公有云平台的例子（英文书503）  
云计算的特点：   
on-demand self service按需自助服务  
ubiquitous network access 无处不在的网络接入  
resource pooling 资源池  
location independence 位置独立  
rapid elasticity快速弹性

云服务模型：  
SaaS  
PaaS  
IaaS  
  
部署模型：  
private cloud私有云  
public cloud公共云  
community cloud社区云  
hybrid cloud混合云

主要技术：511  
hypervisor 管理程序  
page mapper 页映射  
scheduler 调度器  
storage 储存如HDFS 513  
network 网络  
  
例子技术：  
Iaas：  
Virtual resource manager虚拟资源管理器  
persistent object manager持久对象管理器  
cluster manager群集管理器  
filesystem manager文件系统管理器  
node manager节点管理器

PaaS:  
GAE  
Azure  
  
Database:  
typically nosql:  
HBase  
MongoDB  
  
nosql:  
没有schema模式  
没有事务  
最终一致（eventually consistent）  
没有标准化normalization  
  
三点质量属性和其他软件系统有出入：  
安全：  
不经意间的信息共享  
。。  
。。  
拒绝服务  
  
性能:  
有弹性  
  
可用性：  
一直可用  
  
举例P2P的模式（英文书220）  
通过合作达到计算的目的  
  
元素：  
peer，又分普通peer，特别的peer。  
特殊peer提供路由，寻址等功能，能够通过它找到其他peer  
  
请求/响应连接器  
  
限制：  
找到别的peer的hop的次数  
一个peer可以找到的资源  
哪个peer知道其他peer的存在  
  
弱点：  
安全性、数据一致性、可用性、备份和恢复都很复杂  
小的p2p可能达不到性能和可用性目标  
  
例子：  
file-sharing 文件共享  
instant-messaging 即时通讯  
instant-grid-computing 网格计算  
routing 路由  
wireless ad hoc network无线自组织网络  
  
file-sharing有BitTorrent文件下载，还有Skype的VoIP通讯工具  
  
代理模式，黑板系统举个例子（英文书211）  
代理模式例子：  
一般的分布式服务都会用到代理模式，例如EJB和.NET  
  
黑板系统例子：  
黑板模式是一种常用的架构模式，应用中的多种不同数据处理逻辑相互影响和协同来完成数据分析处理。就好像多位不同的专家在同一黑板上交流思想，每个专家都可以获得别的专家写在黑板上的信息，同时也可以用自己的分析去更新黑板上的信息，从而影响其它专家。  
  
在实际应用中常见的实现模式有：  
A 利用数据库  
利用数据库充当黑板，不同的应用共享数据库中信息，并且可以更新数据信息。这也是最常见的实现方式。  
特点：  
1 便于实现信息的查询，筛选和统计，这方面关系数据库提供了SQL 92的强大支持。  
2 不能用于较高实时性要求的环境，这种实现是工作在“拉模式”下的，并且高频率的访问数据库会导致严重的系统性能问题。  
B 利用发布—订阅模式  
这种实现方式通常采用消息队列作为黑板，队列工作在主题模式（Topic），专家作为队列的订阅者，同时可以向队列发送消息，消息会被发送至所有订阅者。以上过程实现了专家间的信息交流。  
特点：  
1 可以有效应用于实时性要求较高的系统，这种实现工作在“推模式”下。  
2 难于实现信息的统计分析，不像实现方式一那样可以通过SQL支持，这些工作必须开发者自己完成。﻿

架构决策点有哪些，架构决策分类？第7条技术选择是最难的。如何紧跟技术潮流？（英文77页）必考  
(第4章,质量设计的7种决策)  
Allocation of responsibilities:责任分配  
Coordination model:协同模型  
data model:数据模型  
management of resource:资源管理  
mapping among architectural elements:元素映射  
binding time decisions:绑定时间决定  
choice of technology:技术选择  
  
如果技术是别人定的，就会对上面7条造成限制；如果不是，决策前先看看  
能不能实现其他决策  
工具足不足够开发  
外部支持够不够（教程，课堂等）  
对其他决策有什么副作用  
和现在的系统兼不兼容

6、SOA是属于三种架构模式中的哪一个？（？）component-and-connector pattern  
  
7、架构模式：211页，ERC，元素，关系，约束。broke元素，设计模式（211）  
broker模式属于component-and-connector模式  
用一个叫做broker的运行时组件作为客户端和服务器通讯的中介  
元素：  
Client  
Server  
Broker  
Client-side proxy  
Server-side proxy  
  
限制：  
一个客户端只能跟一个broker相连  
一个服务器只能跟一个broker相连  
  
弱点：  
增加延迟  
broker可能成为通讯的瓶颈  
可能导致single point failure  
增加复杂程度  
可能成为安全攻击的目标  
broker难以测试

SOA   
关系：可使用的不同种类的组件连接到各自的连接器

约束：服务消费者与服务供应商连接。

弱点：

基于SOA的系统通常是复杂的建立。

SOA有一个与中间件相关的性能开销，服务可能是性能瓶颈，通常不提供性能保证。

架构危险信号／现象（phenomena）：

（1）没有多个候选的设计方案

（2）构架依赖于某个版本的商业产品

（3）产生数据模块和使用数据的模块没分离

商业质量属性：上市时间、成本和收益、系统生命期的长短、目标市场、推出计划、与老系统集成

与构架本身相关的质量属性：可构建性、概念完整性（conceptual integrity）

质量属性场景（Quality Attribute Scenarios）是一种面向特定的质量属性的需求。（课本p66）

包括：刺激源（Source of stimulus），刺激（stimulus），环境（Environment ），制品（Artifact），响应（Response）和响应度量（Response measure）。

涉众STAKEHOLDERS概念：对构建软件系统感兴趣的人。包括：客户，最终用户，开发人员，项目经理，维护人员以及那些对系统进行市场营销策划活动的人。

关键词（**Interoperability）**

Interoperability 互操作性，互用性 互操作性是指不同的计算机系统、网络、操作系统和应用程序一起工作并共享信息的能力。

结合构架发展历史：

单系统、消息传递（面向对象体系构架）->中间件支撑->面向服务体系构架->模型驱动构架、面向语义服务体系架构；

构架之美提供更多的软件复用、降低耦合度（解耦）、提高互操作性，以解决质量问题、效率问题、互操作问题、业务灵活性问题。

互操作性——这条术语用来描述的是不同的程序借助于同一套交换格式来交换数据，读写相同文件格式以及采用相同协议的能力。

是指不同的计算机系统、网络、操作系统和应用程序一起工作并共享信息的能力。

* 访问互操作性（API）

（看上面两个图，理解一下访问互操作性咯，我大概的理解意思就是不同系统可以提供标准的API，用户可通过访问互操作性访问不同的系统，读写相同的文件格式，进行通讯和数据交换）

* 连接互操作性（SOAP）

（大概的理解是不同的客户端可利用不同标准的传输协议与服务器连接通讯就是数据交换，例如SOAP提供的就是连接互操作性）

* 语义互操作性（XML）

对于深入推进互操作性的任何工作而言，语义互操作性都不可或缺。如果一个系统能够进行通讯和交换数据，那么，它就具备语义协同工作能力。就数据的通讯而言，基本的要素包括规定的数据格式、通讯协议以及接口描述等等。一般而言，XML或SQL标准提供的就是语义互操作性。

评估阶段：

* First：ATAM方法的表述
* Second：商业动机的表述
* Third：构架的表述
* Fourth：对架构方法进行分类
* Fifth：生成质量属性效用树
* Sixth：分析构架方法（P238）
* Seventh：集体讨论并确定场景的优先级
* Eighth：分析构架方法
* Ninth：结构的表述

构建效用树的作用，拿来做什么。（P236）

关键词（Quality Attribute Utility Tree）

通过效用树（Utility Tree）的机制来对质量属性目标进行详细的清晰的阐述，其作用在于是质量属性需求具体化，从而迫使设计师和客户代表准确的定义出他们将要提供的相关质量属性。

“效用”是小效用树的根节点，“质量属性”构成该树的2级节点。性能、可修改性、安全性、易用性和可用性是效用树的子节点。

How to create a skeletal system?

* 首先实现处理架构组建的执行和交互的软件部分。这可能要求生成实时系统中的调度程序；实现规则引擎以控制在基于规则的系统中规则的激发；实现多进程系统中的进程同步机制或客户机。之后就拥有了一个运行系统。这为添加有用功能提供了基础。
* 现在可以选择把提供功能的那些元素添加到该系统。这种选择可能是根据以下因素做出的：首先处理问题最多的部分来降低风险；或现在开发人员的类型水平；或尽可能快地将有用的产品推向市场。
* 现在了提供下一个功能增量的元素后，就可以采用使用结构，以获知应该在系统中采用什么软件来支持该功能。

随着过程的继续，系统的增量越来越大，直到软件开发完毕。

评估小组根据一些常见的基本问题或系统缺陷将风险分组为风险主题，从而增添了价值，例如，可以将文档编写不充分或者文档陈旧的若干个风险归为一个风险主题，即没有足够重视文档的编写。可以将关于系统不能再采用多种硬件是正常运行和/或软件故障归结为一个风险主题，即未提供备份能力或未提供高可用性。

敏感点：影响到质量属性

例如，同时数据库客户机的数量将影响数据库每秒钟能处理的交易的数量。因此，对于用每秒钟的交易数据量的响应来说，给服务器分配客户机就是一个敏感点。

ABC是一个来自环境对架构以及反馈回环境的影响的循环。

事物/组织环境必定会影响架构设计决定。

软件架构反过来改变事物环境。架构设计是开发过程的每一步的部分。这些会强烈影响组织的底线。

**什么是好的SA:**

**关于过程的建议**

1. 架构的设计应该由一位设计师或者在某设计师领导下的小组完成
2. 设计师应全面掌握系统的功能需求，并且应有一份所设计架构应满足的划分了优先级的质量属性列表。
3. 架构的文档应该完备。
4. 应该让各种涉众积极参与设计方案的评审。
5. 对架构人身进行分析，得出可应用的量化度量指标。
6. 架构的设计应有助于增量式实现。
7. 允许架构带来一定的资源争用，但应清楚地给出这些资源争用的解决方案。

**架构设计的重要性和时机：**

把架构作为软件开发过程基础的人和组织都需要理解构架在其生命中的位置。

何时开始设计：在生命期模型中，构架设计就是从初步的需求分析开始逐步进行迭代。

**ADD方法的步骤：**

1. 选择要分解的模块
2. 根据这些步骤对模块进行求精

2.1、从具体的质量场景和功能需求集合中选择构架驱动因素

2.2、选择满足构架驱动因素的构架模式。

2.3、实例化模块并根据用例分配功能。

2.4、定义子模块的接口。

2.5、验证用例和质量场景并对其求精。

3、对需求进一步分解的每一个模块重复上述步骤

**架构评审的重要性：**

All design involves tradeoff in system qualities（设计即折中）

System qualities are largely dependent on architectural decisions  
体系结构极大地影响系统质量

Promoting one quality often comes at the expense of another quality  
提高一个质量，经常会降低另一个质量

A software architecture is the earliest life-cycle artifact that embodies significant design decisions: choices and tradeoffs.  
“选择与折中”是设计中首要考虑的问题，软件体系结构是软件生命周期中最早一个遇到此问题的

Choices are easy to make, but hard to change once the system is implemented  
选择很容易做，但是一旦系统已经实现，就很难更改

**成本收益：**

Clarified quality attribute requirements

明确质量需求

Improved architecture documentation

提高体系结构文档质量

Documented basis for architectural decisions

文档化了的体系结构方案基础

Identified risks early in the life-cycle

及早发现风险

Increased communication among stakeholders

促进了角色之间的交流

The results are improved architectures.

结果是，体系结构得到改进

**时机：**

Academically, the time to use ATAM is right after the architecture has been specified when there is little or no code.

学术上说，应该在体系结构确定之后使用ATAM，此时没有或者仅有少量的代码

However, in practice, ATAM has been very effective in the following situations:

Evaluating alternative candidate architectures

评审候选的体系结构

Evaluating existing systems prior to committing to major upgrades

在升级之前，评审已有的系统

Deciding between upgrade or replace

无法决定是升级还是替换时

**前提条件：**

**Clients must have a Software Architecture**

**客户必须有一个软件架构**

Scope/scale must be manageable

其作用范围和程度必须可管理

ATAM will not work if the software architecture has not been created yet

如果体系结构还没有被建立，那么ATAM毫无用武之地

ATAM team members will review architectural artifacts, and may help refine documentation ATAM

组将评估体系结构，并帮助改善文档

Architect must prepare an architecture presentation

架构师必须准备一个体系结构讲解

**Clients must prepare a business/mission goals presentation**

**必须有一个商业/任务目标讲解**

**ATAM will review architecture artifacts, presentations, and read ahead material to become familiar with domain ATAM**

**要事先阅读一些材料来熟悉这个领域**