些实现，使得架构模块化程度降低。

帮助保持代码和架构一致的四种技术：

1. Embedding the design in the code 将设计嵌入代码中
2. Frameworks 框架
3. Code templates 代码模板
4. Keeping code and architecture consistent 保持代码和架构一致

书p370。各种测试类别：

- 单元测试：测试系统一部分
- 整合测试：测试各个单元一起工作的情况
- 系统测试：直接测试整个系统
- 验收测试：看是否达到用户需求
- 黑盒测试：不依赖系统内部设计、结构或实现信息来测试
- 白盒测试：跟上面反过来
- 灰盒测试：使用部分有用的设计、结构或实现信息来测试
- 基于风险的测试。

架构师在测试中的角色：

1. 测试计划
2. 测试开发
3. 测试解释

## 4. 测试工具构建

# 架构评估

架构评估的三种形式及其特点

1. Evaluation by the designer within the design process 设计者在设计过程中的评估
2. Evaluation by peers within the design process 在设计过程中由同行评估
3. Analysis by outsiders once the architecture has been designed 设计架构后由外部人员进行分析

## The architecture tradeoff analysis method(ATAM)

ATAM的设计使得评估者不需要熟悉架构或其业务目标，系统不需要构建，并且可能有大量的涉众。

ATAM的参与者：

1. The evaluation team
2. Project decision makers
3. Architecture stakeholders

ATAM的步骤：

1. Present the ATAM 展示ATAM
2. Present business drivers 展示业务驱动力
3. Present the architecture 展示架构
4. Identify architectural approaches 确定架构方法
5. Generate utility tree 生成效用树
6. Analyze architectural approaches 分析架构方法
7. Brainstorm and prioritize scenarios 集思广益并确定方案的优先级
8. Analyze architectural approaches 分析架构方法
9. Present results 呈现结果

书p409, watchdog是什么，为什么重要：watchdog本质上是一个定时器，用于监视系统是否正常工作，可以通过硬件或软件的方式实现。在linux操作系统中，一旦程序启用了watchdog，就必须在固定时间内(比如每一分钟)向watchdog发送数据以重置定时器，否则watchdog将认为服务运行状态不正常，并执行重启等恢复操作。watchdog多用于嵌入式系统。

# 云服务中的架构

## 云的定义

- On-demand self-service 按需自助服务
- Ubiquitous network access 无所不在的网络访问
- Resource pooling 资源池
- Location independence 位置独立
- Rapid elasticity 快速弹性
- Measured service 实测服务
- Multi-tenancy 多租户

## 云服务模型

1. Software as a service 软件作为服务
2. Platform as a service 平台作为服务：为开发人员提供集成堆栈；PaaS管理对堆栈底层的分配。
3. Infrastructure as a service 基础架构作为服务：服务器以集群形式排列，一些服务器作为IaaS的基础结构，每个服务器都有一个管理程序作为其基础。

## 云服务的部署模型

私有云、公有云、社区云、混合云

## 云服务的主要机制

管理程序、虚拟机、文件系统、网络

## 云服务的主要技术

- Virtual memory page table 虚存页表
- Hypervisor manages 程序管理
- Virtualization 虚拟化
- HDFS Hadoop
- IaaS
- PaaS
- Databases 数据库：关系型数据库已经完全不适用，引入新的数据模型：键值对。

## Quality Attributes of Cloud Architectures 云服务架构的质量属性

### 1. Security

1. 多租户引起了对非云环境的额外关注：无意中的信息共享、虚拟机“逃离”、侧信道攻击、拒绝服务攻击
2. 当决定在云中托管什么应用程序时，组织需要考虑风险

### 2. Performance

1. 自动缩放在负载增长时提供额外的性能。新资源的响应时间可能不适合，架构师需要了解应用程序的资源需求

### 3. Availability

1. 故障是云系统的常见事件。云服务提供商必须在出现一些异常时确保云服务本身仍然可用。应用程序开发人员必须假设实例会失败，并在失败的情况下构建检测和纠正体制。

## 林博选的课后题(这学期和以前的)

### p77 第一题

问题：What's the relationship between a use case and a quality attribute scenario? If you wanted to add quality attribute information to a use case, how would you do it?

翻译：用例和质量属性场景之间有什么关系？如果要将质量属性信息添加到用例中，您将如何进行？

解答：用例和质量属性场景是同一件事情的不同表示方法。在软件工程中，用例描述用户想要实现的功能，而质量属性场景包含功能在设计的应用程序中如何运行的描述。功能和质量属性是正交的，如果要将质量属性信息添加到用例，那就添加新的用例。质量属性场景在质量属性需求规范中的作用与用例在功能需求规范中所扮演的角色相同。每个用例提供了一个或多个场景，该场景揭示了系统是如何同最终用户或其他系统交互的，从而获得一个明确的业务目标。用例要避免技术术语，取而代之的是最终用户或者领域。

### p78 第二题

问题：Do you suppose that the set of tactics for a quality attribute is finite or infinite? Why?

翻译：你认为质量属性的一套策略是有限的还是无限的？为什么呢？

解答：策略是无限的。策略的重点是单一质量属性响应。在一种策略中，必须有设计者明确考虑和控制权衡。通常可以选择多种策略来改善特定的质量属性。选择使用哪种策略取决于诸如其他质量属性之间的权衡和实施成本等因素。

#### p78 第三题

问题：Discuss the choice of programming language (an example of choice of technology) and its relation to architecture in general, and the design decisions in the other six categories? For instance, how can certain programming languages enable or inhibit the choice of particular coordination models?

翻译：讨论编程语言的选择（例如选择一门技术）及其与架构的大体关系，以及其他六个类别的设计决策？例如，某些编程语言如何允许或禁止选择特定的协作模型？

解答：一个好的软件架构能够满足系统的品质，能够支持计划编制过程，对系统开发的指导性，能够有效的管理复杂性，为复用奠定了基础。绝大多数架构或者编程语言的产生都是来源于项目。比如php语言属于开源语言，利于学习，运用普遍。只能用来开发网站，无法用来开发软件或其他的一些应用，局限性比拟大。普通运用php开发的企业网站，通常会运用网上现成的开源CMS来搭建程序。

#### p78 第四题

问题：We will be using the automatic teller machine as an example throughout the chapters on quality attributes. Enumerate the set of responsibilities that an automatic teller machine should support and propose an initial design to accommodate that set of responsibilities. Justify your proposal.

翻译：我们将在整个章节中使用ATM作为质量属性的示例。列举自动柜员机应支持的一系列功能，并提出初始设计以适应这组功能。证明你的提议是正确的。

解答：这个书上很多

#### p78 第六题

问题：Consider the choice between synchronous and asynchronous communication (a choice in the coordination mechanism category). What quality attribute requirements might lead you choose one over the other.

翻译：请考虑在同步和异步通信机制之间进行选择（在协调机制下选择）。是什么质量属性要求导致你选择这一个而不是另一个？

解答：同步通信效率更高，更加适合对速度要求高的传输，在有性能这一质量属性要求下会选择同步通信。异步通信是点对点的，对时序要求低，在要求易用性这一质量属性时会选择异步通信。同步是当用户发送一个请求，需要等待返回后才能发送下一个请求，用户有等待时间。异步是增加了消息队列，将用户请求放入队列中等待返回，用户无需等待。

#### p78 第七题

问题：Consider the choice between stateful and stateless communication (a choice in the coordination mechanism category). What quality attribute requirement might lead you to choose one over the other ?

翻译：请考虑在有状态和无状态通信机制之间进行选择（在协调机制下选择）。是什么质量属性要求导致你选择某一个。

解答：有状态通信无需额外的调用，具有低延迟的优点，在性能和可用性的质量属性要求下选择；无状态通信有防止数据丢失的优点，在安全性的质量属性要求下选择。

#### p78 第八题

问题：Most peer-to-peer architecture employs late binding of the topology. What quality attributes does this promote or inhibit ?

翻译：大多数p2p架构使用后期绑定的拓扑，什么质量属性促进还是抑制？

解答：抑制了安全性、可用性。促进了互操作性和性能。

#### p100 第一题

问题：Write a set of concrete scenarios for availability using each of the possible responses in the general scenario.

翻译：可用性的通用场景下有多个可能的响应，为每个响应写一个通用场景。

解答：下面有

#### p100 第三题

问题：Write a concrete availability scenario for a program like Microsoft Word.

翻译：为类似于Microsoft Word的程序编写具体的可用性场景。

解答：

A. Log the fault：把错误记录到日志里。记录使用word时可能产生的错误信息（无法打开文档的问题，格式转换的问题，图片显示问题，单词拼写问题，语法问题等）。

B. Notify appropriate entities (people or systems)：通知恰当的实体（人或系统）。如无法打开文档的问题会通知给开发者，单词拼写问题、语法问题会显示在屏幕上通知给使用者。

C. Disable source of events causing the fault：使导致错误的事件源失效。如单词拼写问题会用红线标注给与警示。

D. Be temporarily unavailable while repair is being effected：修复时暂停服务。如无法打开文档则会提醒错误并停止尝试打开。

E. Fix or mask the fault/failure or contain the damage it causes：修复或屏蔽错误，或者控制损失。如无法转换格式则会自动保存原格式。

F. Operate in a degraded mode while repair is being effected：修复时在降级模式下操作。如无法显示图片则会预留图片空间，待使用者解决。

#### p100 第四题

问题：Redundancy is often cited as a key strategy for achieving high availability. Look at the tactics presented in this chapter and decide how many of them exploit some form of redundancy and how many do not.

翻译：冗余通常被认为是实现高可用性的关键策略。看看本章介绍的策略，并决定其中有多少采用某种形式的冗余，有多少则没有。

解答：

在Detect Faults (Ping/Echo, Monitor, Heartbeat, Timestamp, Sanity Checking, Condition Monitoring, Voting, Exception Detection, Self-Test) 和Prevent Faults (Removal from Service, Transactions, Predictive Model, Exception Prevention, Increase Competence Set)的过程中没有采用冗余。

在Recover from Faults的过程中：

Active redundancy: 采取主动冗余

Passive redundancy: 采取被动冗余

Spare: 采取被动冗余

Exception Handing: 没有采用冗余

Rollback: 采取主动冗余和被动冗余结合

Software Upgrade: 采取主动冗余或者被动冗余二者都可

Retry: 没有采用冗余

Ignore Faulty Behavior: 没有采用冗余

Degradation: 没有采用冗余

Reconfiguration: 没有采用冗余

Shadow: 没有采用冗余

State Resynchronization: 没有采用冗余,但是工作的时候和主动冗余以及被动冗余合作

Escalating Restart: 采用了被动冗余(Level 0)

Non-Stop Forwarding (NSF): 采取主动冗余

p100 第五题

问题：How does availability trade off against modifiability ? How would you make a change to a system that is required to have "24/7" availability (no scheduled or unscheduled downtime , ever)?

翻译：可用性与可修改性如何进行权衡？当需要具有“24/7”可用性时应该如何更改系统（没有计划或非计划的停机时间）？

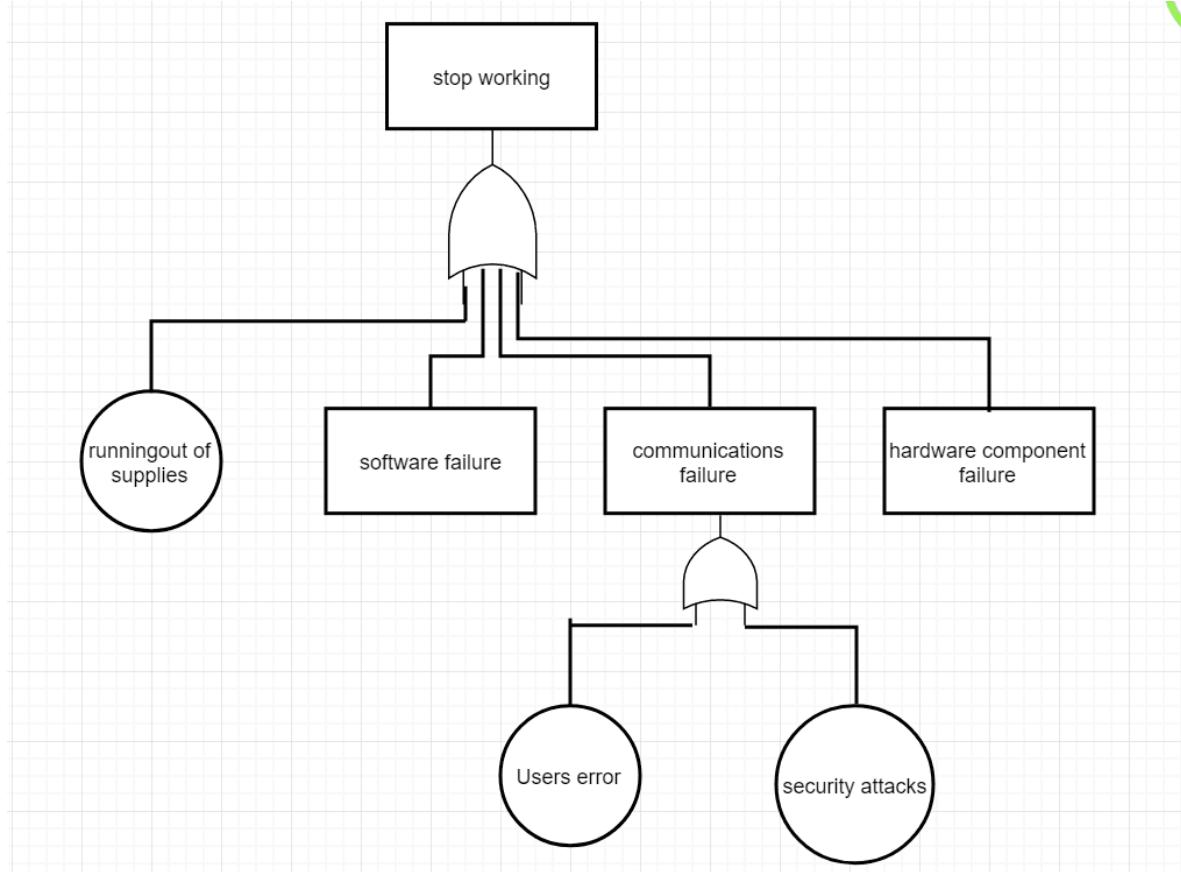
解答：可用性指标是架构设计的重要指标，对外是服务承诺，对内是考核指标。将系统分层、分割的主要目的是增加系统可维护性，以及对后期发展提供更好的功能扩展能力、并发处理能力。在进行权衡时需要根据具体的使用场景以及需求来决定，比如在项目会有多次进行迭代的情况，而产品在迭代的过程中本身会暂时影响了正常使用，初期阶段重视可修改性会对项目进程有很好的帮助。而当项目趋向于稳定时更应该重视可用性以便提供更好的客户体验。对于需要 “24/7”可用的情况可以采用镜像的方式来确保业务的连续性。

p101 第六题

问题：Create a fault tree for an automatic teller machine. Include faults dealing with hardware component failure, communications failure, software failure, running out of supplies, user errors, and security attacks. How would you modify your automatic teller machine design to accommodate these faults?

翻译：为自动出纳机构造故障树。包括处理硬件组件故障，通信故障，软件故障，资源贫乏，用户错误和安全攻击等故障。您如何修改自动出纳机设计以适应这些故障？

解答：



在自动出纳机程序编写的时候增加相应的查错机制，同时删除一些长久不需要存储的、无关紧要的数据，对自动出纳机进行定期维修，检查软硬件使用情况。

p101 第七题

问题：Consider the fault detection tactics(ping/echo, heartbeat , system monitor, voting, and exception detection). What are the performance implications of using these tactics?

翻译：考虑故障检测策略（响应机制，心跳机制，系统监控，投票和异常检测）。使用这些策略有什么性能影响？

解答：Ping/echo（响应）：通过系统监视器发出Ping请求来探测服务的通信路径是不是通的，某个组件是不是还在工作。

监视(Monitor)：通过使用一个监视器的组件来检测其他组件的健康状况。

心跳(Heartbeat)：通过组建定区发出一个心跳信息，来主动的告诉系统它正在工作。

投票(Voting)：使用冗余的组件做同一件事情，以相同的输入，如果产生不同的输出，则忽略少数，采纳多数的结果。

性能：性能是指软件系统及时提供相应服务的能力。包括速度、吞吐量和持续高速性三方面的要求。

以上的可用性策略都需发出请求并通过返回的信息判断组件的可用性，在一定程度上会增加系统负载，提高系统响应时间，但也提高了服务可用时间段，例如，一年内90%可用性的系统最多有36.5天的停机时间，使用可用性策略，可使系统可用性达到95%或以上。使用可用性策略或多或少都会影响系统的响应速度，即使它可以增加服务时间，你需要决定性能或可用性在当前环境下哪个更关键一点。

p116 第一题

问题：Find a web service mashup. Write several concrete interoperability scenarios for this system.

翻译：找一个web服务的复合应用程序实例，并写出几个这个系统的具体的互操作场景。

解答：比如登录微博，登录微信等等。

### p116 第二题

问题：What is the relationship between interoperability and the other quality attributes highlighted in this book? For example, if two systems fail to exchange information properly, could a security flaw result? What other quality attributes seem strongly related (at least potentially) to interoperability?

翻译：互操作性和这本书里面强调的其他的质量属性的关系怎么样？例如，如果两个系统不能正常地交换信息，可能会导致一个安全瑕疵（隐患）吗？哪些别的质量属性看上去和互操作性有紧密的关系（至少有可能是）？

解答：互操作性指的是系统内或者系统之间不同的组件可以有效地进行信息交换，通常是以服务（Service）的形式来进行的。互操作性的关键因素包括通信协议，接口定义，数据格式的定义等等，而标准化是实现互操作性的重要手段。互操作性与其他质量属性关系如下：

- 可用性Availability：互操作性对可用性具有良好的保证，因为对于可用性来说，用户对于操作可能会出现各种的问题，而互操作性会在一些服务场景上降低错误的发生，在很大的程度上增加了补救措施的种类。
- 可修改性Modifiability：在使用具有互操作性强度高的系统时，系统的可修改性会提高很多。在进行本系统的更新时，系统自身的可修改性会因互操作性的提高而降低开发成本。
- 性能Performance：由于互操作性的介入，系统的性能可能会随着功能的完善，导致性能的提高。由于互操作性的介入会使很多板块的灵活性增加，届时使得整个系统更加具备高内聚、低耦合的特征，同时使得整个系统可以更加有效的得到优化，提高运行效率。
- 安全性Security：安全性可能会降低，比如一个论坛如果安全性较低，那么和他具有良好互操作性的一些外部系统的安全性将会受到极大的不确定性保证。
- 可测试性Testability：互操作性较高的情况下，可测试性将会降低。介于安全性的考虑因素，某些外部系统对于测试会有高度敏感性。因此具有良好互操作性的系统在可测试性上会具有相对不好的表现。
- 易用性Usability：在一定应用场景下，高度互操作性的软件会使得软件的易用性提高。具备不同的系统之间的交互，在一定情况下会降低用户的学习成本和使用成本，因为用户可以借由其他系统的信息进行合适的了解性操作。例如我们使用QQ登录一些论坛时，就降低了我们的注册成本或者是输入账号密码的登录成本。

如果两个系统不能正常的交换信息，可能会是一个安全隐患导致的。安全性，可用性，可修改性，看起来和互操作性关系很紧密。

### p116 第三题

问题：Is a service-oriented system a system of system? If so, describe a service-oriented system that is directed, one that is acknowledged, one that is collaborative, and one that is virtual.

翻译：面向服务的系统是否为一个由很多系统组成的更大系统？

解答：当一个面向服务的系统是由许多各自提供服务的“小型”系统组成，以提供具有一致目的完整服务的更庞大系统时，可以认为该面向服务系统是一个“系统的系统”（System of System，缩写为 SoS）。这里的“小型”系统是针对这个面向服务的“大型”系统来说的，并不是指系统的规模很小。教材中关于SoS的定义为：If you have a group of systems that are interoperating to achieve a joint purpose, you have what is called a system of system (SoS). An SoS is an arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities. 翻译一下就是：SoS是由一组集成的系统组成，以实现一个联合目的。SoS是一个对各个系统的管理和组织，使得各个独立的系统能够被集成到一个更大的系统，来实现特定的功能。按照组织和管理性质，SoS系统可以划分为Directed, Acknowledged, Collaborative, Virtual四种。

- Directed（直接性）：这些“小型”系统具有组成一个SoS的一致目的，有着中心化管理、经费运作和权力管理的SoS组织。这些“小型”系统是SoS的下属。
- Acknowledge（公认性）：这些各自独立的系统和SoS属于同一层次的级别，各自发展、管理、运营，同时受到同个级别的SoS的中心化管理、协同。
- Collaborative（协作性）：这些独立的系统没有受到一致的组织和中心化管理，各自独立存在。系统们自愿一起协作，以达到共同的利益。
- Virtual（虚拟性）：和Collaborative类似，但是这些独立的系统之间甚至不清楚与自己协作的系统的身份。

#### p116 第四题

问题：Universal Description, Disco, and Integration(UDDI) was touted as a discovery service, but commercial support for UDDI is being withdrawn. Why do you suppose this is? Does it have anything to do with the quality attributes delivered or not delivered by UDDI solutions?

翻译：统一描述、发现和集成（UDDI）被吹捧为发现服务，但UDDI的商业支持正在被撤销。你认为这是为什么？是否与UDDI解决方案提供或未提供的质量属性有关？

解答：UDDI的衰落是必然的。

UDDI在提出时听起来是个好主意，他是一种基于XML的跨平台的描述规范，可以使世界范围内的企业在互联网上发布自己所提供的服务。具有如下好处：

- 强移植性，任何规模的行业或企业都能得益于 UDDI。
- 高性能，在 UDDI 之前，还不存在一种 Internet 标准，可以供企业为它们的企业和伙伴提供有关其产品和服务的信息。也不存在一种方法，来集成到彼此的系统和进程中。
- 强扩展性，定义一旦首选的企业被发现后如何启动商业
- 扩展新客户并增加对目前客户的访问
- 满足用户驱动的需要，为在全球 Internet 经济中快速合作的促进来清除障碍

UDDI的出现为web service提供了良好的理论基础和技术抽象，但是在实际使用中，它缺失了某些重要的软件质量属性：

- 过于复杂，低易用性。大型组织应该拥有其Web服务的目录，但这可以像维基页面一样简单。
- 忽略安全性。如果UDDI代理为用户动态选择服务提供商，用户就没有机会对服务的安全性进行任何尽职调查。
- 管理和收集小额支付的困难仍然存在。
- Web服务通常用于防火墙后面，用于SOA，将应用程序与业务伙伴集成，以及调用众所周知的 API。对于这些目的，UDDI是完全多余的。

#### p116 第五题

问题：Why has the importance of orchestration grown in recent years?

翻译：为什么最近这些年来协调的重要性一直在增长？

解答：这些年来orchestrate策略被人们越来越重视的原因，我觉得应该是和基于部件的技术的发展有关。软件部件是指在软件系统设计中能够重复使用的建筑模块，它包装了一系列互相关联的操作和服务。当各个部件之间需要用一种非常复杂的形式来完成一个很复杂的任务时，协调这项策略便会被使用，它负责描绘这些交互的过程的蓝图。这有点像mediator,这个设计模式。按我的理解的话，就像是虚拟化出一份现实中不存在的部件，它充当媒介，对外，与其他部件的接口进行交互，在内部则实现复杂的逻辑功能。这个与高内聚低耦合的软件设计模式也是十分吻合的。所以Orchestrate这项策略的重要性也就不言而喻的，善用Orchestrate能够使我们的系统更加的简洁，可复用性及可操作性提高，这也是软件工程这些年的目标吧，所以说，这些便是Orchestrate这项策略这些年来越来越受重视的原因。

#### p116 第六题

问题：If you are a technology producer, what are the advantages and disadvantages of adhering to interoperability standards? Why would a producer not adhere to a standard?

翻译：如果你是一个技术生产者，坚持互操作性准则的优缺点有哪些？为什么生产者不会遵守一个准则？

解答：优点：提供已有功能给更多应用，获取其他功能，更好的可用性，可修改性。缺点：降低产品竞争力，快速发展的技术打破操作闭环，安全性降低。理由：产品独立性，保持闭环，避免竞争，难以平衡多个准则。

### p116 第七题

问题：With what other systems will an automatic teller machine need to interoperate? How would you change your automatic teller system design to accomodate these other systems?

翻译：自动柜员机需要和其他哪些系统进行交互操作？你会如何改进你的自动柜员机系统的设计来使它和其他系统和适应？

解答：（1）自动柜员机需要用到REST技术，对数据进行CURD操作，所以我认为它需要和本地或者远程数据库系统进行交互，这就要求能正确、准时、安全的完成相应的操作；（2）它是一种终端设备，所以要和人进行交互（如果把人看作一个系统的话），这要求它能提供友好的见面，傻瓜式操作方法，简单易用；（3）它可能与电子货币（虚拟货币、银行卡、二维码、指纹、人脸等）有关，所以要和支付系统、安全保障系统进行交互（这里的支付系统、安全系统可能与之前的数据库系统有交集，但不完全相同），这就需要它可靠、稳定，出错率极低，出故障后恢复快等等。综上，自动柜员机需要和数据库系统、支付系统、安全保障系统以及人进行交互。

改进措施：1. 准确而快速的连接数据库。可以提高数据存储设备的性能、容量，优化存储方式，及时清理垃圾数据，减少冗余，优化查询机制等等。这同时考虑到了互操作性质量属性中Discovery、Handing of the response两个重要的方面；2. 简化操作，美化界面，以便用户能快速流畅的使用这个自动柜员机。要做好这方面，必须理解用户需求，做好调研、市场走向，做好充分的测试，将bug最小化。这主要考虑到了互操作性的Discovery方面；3. 力求搭建一条快速又安全的绿色支付通道。要充分与第三方（如果存在）进行沟通，与相关系统有一个完美的衔接过程，使得每一笔交易都能走好、走准属于它们的那一条路，并完善故障机制处理，增强系统的健壮性。这主要考虑到了互操作性的响应处理机制方面。

### p128 第一题

问题：Modifiability comes in many flavors and is known by many names. Find one of the IEEE or ISO standards dealing with quality attributes and compile a list of quality attributes that refer to some form of modifiability. Discuss the differences.

翻译：可修改性有多种风格，并且有许多名称。找到处理质量属性的IEEE或ISO标准之一，并编制一系列质量属性，这些属性涉及某种形式的可修改性。并讨论它们的差异。

解答：涉及可修改性的质量属性：可用性(Availability), 可测试性(Testability), 性能效率(Performance efficiency), 安全性(Security)

可用性：系统在考察时间内是否能够正常运行。

可测试性：发现软件系统错误，并且设计测试、执行测试的能力。

性能：一般指的是系统对于用户操作的响应时间的快慢。

安全性：系统保护数据和信息不被未授权访问所获取，而且提供数据信息给已授权的访问操作。

### p128 第二题

问题：For each quality attribute that you discovered as a result of the previous question, write a modifiability scenario that expresses it.

翻译：对于您在上一个问题中发现的每个质量属性，请编写一个表达它的可修改场景。

解答：

Availability:

Source: Heartbeat Monitor

Stimulus: Server Unresponsive

Environment: Normal Operation

Response: Inform Operator Continue to Operate

Response Measure: No Downtime

Testability:

Source: Unit Tester

Stimulus: Code Unit Completed

Environment: Development

Response: Results Captured

Response Measure: 85% path Coverage in Three Hours

Performance:

Source: Users

Stimulus: Initiate Transactions

Environment: Normal Operation

Response: Transactions Are Processed

Response Measure: Average Latency of Two Seconds

Security:

Source: Disgruntled Employee from Remote Location

Stimulus: Attempts to Modify Pay Rate

Environment: Normal Operations

Response: System Maintains Audit Trail

Response Measure: Correct Data is Restored within a Day and Source of Tampering Identified.

p129 第三题

问题：In a certain metropolitan subway system, the ticket machine accept cash but do not give change. These is a separate machine that dispenses change but does not sell tickets. In an average station there are six or eight ticket machines for every change machine. What modifiability tactics do you see at work in this arrangement? What can you say about availability?

翻译：在某大都会地铁系统中，售票机接受现金但不会找零钱。有一台独立的机器，它提供零钱但不卖票。平均每个车站有六台或八台售票机搭配一台换置零钱的机器。在这种安排下，你在工作中看到什么样的可适应性策略？关于可用性你能说些什么？

解答：策略有二：

1. reduce the size of module.
2. increase cohesion

Availability（可用性）是指能够满足用户的需求，在这里面由于模块化做的很好，这大大降低了系统出现问题的可能性，找零系统跟售票系统不属于同一个模块，当其他模块出问题的时候剩余模块还可以继续工作，这个系统属于一个High Availability的系统。

p129 第四题

问题：For the subway system in the previous question, describe the specific form of modifiability (using a modifiability scenario) that seems to be the aim of arranging the ticket and change machines as described.

翻译：对于前面问题中的地铁系统，描述可修改性的具体形式（使用可修改性场景），该可修改性似乎是按照描述的方式安排票和兑换机的目的。

解答：在这里的可修改性具体体现形式在于分割（split module）模块和增加内聚（increase cohesion）。在此系统中，有六到八台售票机器搭配一台找零机器，这两种机器的要完成的功能不一样，售票机只售票，找零机只找零，根据他们不同的职责就可以将他们划分为两个模块，在划分模块的过程中既实现了分割模块的目的又实现了增加内聚的目的

p129 第五题

问题：A wrapper is a common aid to modifiability. A wrapper for a component is the only element allowed to use that component; every other piece of software uses the component's services by going through the wrapper. The wrapper transforms the data or control information from the component it wraps. For example, a component may expect input using English measures but find itself in a system in which all of the other components produce metric measures. A wrapper could be employed to translate. What modifiability tactics does a wrapper embody?

翻译：包装是可修改性的常见辅助。组件的包装器是允许使用该组件的唯一元素；每个其他软件都通过包装器使用组件的服务。包装器将数据或控制信息转换为它包装的组件。例如，组件可以使用英语测量的专家输入，但发现自己处于所有其他组件产生度量测量的系统中。可以使用包装器进行翻译。包装器体现了什么可修改性策略？

解答：Wrapper embody 体现了cohesion 和 coupling 两个策略。首先原本系统中的每个组件都是相互独立的个体，因为包装器是唯一可以使用组件的元素，所以通过包装器将各个部件连接在了一起，这里体现了cohesion策略。其次，组件期望用英语度量作为输入，这里包装器就可以充当英语翻译的中介，让所有组件的输出都由包装器进行翻译，这样子，就不用每个组件自己进行翻译，减少了组件的coupling.

p129 第六题

问题：Once an intermediary has been introduced into an architecture, some modules may attempt to circumvent it, either inadvertently (because they are not aware of the intermediary) or intentionally (for performance, for convenience, or out of habit). Discuss some architectural means to prevent inadvertent circumvention of an intermediary.

翻译：一旦将中介引入到体系结构中，某些模块可能会无意中（因为他们不了解中间人）或故意（为了方便，为了方便或出于习惯）而试图绕过它。讨论一些建筑手段，以防止无意中规避中间人。

解答：

1. increase cohesion and reduce coupling ,通过这样使各个模块成为耦合度较低的各部分，然后就要形成整体运作的时候就必须通过中介的作用将他们cohesion在一起。

2. split module, encapsulate, abstract common services等方法都可以。

#### p129 第七题

问题：In some project, deployability is an important quality attribute that measures how easy it is to get a new version of the system into the hands of its users. This might mean a trip to your auto dealer or transmitting updates it arrives. In project that measures deployability separately, should the cost of a modification stop when the new version is ready to ship? justify your answer.

翻译：在某些项目中，可部署性是一个重要的质量属性，用于衡量将新版本的系统交到用户手中的难易程度。这可能意味着您的汽车经销商旅行或传输它到达的更新。在独立测量可部署性的项目中，如果新版本准备好发布，用来修改的开销是否应该停止？

解答：是，在一个独立测量软件系统的可部署性系统，软件的基本功能能够完成即可，对于用户获取软件的便利程度才是最重要。在新版本发布之际，此时的修改会带来极大的不确定性，所以此时应尽量减少修改的开销，尽量不修改。

#### p129 第八题

问题：The abstract common services tactic is intended to reduce coupling, but it also might reduce cohesion. Discuss.

翻译：抽象公共服务策略旨在减少耦合度，但也可能降低内聚度，请讨论。

解答：软件设计中通常用耦合度和内聚度作为衡量模块独立程度的标准。划分模块的一个准则就是高内聚低耦合。耦合度从以下几个方面衡量：

- 一个模块对另一个模块的调用；
- 一个模块向另一个模块传递的数据量；
- 一个模块施加到另一个模块的控制的多少；
- 模块之间接口的复杂程度。

抽象公共服务是将两个模块中类似功能进行抽象统一接口，通过不同参数类型，提供不同服务。通过这种策略，代码量直接减少，模块间共用代码，直接降低耦合度。但是这也意味着多了一层抽象，致于类之间的关系变得模糊，引来了内聚度降低的问题。

#### p158 第一题

问题：Write a set of concrete scenarios for security for an automatic teller machine. How would you modify your design for the automatic teller machine to satisfy these scenarios?

翻译：请描述一组ATM（自动柜员机）的具体安全场景，并回答你将如何修改ATM的设计以满足这些安全需求？

解答：下面有

## 林博给的题(czq版)

2. Write a set of concrete scenarios for availability using each of the possible response in the general scenario.
- Artifact:** Processes
- Response:** Lay the fault
- Environment:** Notify appropriate activities
- Source:** People
- Stimulus:** Crash caused by invalid operation
- Response Measure:** Measure MTBF & MTTR
- Artifact:** Communication channels
- Response:** Detail some of events file or make failure.
- Environment:** Normal operation
- Source:** Communication channels break down
- Stimulus:** Try to get super admin privileges
- Response:** REST has a simple system and only based on XML information.
- Difference:**
1. SOAP has a complex system supporting CRUD operations.
  2. SOAP offers completeness. REST offers simplicity.
  3. REST is self-descriptive and it's a stateless protocol. SOAP is not and it depends on XML files.
  4. SOAP can be implemented on top of any communication protocol.
- Same:**
1. Both protocols are using HTTP.
  2. Both are protocol for resources access.

How to choose:

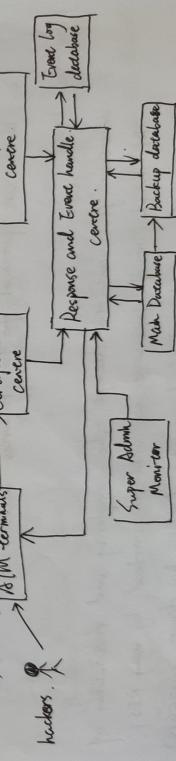
1. Whether you want to accept the complexity and restrictions of SOAP + WSDL or if you want to avoid the overhead by using REST.
2. Consider the size of individual messages (REST has fewer characters)
3. Using REST when performance is needed and SOAP when structured info.

SHOTON REDMI K20  
AI TRIPLE CAMERA

File 2 - 346 Page No. 201730681303

1. Write a set of concrete scenarios for security for an ATM. How would you modify your design for the automatic teller machine to satisfy these scenarios?
- Artifact:** Data projected by the system
- Response:** Define unauthorized attempt. Revoke audit trail.
- Environment:** Online system, fully operational.
- Source:** Unknown hacker or other attacker
- Stimulus:** Unauthorized attempt to get other user's data
- Response Measure:** Response: Correct data is restore within a day. User Sender is given right hacker's attempt and cut off connection between ATM and hacker.
- Artifact:** Data processed by the system
- Response:** Detect hacker's attempt and cut off connection between ATM and hacker.
- Environment:** Online system, fully operational.
- Source:** Unknown hacker or other attacker
- Stimulus:** Try to change data of other users' accounts
- Response Measure:** Response: Correct data is restore within a day. User Sender is given right to other servers daily.
- Artifact:** Super admin privileges.
- Response:** Out of connection between ATM and hacker.
- Environment:** Online system, fully operational.
- Source:** Unknown hacker

- Response Measure:** Response: Check system log and undo illegal operations.
- Artifact:** Certificate
- Response:** Create log of certificate
- Environment:** Online system, fully operational.
- Source:** Unknown hacker



For scenario as mentioned above, we have redesign ATM's architecture as follow:

SHOTON REDMI K20  
AI TRIPLE CAMERA

C/S 架构的优缺点 (优点和缺点) 为：  
 1、具有更好的可伸缩性，系统中每个服务器都可以通过一个全局统一的  
 要求，易于对系统进行扩充和缩小。  
 2、具有可交互性，系统功能部分隔离，客户端机与服务器集中于数据的呈现。  
 和分析，而数据存储则集中于数据库管理。

6. 对于题的架构模式有何建议？可用何种技术来弥补？  
 由于外部单向调用服务，故该架构模式可阅读性，可修改性较差。  
 建议可阅读性技术：1. 给予服务增加一个可操作的规则模块，用于监控和修改状态。  
 2. 降低 WS, WS provider, WS Client 的设计复杂性。  
 增强可修改性技术：1. 分割可分割的模块，尽量做到高内聚，低耦合。  
 2. 限制系统内模块之间的依赖。  
 3. 对于重要的服务，独立出来形成一个公共的服务。  
 主要使用

4. Consider quality of service, WS (SOAP) implementation has greater support for security, availability and type of functionality. REST implementation is more appropriate for read-only functionality, typical of meadow.

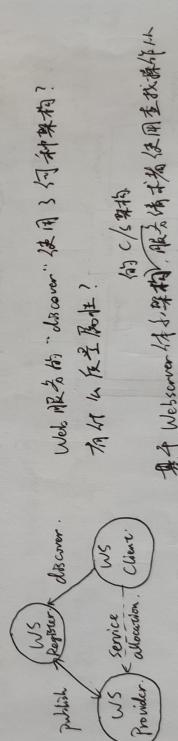
4. Which web architecture does P2P architecture inhibited / enable?

The P2P architecture enables:

1. Availability: P2P architecture have specialized peer nodes that having indexing or memory capabilities and allow a regular peer's search to reach a larger number of peers.
2. Performance: P2P architecture allows users share their resources and use network bandwidth fully, which brings good experience to users.
3. Usability: Usability: Once a resource has been save in a computer or server, other users in the network nearby can get it easily.

The P2P architecture inhibited:

1. Security: Any users can modify local files, which may lead a serious problem; ~~but~~ hackers may spread virus by this service.



为什么 Web Service “discover” 使用了何种类机构？

有什么具体 Role？

何为 C/S 架构

何为 Web Server / 客户端

服务请求者

使用至操作层

层

层

层

层

何为 Web Service 及端点 S 之交互。

SHOT ON REDMI K20

AI TRIPLE CAMERA

OO

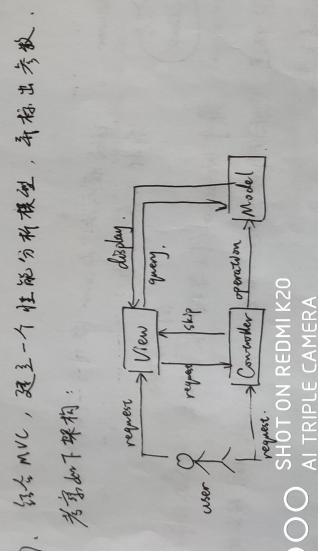
OO

OO

OO

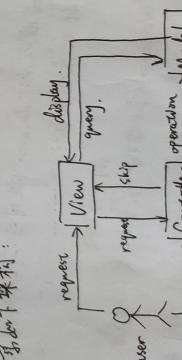
OO

OO



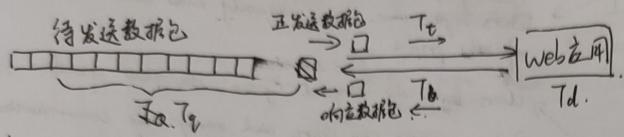
7. 编写 MVC，建立一个性能分析模型，并指出考核。

考察如下架构：



SHOT ON REDMI K20  
AI TRIPLE CAMERA  
OO  
OO  
OO

上面是一个典型的MVC架构，我们不妨构造一个用户请求队列，通过计算用户请求发出与接收响应的时间差来对MVC架构进行评估。



定义如下概念：

$T_q$ : 请求等待时间

$T_t$ : 请求发送时间

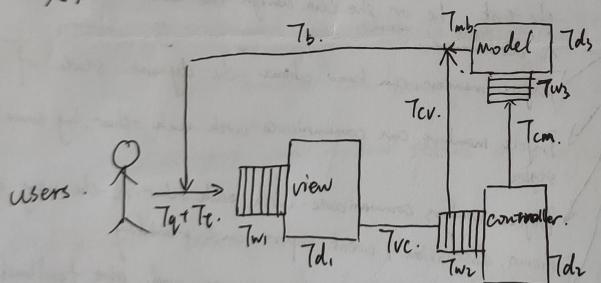
$T_d$ : 请求处理时间

$T_b$ : 响应发送时间

故对于每个请求，总时间为代价为：

$$T = T_q + T_t + T_d + T_b.$$

考虑MVC架构，我们把请求处理过程精细化。



其中， $T_{vc}$  是从 view 层传输到 controller 所需时间， $T_{cm}$  是从 controller 层到 model 所需时间。

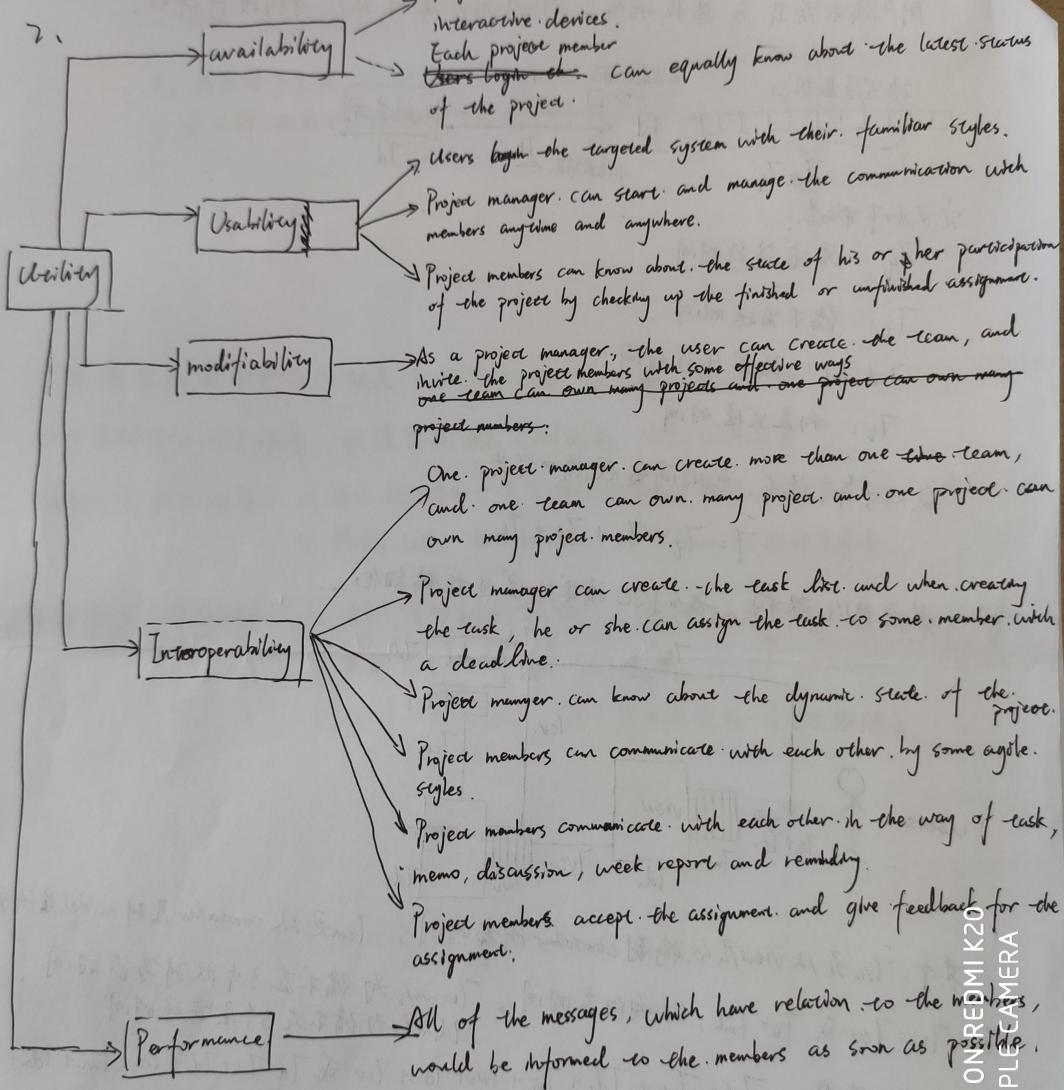
$T_b = T_{vc}$  或  $T_b = T_{mb}$ ，为响应时间。 $T_{w1-w3}$  为请求在 3 个队列等待时间。  
 $T_{d1-d3}$  为请求在 3 个层处理时间。

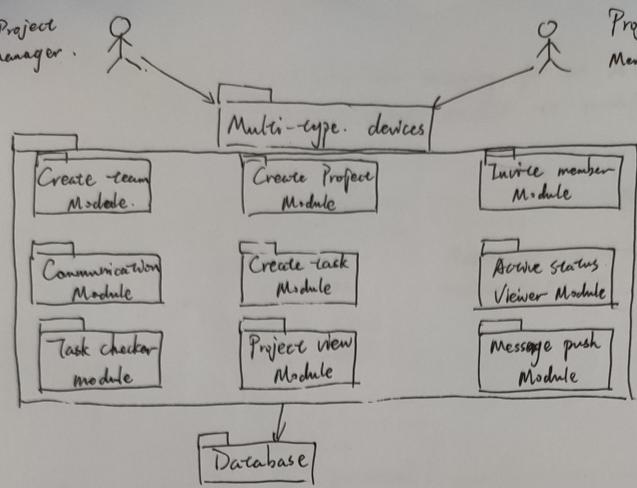
以上过程， $T_d = T_{w1} + T_{d1} + T_{vc} + T_{w2} + T_{d2} + T_{cv}$  或  $T_d = T_{w1} + T_{d1} + T_{vc} + T_{w2} + T_{d2}$

## 7X7 题

1. 问

2.





4.

A completed software design specification meet the following criteria:

1. It should be able to provide service quickly and be quick.
2. Project manager can manage teams, projects and members on it.
3. Everyone can follow status of projects and other peoples work status.

Architectural Strategies:

1. Make good design to keep high cohesion and low coupling.
2. Divide Team groups and projects to make them easy to follow status and modify.

Policies and Tactics:

1. Make monitors to follow project and team members' status.
2. Locate watching and modifying service and get location of project and people.
3. Split module as small as possible and increase semantic coherence.
4. Limit event rate and reduce overhead to increase performance
5. Support user initiative like cancel and undo.

SHOT ON REDMI K20  
AI TRIPLE CAMERA

Date: 12/03/2019

Author: JHSeng