**Uml图**

**看pad**

**软件模式是将模式的一般概念应用于软件开发领域，即软件开发的总体指导思路或参照样板。**

**针对某一类反复出现的问题给出在一定约束条件下通用的解决方案**

**软件模式的基础结构由4个部分构成：问题描述、前提条件（环境或约束条件）、解法和效果。**

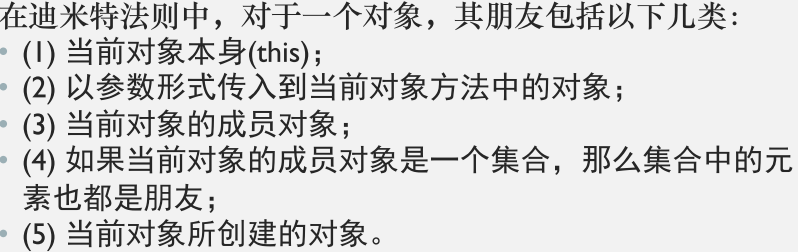
**设计模式：大三律(Rule of Three)：只有经过三个以上不同类型（或不同领域）的系统的校验，一个解决方案才能从候选模式升格为模式**

**目标：**

**1开闭原则⭐5：扩展开放，修改关闭**

**指导：**

**2最小知识原则⭐3：迪米特法则**

****

**基础：**

**3单一职责原则⭐4：每个类承担一个职责**

**一个类（或者大到模块，小到方法）承担的职责越多，它被 复用的可能性越小**

**实现 高内聚、低耦合**

**可变性封装原则⭐5是开闭原则的具体表现**

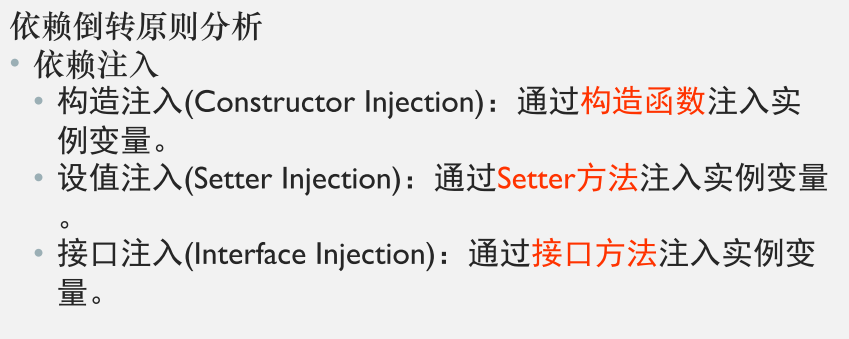
**实现：**

**4依赖倒转原则⭐5 OOP主要手段**

**高层模块不应该依赖低层模块，它们都应该依赖抽象**

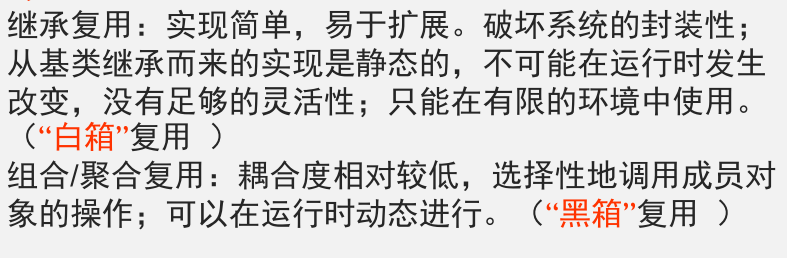
**抽象不应该依赖于细节，细节应该依赖于抽象**

<https://blog.csdn.net/king123456man/article/details/81626127>

****

**5合成复用原则⭐4**

**尽量使用对象组合，而不是继承来达到复用的目的**

****

**6里氏代换原则⭐4**

**所有引用基类（父类）的地方必须能透明地使用其子类的对象**

* 子类必须实现父类的抽象方法，但不得重写（覆盖）父类的非抽象（已实现）方法
* 子类中可以增加自己特有的方法
* 当子类覆盖或实现父类的方法时，方法的形参要比父类方法的输入参数更宽松
* 当子类的方法实现父类的抽象方法时，返回值要比父类更严格

**7接口隔离原则⭐2**

**一旦一个接口太大，则需要将它分割成一些更细小的接口，使用该接口的客户 端仅需知道与之相关的方法即可**

**在 JDK 中，java.util.Stack是java.util.Vector类的子类，该设计 合理吗？若不合理，请分析解释该设计存在的问题**

**这个类是java.util.Vector的子类，由于是继承，stack就会将父类的方法继承过来  
如：add(int index,E element)等（具体见Vector的API说明）。这个破坏了stack约定的规则（只能从栈顶进，栈顶出）**

**如果非要继承Vector合理的做法应该是什么：Stack不继承Vector，而只是在自身有一个Vector的引用，聚合**

1. **策略模式：🐖考**

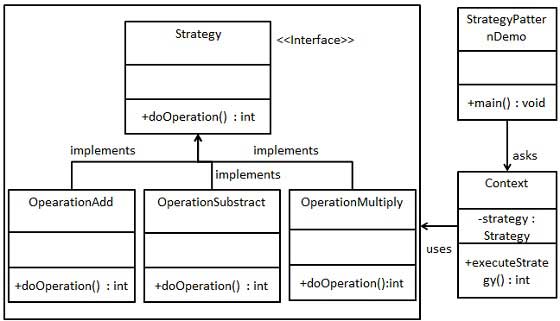
**• 需要换行功能的客户程序如果直接包含换行算法代码的话将会变得复杂，这使得客户程 序庞大并且难以维护, 尤其当其需要支持多种换行算法时问题会更加严重。**

**• 不同的时候需要不同的算法，我们不想支持我们并不使用的换行算法。**

**• 当换行功能是客户程序的一个难以分割的成分时 ,增加新的换行算法或改变现有算法将 十分困难**

**对象行为模式**

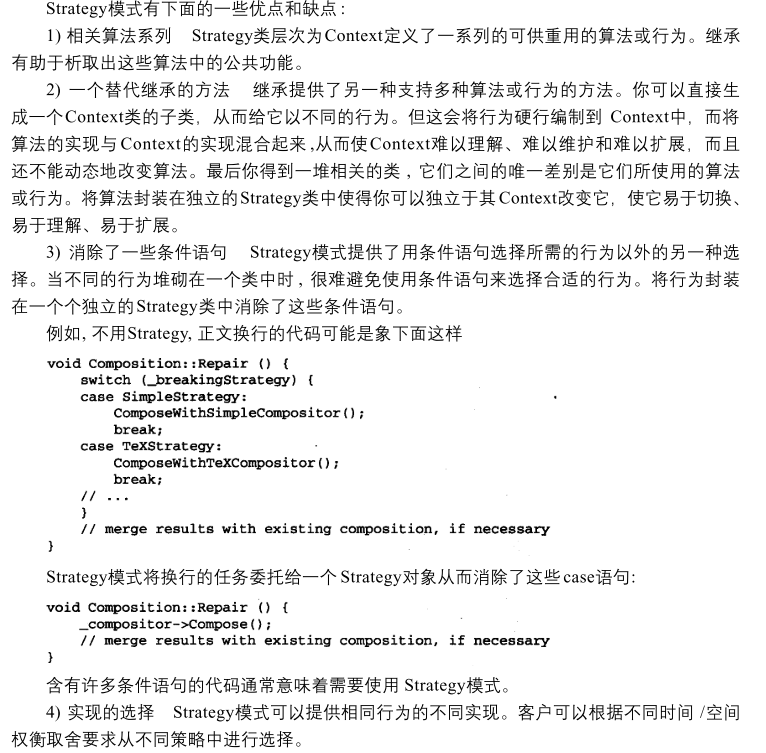
**别名policy**

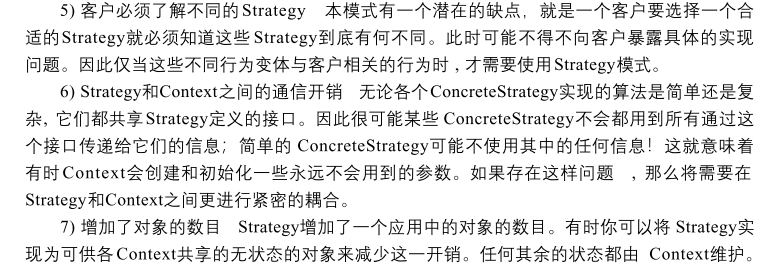


**联用-算法替换 享元模式S t r a t e g y对象经常是很好的轻量级对象**

**优缺点**

**优点：开闭原则 可扩展**

****

****

1. **简单工厂：肯定考🐖**

**静态方法工厂**

**简单工厂模式包含如下角色**

**Factory：工厂角色**

**Product：抽象产品角色**

**ConcreteProduct：具体产品角色**

**传正确的参数**

public class ComputerFactory {

public static Computer createComputer(String type){

Computer mComputer=null;

switch (type) {

case "lenovo":

mComputer=new LenovoComputer();

break;

case "hp":

mComputer=new HpComputer();

break;

case "asus":

mComputer=new AsusComputer();

break;

}

return mComputer;

}

}

**缺点是当有新产品要加入到系统中时 ，必须修改工厂类，加入必要的处理逻辑，这违背了“ 开闭原则”**

1. **工厂方法：🐖**

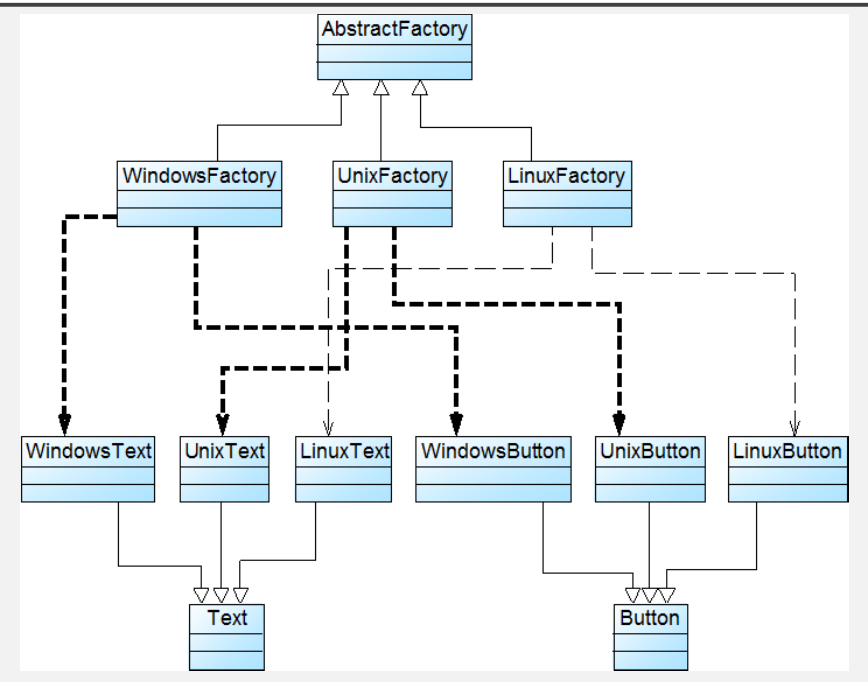
**是简单工厂的推广，保持其优点**

**优点：增加新的产品类时无须修改现有系统，并封装了产品对象的创建细节，系统具有良好的灵活性和可扩展性**

1. **抽象工厂：🐖**

**四个角色**

****

****

**抽象工厂模式与工厂方法模式最大的区别：工厂方法模式针对的是一个产品等级结构，而抽象工厂模式则需要面对多个产品等级结构**

**优点：隔离了具体类的生成，使得客户并不需要知道什么被创建，而且每次可以通过具体工厂类创建一个产品族中的多个对象，增加或者替换产品族比较方便，增加新的具体工厂和产品族很方便**

**缺点：增加新的产品等级结构很复杂，需要修改抽象工厂和所有的具体工厂类，**

**对“开闭原则”的支持呈现倾斜性：增加新的工厂和产品族容易， 增加新的产品等级结构麻烦**

1. **建造者：**

**四个角色：**

**指挥者作用：**

**优点：客户端不必知道产品内部组成的细节，将产品本身 与产品的创建过程解耦，使得相同的创建过程可以创建不同的产品对象，每一个具体建造者都相对独立，而与其他的具体建造者无关，因此可以很方便地替 换具体建造者或增加新的具体建造者，符合“开闭原则”，还可以更加精细地 控制产品的创建过程；**

**缺点：由于建造者模式所创建的产品一般具有 较多的共同点，其组成部分相似，因此其使用范围受到一定的限制，如果产品 的内部变化复杂，可能会导致需要定义很多具体建造者类来实现这种变化，导致系统变得很庞大**

**适用情况：**

1. **原型模式：**

**对象创建型模式**

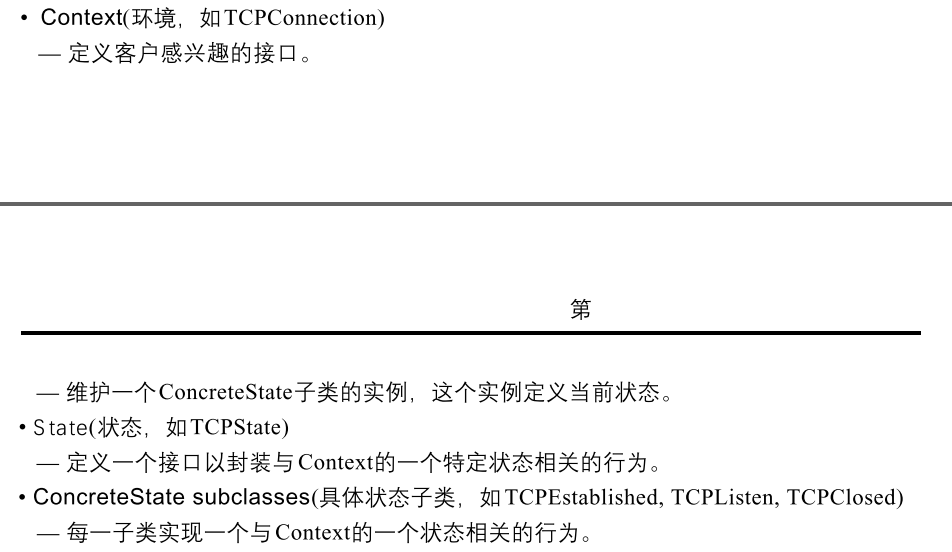
**Java object**

**浅克隆**

**原型管理器**

1. **状态模式：肯定考🐖**

**三角色：**

****

public interface PersonState {

void movies();//看电影

void shopping();//逛街

}

public class DogState implements PersonState {//单身狗状态

@Override

public void movies() {

System.out.println("一个人偷偷看岛国大片");

}

@Override

public void shopping() {

//单身狗逛条毛街啊

//空实现

}

}

public class LoveState implements PersonState {//恋爱状态

@Override

public void movies() {

System.out.println("一起上电影院看大片~");

}

@Override

public void shopping() {

System.out.println("一起愉快的逛街去~");

}

}

public class Context {

private PersonState mPersonState;

public void setPersonState(PersonState personState) {

mPersonState = personState;

}

public void fallInLove() {

System.out.println("恋爱了,陷入热恋状态:");

setPersonState(new LoveState());

}

public void disappointmentInLove() {

System.out.println("失恋了,变成单身狗状态:");

setPersonState(new DogState());

}

public void movies() {

mPersonState.movies();

}

public void shopping() {

mPersonState.shopping();

}

}

public void test() {

Context context = new Context();

context.fallInLove();

context.shopping();

context.movies();

context.disappointmentInLove();

context.shopping();

context.movies();

}

**主要优点：封装了转换规则**

**缺点：不满足开闭原则，肯定考（为什么不满足、用别的替换）**

**F l y w e i g h t模式( 4 . 6 )解释了何时以及怎样共享状态对象。**

**状态对象通常是S i n g l e t o n ( 3 . 5 )。**

**其缺点在于使用状态模式会增加系统类和对象的个数，且状态模式的结构与实现都较为复杂，如果使用不当将导致程序结构和代码的混乱，对于可以切换状态的状态模式不满足“开闭原则”**

**适用情况**

**使用状态模式可以描述工作流对象（如批文 ）的状态转换以及不同状态下它所具有的行为**

**使用状态模式可以对游戏角色进行控制**

1. **命令模式：肯定考🐖**

**识别这个模式**

**动作( A c t i o n )，事务( Tr a n s a c t i o n )**

**场景**

**优点：降低系统的耦合度，增加新的命令很方便，而且可以比较容易地设计一个命令队列和宏命令，并方便地实现对请求的撤销和恢复**

**缺点：可能会导致某些系统 有过多的具体命令类**

**在不同语言之间差异比较大：java、c++**

1. **观察者模式：会考(高标)**

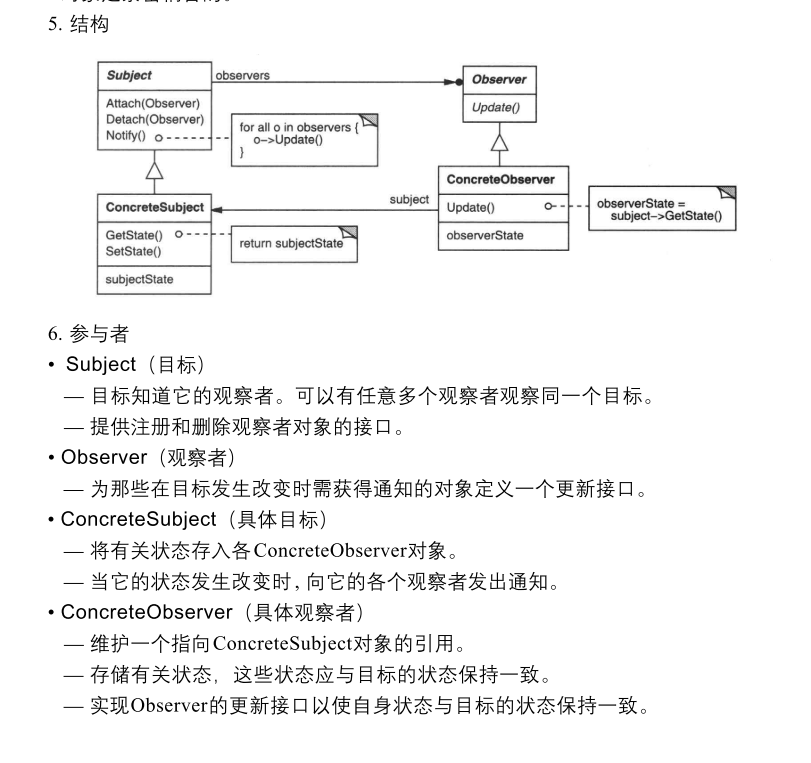
**别称：发布-订阅模式、模型-视图模式、源-监听器模式、从属者模式 Dependents**

**联用—中介者 通过封装复杂的更新语义 , ChangeManager充当目标和观察者之间的中介**

**者**

**变体**

**Java中的使用**

****

1. **中介者模式：好像不怎么考**

**适用情况**

1. **模板方法模式：会考**

**识别**

**容易出现在别的方法中**

**方法可以分为模板方法和基本方法**

**为什么符合开闭原则**

1. **适配器模式：🐖**

**变化**

1. **组合模式：**

**透明性和安全性（不限于组合模式）**

**好处？**

1. **桥接模式：🐖**

**变化：抽象化(Abstraction)与实现化(Implementation)脱耦， 它们可以沿着各自的维度独立变化如何变化**

1. **装饰模式：**

**适合地添加行为方式**

**类图**

**缺点:怎样对对象的组合进行控制**

1. **外观模式：**

**思想**

**如何应用在具体问题上—简答**

1. **享元模式：**

**能做到共享的关键是区分内部状态和外部状态**

1. **代理模式：**

**思想**

**远程代理、虚拟代理、保护代理**

**\*🐖是被单列出来的模式**

**联用：策略模式 观察者模式 组合模式**

**防御式编程：**

**防御式编程是什么**

**子程序应该不因传入错误数据而被破坏，哪怕是由其他子程序产生的错误数据**

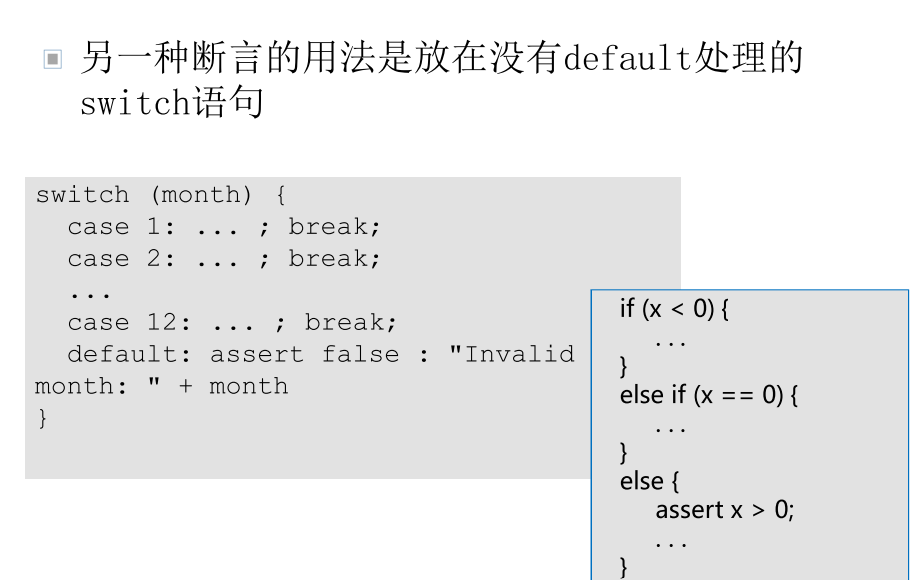
**–换句话说，要承认程序都会有问题，都会被修改**

**为什么要防御式编程**

**Assert error 异常处理不同的应用场景/异同：分析代码**

**断言处理代码中不应发生的错误**

**错误处理：**

**处理那些预料中可能发生的错误**

**用错误处理代码来处理预期会发生的非正常情况， 用断言来检查永远不该发生的情况**

**隔离程序（模块上）**

**隔栏的使用使断言和错误处理有了清晰的区分**

**隔栏部分包含了“脏数据” –隔栏外部的程序应使用错误处理技术，在那里对数 据做的任何假定都是不安全的**

**通过隔离部分之后的是“干净数据**

**辅助调试代码的使用与去除**

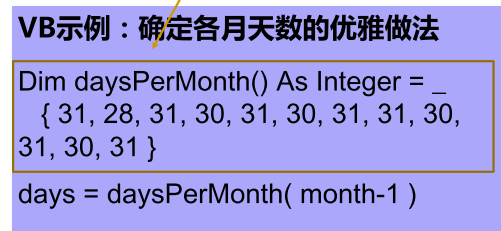
**表驱动法：**

**目的**

**原理（处理什么问题）**

**三种（代码）**

**直接访问 本身是索引**

****

**索引访问表 本身不是索引不能一对一**

**阶梯访问表**

**15年考题**

1. Where do software architecture come from? List five possible sources of software architecture.
2. What distinguishes an architecture for a software product line from an architecture for a simple product**?**
3. Describe relationships between architecture patterns and tactics. List four tactics names and describe their usage.
4. Briefly describe the general activities involved in a software architecture process.
5. Explain the context, benefits and limitations of Broker Architecture Pattern.
6. Why should a software architecture be documented using different views? Give the name and purposes of 4 example views.
7. Briefly describe the fundamental principles of SOA and discuss the impact of SOA on quality attributes like interoperability, scalability and security.
8. Why SPL and MDA have high reusability? Compare and discuss their commonality and differences.

**17年考题**

1. **Briefly describe the general activities in a software architecture process, and the major inputs and outputs at each activity.**
2. **What distinguishes an architecture for a software product line from an architecture for a single product?**
3. What are generic design strategies applied in designing software? Give a concise working example with software architecture for each strategy.
4. **How to model quality attribute scenarios? Graphically model two quality attributes in “stimulus-response” format: availability and modifiability.**
5. Please name at least three Object-Oriented principles, and explain how they are applied in Strategy pattern?
6. What should be included in a typical software architecture documentation package? Briefly describe each component and its purpose.
7. 软件设计的的三个变化维度，每个维度的变化点。differing binding time如何影响可修改性和可测试性。

Variation: forms of variation \* software entity varied \* binding time

**18 年考题（梁神回忆）**

1. 软件架构的关注点有哪些？利益相关方有哪些？
2. Software requirements, quality attributes, ASRs 的区别和联系
3. What is the nature of component-connector style? 以 MVC pattern 举例
4. Layered pattern 和 Multi-tier pattern 的区别
5. **为什么软件架构需要用不同的视图描述？举出四种视图的例子（列出名称和目的）**
6. 软件产品线架构如何实现可变性？描述可变性机制的工作方式
7. 设计一个飞行模拟软件，要求能模拟多种飞机的特性。为了在将来支持更多飞机种类，要求使用策略模式。画出架构图和类图

**19年押题**

这 7 个估计考的概率很大， 看明天是不是打脸吧…..

1. 软件产品线…..（肯定会考，但是没法猜考什么）
2. 微服务架构

**19 年最终考题 （个人回忆）**

1. **Explain the context, benefits and limitations of Broker Architecture Pattern.**
2. **微服务 和 SOA 的区别，相同点**

**微服务：**微服务架构风格是一种将一个单一应用程序开发为一组小型服务的方法，每个服务运行在自己的进程中，服务间通信采用轻量级通信机制，这些服务围绕业务能力构建并课通过自动部署机制独立部署。

微服务架构的本质仍然是一种分布式架构，也是面向服务架构的一种。

**优点**

1. **简单**
2. **团队独立性**
3. **松耦合**
4. **平台无关性**
5. **通信协议轻量级**

****

**Software Architecture in General**

**What is software architecture?**

**the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.**

**Structure, Elements, Relationships, Design**

**What does a software architect do?**

**1. Liaison 协调各方，沟通交流**

**2. Software Engineering 软件工程的基本技能**

**3. Technology Knowledge。软件开发技术的基本知识**

**4. Risk Management。 风险管理能力**

**Where do architectures come from? 15年**

**1. NFRs, 非功能需求，一般包括约束和质量属性**

**2. Architecture Significant Requirement 架构攸关需求。**

**3. Quality Requirements. 在功能需求的基础上需要满足的特征**

**4. Stakeholders 系统的利益相关者**

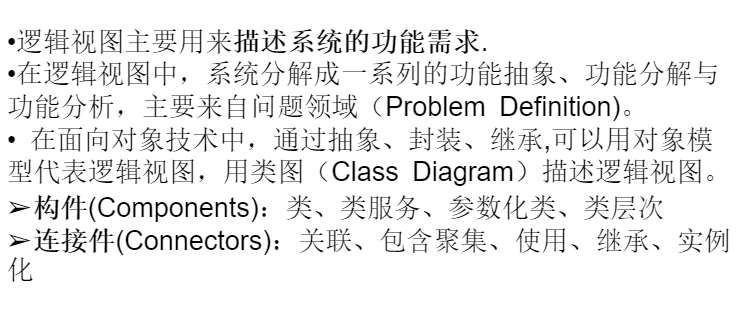
**5. Organisations 开发组织**

**6. Techinal Environments 技术环境**

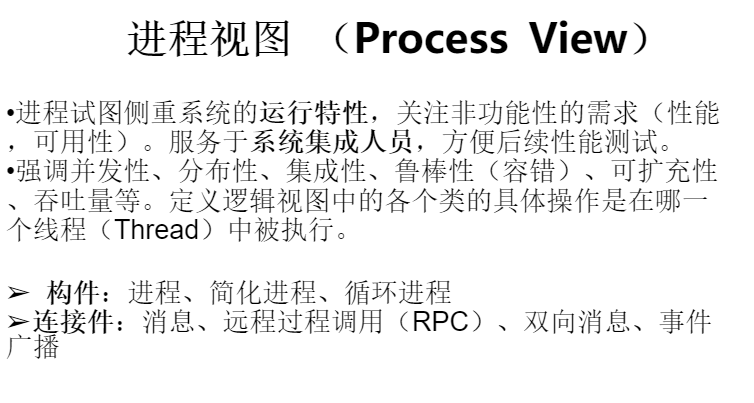
**不不仅仅局限于需求，可能是商业和技术决定的集合**

**Architecture Views架构视图 必考**

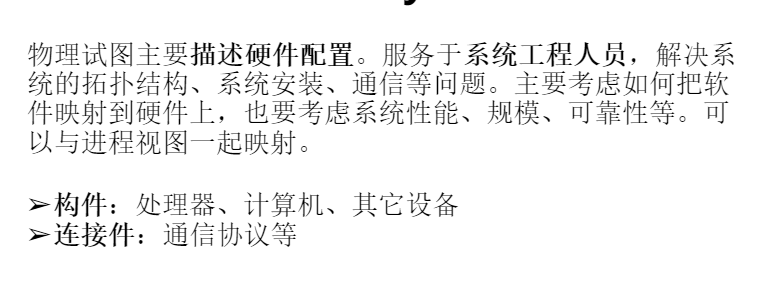
**Logical view：describes architecturally significant elements of the architecture and the relationships between them**

****

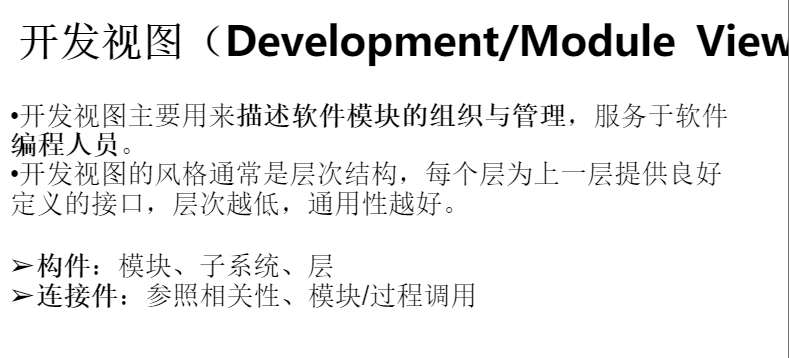
**Process view：Concurrency and communications elements of an architecture**

****

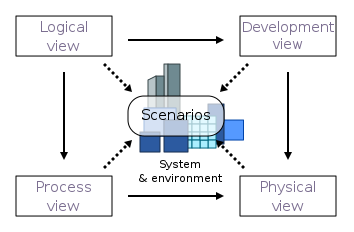
**Physical view：how the major processes and components are mapped on to the applications hardware**

****

**Development view：captures the internal orgnazition of software components**

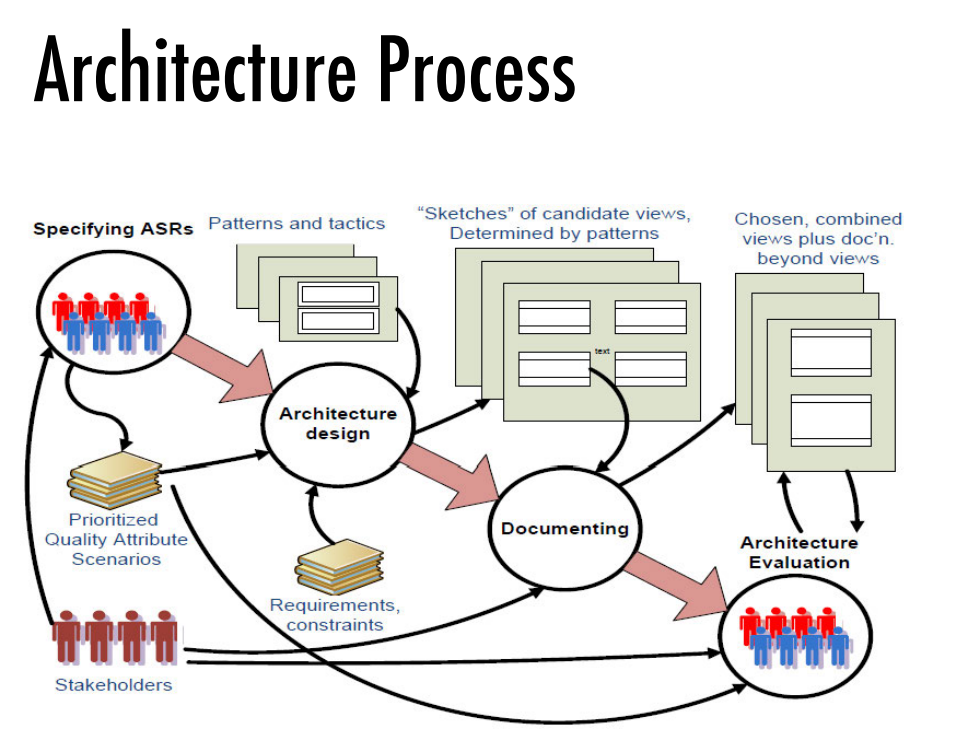
****

**Use case scenarios…**



**Architectural activities and process**

**Software architecture knowledge areas**

****

**Quality Attributes**

**Software Requirements**

**Functional requirements, Quality requirements (NFRs), Constraints**

**Quality Attributes 必考**

**已考建模：**

**15**

**考：availability and performance**

**17 题目：**

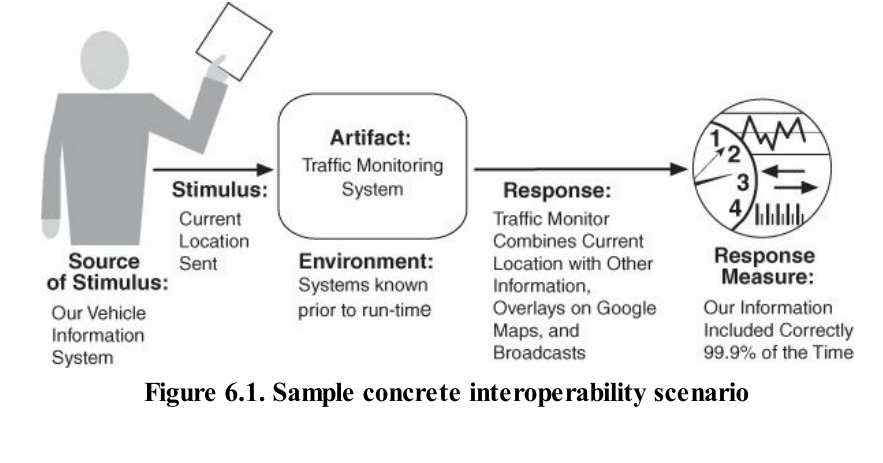
**考：availability 和 modifiability**

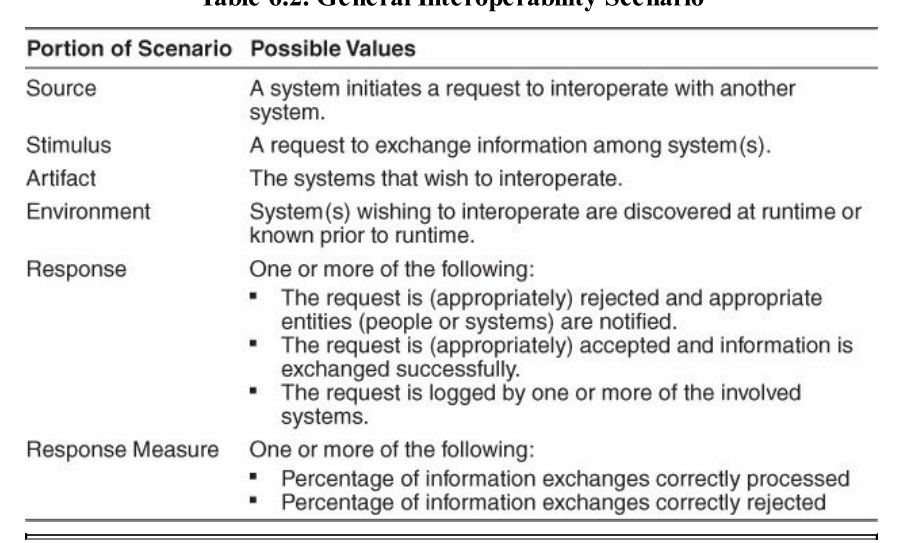
**18题目：stability vs. reliability, extensibility vs. scalability, security vs. safety**

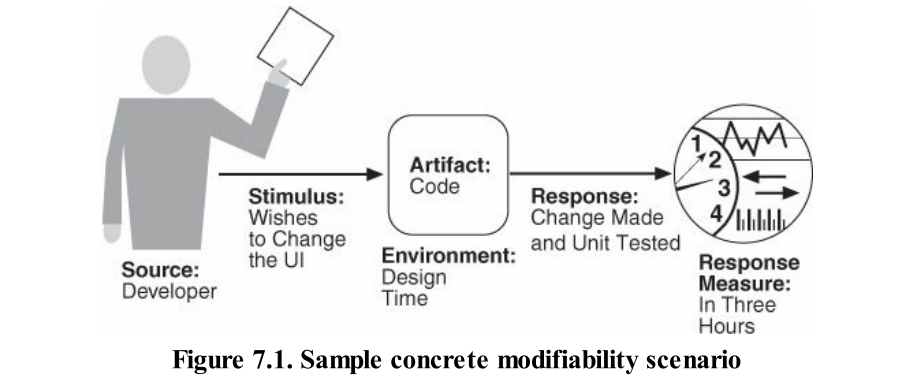
**考Interoperability 和 modifiability**

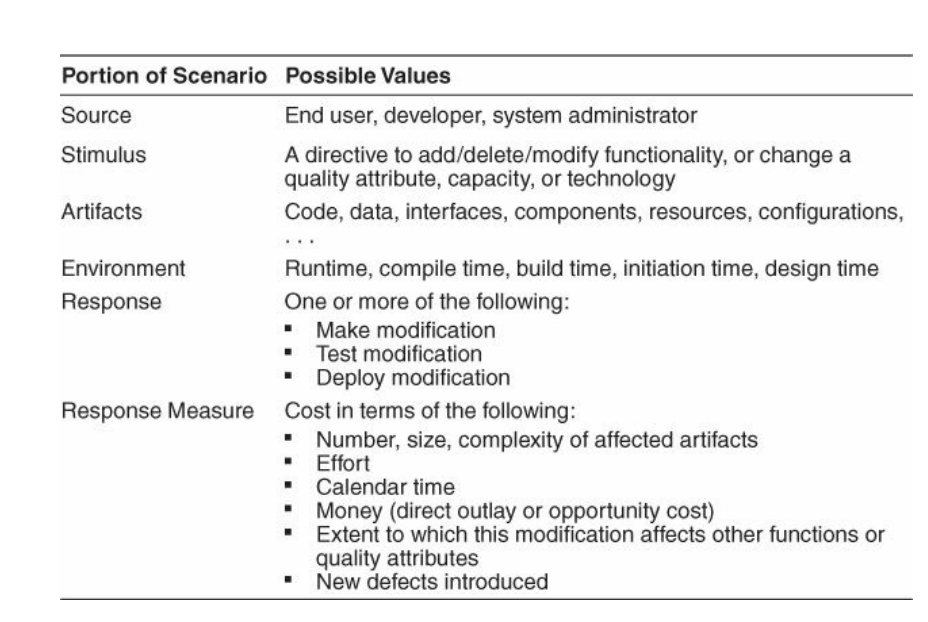
**今年 题目：dependability vs. security, sustainability vs. scalability, data integrity vs. consistency**

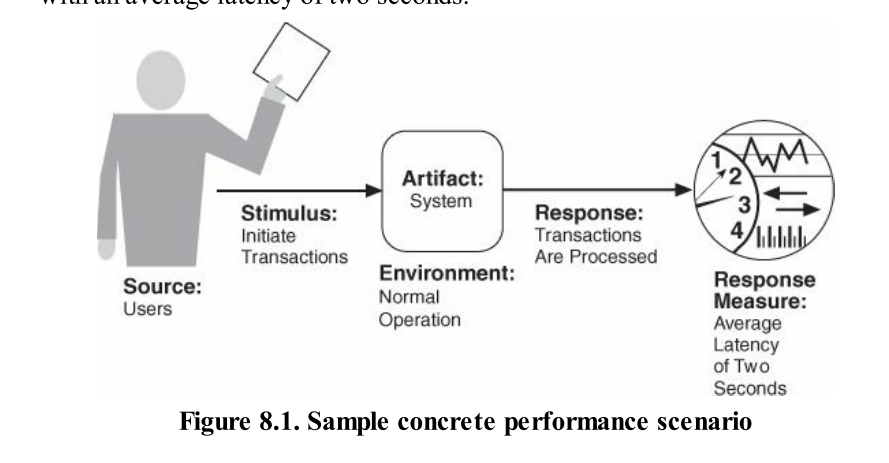
**考：Interoperability, Modifiability, Performance, Testability, Usability**

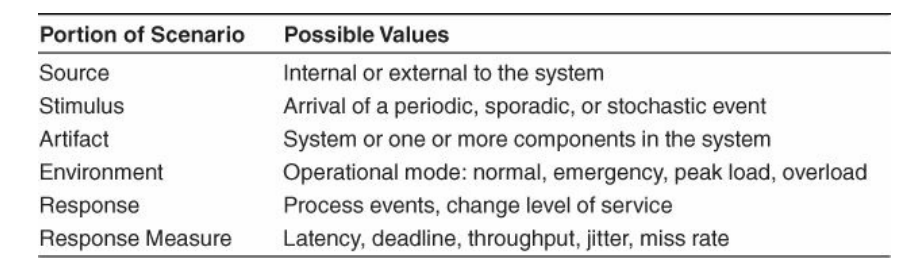
****

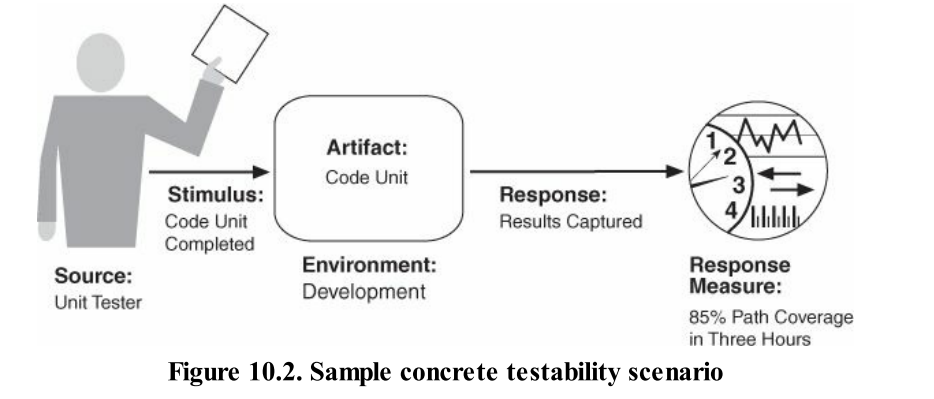
****

****

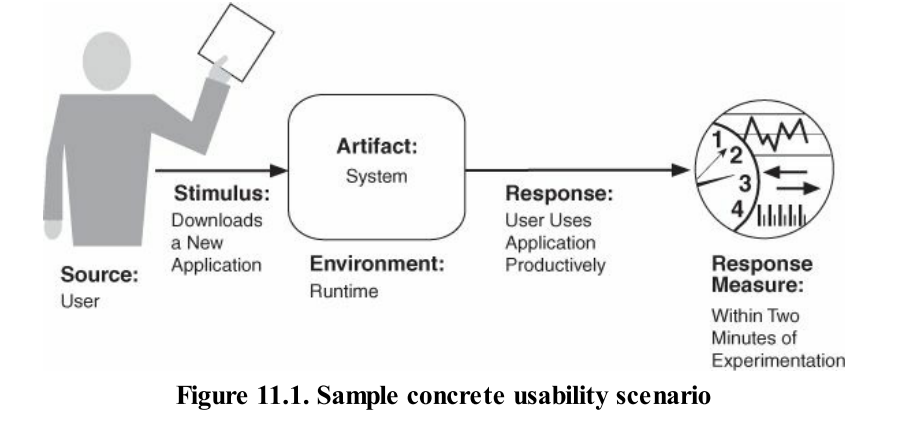
****

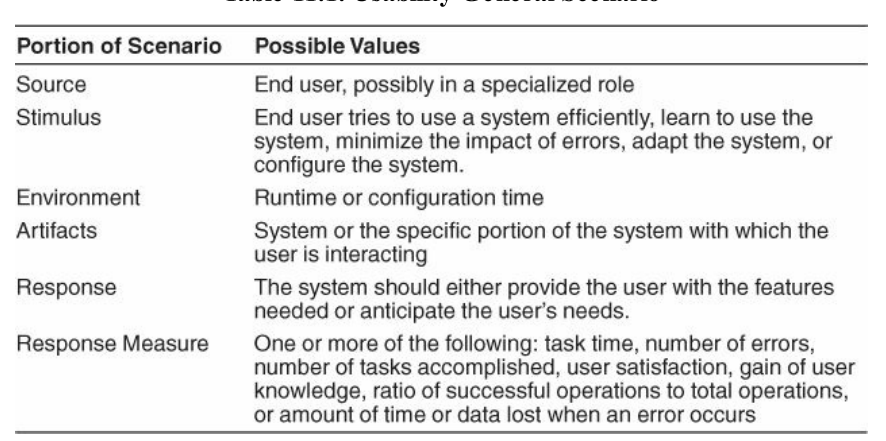
****

****

****

****

****

****

**Modeling quality attribute scenarios:**

**Source**

**Stimulus**

**Artifact**

**Environment**

**Response**

**Measure**

**Tactics for quality attributes**

**Architecturally Significant Requirements**

**题目**

What are ASR? List four sources and methods for extracting and identifying ASRs.

定义：requirements that will have profound effects on the architecture

Source：**Requirements documents, Interviews stackholders, Business goals, Utility tree**

**Method：**

**Architecture Patterns**

**Architecture Patterns**

**Context, Problem, Solution: elements + relations + constraints**

**Module Patterns**

**Layered pattern**

**Component-Connector Patterns**

**Broker pattern, Model-view-controller pattern, Pipe-and-filter pattern, Client-server pattern, Peer-to-peer pattern, Service-oriented pattern, Publish-subscribe pattern, Share-data pattern**

**Allocation Patterns**

**Map-reduce pattern, Multi-tier pattern**

**Patterns vs. Tactics**

**Designing Architecture**

1. **What are generic design strategies applied in designing software? Give a concise working example with software architecture for each strategy. (和17年一样的)**

**答案：**

**General Design Strategy**

**Abstraction, Decomposition, Divide & conquer, Generation and test, Iteration, Reuse**

**Attribute-Driven Design (ADD)**

**Choose a part to design**

**Marshal all ASRs for that part**

**Create and test a design for that part**

**Inputs to and outputs of ADD**

**8-step process: 1. confirm requirements, 2. choose an element to decompose, 3. identify ASRs, 4. choose a design satisfying ASRs, 5. instantiate elements & allocate responsibilities, 6. define interface, 7. verify & refine requirements, 8. repeat step 2-7 until all ASRs satisfied**

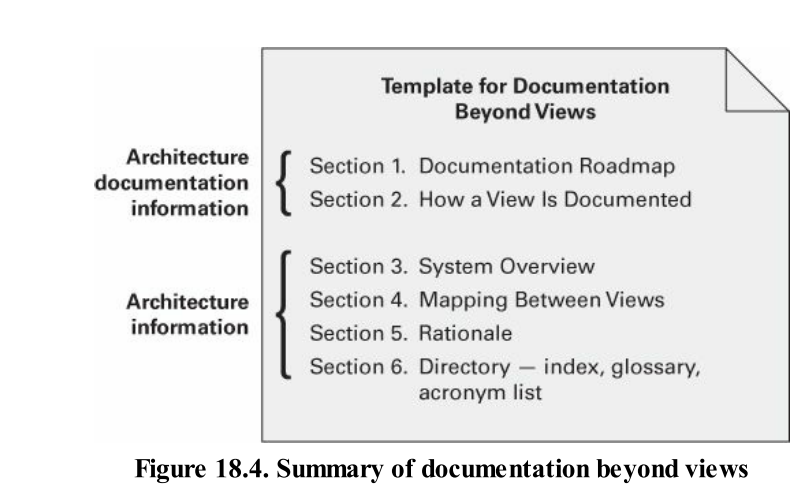
**Documenting Architecture**

**题目**

**What should be included in a typical software architecture documentation package? Briefly describe each component and its purpose. （和17年基本一样）**

**Document a view：primary presentation，element catalog，context diagram，variability guide，rationale**

**Beyond view ：**

****

**Views and Beyond**

**Views:**

**Styles (viewpoints), patterns and views**

**Structural views: module views, component-and-connector views, allocation views**

必考题

1. How it structed as a set of implementaion ———— Module styles
2. How it is structed as a set of elements taht have runtime behavior and interactions C&C styles
3. How it relates to non-software structures in its environment ——— Allocation styles

**Quality views**

Style: 关注组件和组件之间的交互和职责限制。更关注架构方法，不包含具体的上下文和 问题。 Pattern: 体现软件系统基本的结构组织模式。关注上下⽂和问题，以及如何解决。

View: 是系统中特定类型的元素及其关系的表示。不同的视图⽀持不同的⽬标和⽤户，强调不同的系统元素和关系，同时在不同程度上表示不同的质量属性。

**Documenting views: 1. build stakeholder/view table, 2. combine views, 3. prioritise & stage**

**Beyond views: documentation info & architecture info (mapping between views)**

**Documentation package: views + beyond**

**Evaluating Architecture**

**题目：**

**ATAM目的发现架构决策造成的风险**

**Describe outputs generated from each phase of ATAM process.**

**risks, sensitivity points, trade-off points 是什么？各举一个例子**

**risks风险：可能将来产生质量问题的架构决定**

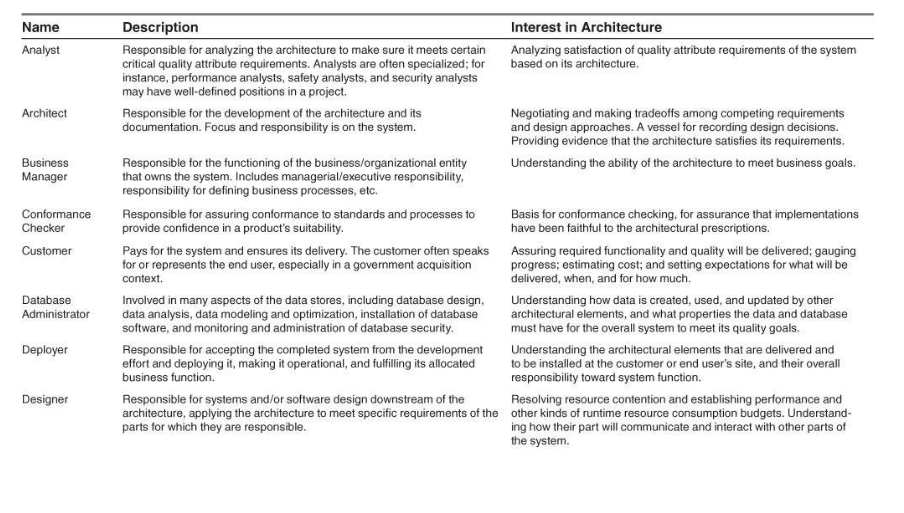
**sensitivity points敏感点：某个QA所敏感关注的架构决定**

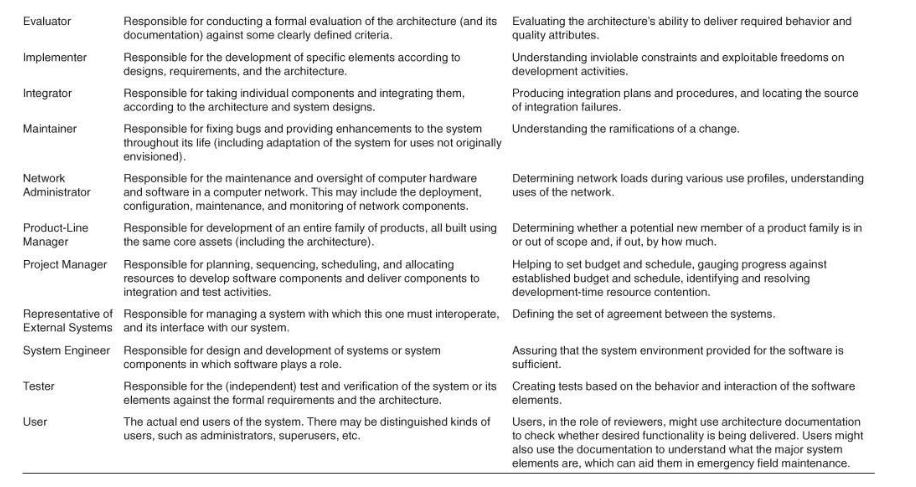
**trade-off points折衷：影响不止一个的架构决定**

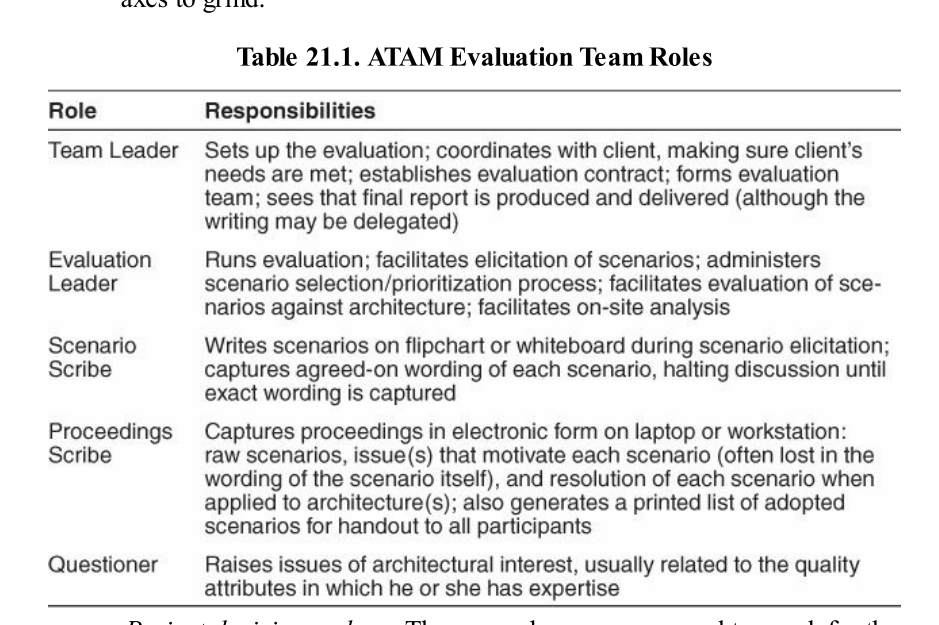
**nonrisks**

**描述 在 ATAM 的每一个过程中 有哪些 stake holder 和他们的职责**

**Architectural stakeholder**

****

****

****

**Phase 0 :参与者和准备阶段**

**参与者: 评估⼩组长和关键的项⽬决策者**

**输⼊:架构⽂档**

**输出: 评估计划: 谁、什么时间、提供什么样子的评估报告**

**Phase 1:评估(1)**

**参与者:评估小组和项目设计决策者(肯定包括了项⽬经理和架构师)**

**输出: 架构简要展示、业务目标、质量属性和相关场景、效⽤用树、⻛险和非⻛险点、敏感点、权衡点**

**Step 1: 介绍ATAM⽅法(评估小组长)**

**Step 2: 介绍商业动机(项⽬经理或系统客户)**

**Step 3: 介绍架构(⾸席结构师)**

**Step 4:识别使用的架构方法(评估小组)**

**Step 5:⽣成质量属性效⽤树(评估⼩组和项目设计决策者) 决定性的一步**

**Step 6:分析架构⽅法(评估小组) 确保方法是正确的 获得风险点、⾮风险点、敏感点和权衡点列表**

**Phase 2:评估(2)**

**参与者: 评估小组、项⽬设计决策者和架构涉众**

**输出: 从涉众群体获得的一个优先级场景列表、风险主题和商业动机**

**Step 1: 介绍ATAM⽅法和之前的结果(评估小组长) 重复以确保涉众也知道方法并回顾分享之前2~6步的结果**

**Step 7:头脑风暴、场景划分优先级(评估⼩组问涉众) 与质量属性效⽤树进行⽐对**

**Step 8:分析架构⽅方法(评估小组、架构师) 使用新产生的优先级靠前的场景、架构师解释与之相关的架构决定**

**Step 9:展示结果(评估小组)**

**Phase 3:后续⼯工作 Follow up**

**参与者:评估⼩小组和主要涉众**

**输出:最终的评估报告**

**ATAM 输出**

**架构简要介绍**

**业务⽬目标**

**以质量属性场景表示的带优先级的质量属性需求**

**效用树**

**⼀系列⻛险点和非风险点**

**风险主题**

**架构决定与质量需求之间的映射**

**敏感点、权衡点**

**最终评估报告**