1. 结合实际案例了解体系结构的风格？

经典的体系结构风格主要有以下几种：

1. 数据流风格：批处理序列；管道/过滤器

管道/过滤器：即用数据流的观点来观察系统。整个系统由一些管道和过滤器组成，需要处理的数据同管道传送给每一个过滤器，每个过滤器就是一个处理步骤。每个过滤器可以单独修改，功能单一，并且它们之间的顺序可以进行配置。数据通过了所有的过滤器后，就完成了所有的处理操作并得到了最终的结果。

典型的例子就是编译器，编译器是一个管道系统，它的功能包括此法分析，语法分析，语义分析和代码生成等等，每一个阶段的输出都是其余阶段的输入，每一个部分的工作都相当于是一个过滤器。

1. 调用/返回风格：主程序/子程序；面向对象风格；层次结构

层次结构常用于通信协议。最著名的分层风格的体系结构的例子就是OSI-ISO的分层通信模型。

C/S风格也是典型的层次结构:

表示层：社交信息的显示，并提供了更新和搜索等操作

功能层：具有搜索、在线聊天、离线留言、文件传输等功能

数据层：有数据库服务器提供留言、相册、好友信息等数据。

优点：使逻辑结构更为清晰，分类明确，给用户更好的体验

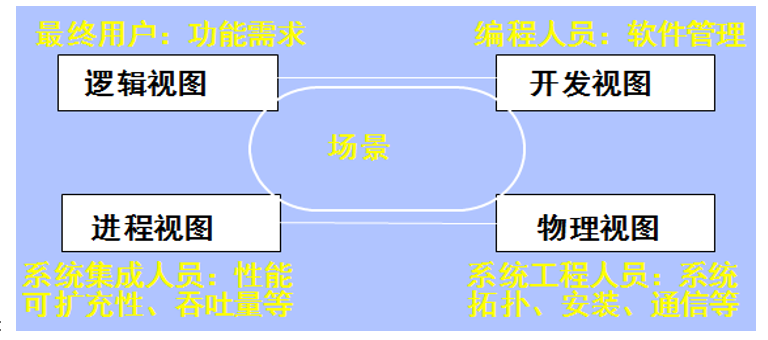
缺点：需要数据通信的支持，对网络的依赖很高，没有网络，许多功能将没有意义

1. 独立构件风格：进程通讯；事件系统

基于事件的隐式调用：构件发布事件到注册中心。或通过事件注册将某个过程或方法和事件进行绑定，一旦该事件被触发，所有与该事件想帮顶的过程或方法就被调用。

典型例子：编辑器和变量监视器为调试器的断点事件注册，遇到断点时，调试器发布断点，系统调用编辑器和变量监视器，优点是为软件重用提供了强大的支持，为该进系统带来了方便，缺点是不能保证响应事件的顺序以及无法保证数据交换。

1. 虚拟机风格：解释器；基于规则的系统
2. 仓库风格：数据库系统；超文本系统；黑板系统
3. 4-1视图是什么？

4+1视图的示意图如下所示：

“4+1”视图模型从5个不同的视角来描述软件体系结构。包括逻辑视图，进程视图，物理视图，开发视图和场景视图。每一个视图只关心系统的一个侧面，5个视图结合在一起才能反映系统的软件体系结构的全部内容。

* 逻辑视图

逻辑视图主要支持系统的功能需求，即系统提供给最终用户的服务。在逻辑视图中，系统分解成一系列的功能抽象，这些抽象主要来自问题领域。这种分解不但可以用来进行功能分析，而且可用作标识在整个系统的各个不同部分的通用机制和设计元素。

在面向对象技术中，通过抽象、封装和继承，可以用对象模型来代表逻辑视图，用类图来描述逻辑视图。

设计：逻辑视图中使用的风格为面向对象的风格，逻辑视图设计中要注意的主要问题是要保持一个单一的、内聚的对象模型贯穿整个系统。

* 开发视图

开发视图也称模块视图，主要侧重于软件模块的组织和管理。

开发视图要考虑软件内部的需求，如软件开发的容易性、软件的重用和软件的通用性，要充分考虑由于具体开发工具的不同而带来的局限性。

开发视图通过系统输入输出关系的模型图和子系统图来描述。

设计：在开发视图中，最好采用4-6层子系统，而且每个子系统仅仅能与同层或更底层的子系统通讯，这样可以使每个层次的接口既完备又精练，避免了各个模块之间很复杂的依赖关系。

设计时要充分考虑，对于各个层次，层次越低，通用性越强，这样，可以保证应用程序的需求发生改变时，所做的改动最小。开发视图所用的风格通常是层次结构风格。

* 进程视图

进程视图侧重于系统的运行特性，主要关注一些非功能性的需求。

进程视图强调并发性、分布性、系统集成性和容错能力，以及从逻辑视图中的主要抽象如何适合进程结构。它也定义逻辑视图中的各个类的操作具体是在哪一个线程中被执行的。

设计：进程视图可以描述成多层抽象，每个级别分别关注不同的方面。在最高层抽象中，进程结构可以看作是构成一个执行单元的一组任务。它可看成一系列独立的，通过逻辑网络相互通信的程序。它们是分布的，通过总线或局域网、广域网等硬件资源连接起来。

* 物理视图

物理视图主要考虑如何把软件映射到硬件上，它通常要考虑到系统性能、规模、可靠性等。解决系统拓扑结构、系统安装，通讯等问题。

当软件运行于不同的节点上时，各视图中的构件都直接或间接地对应于系统的不同节点上。因此，从软件到节点的映射要有较高的灵活性，当环境改变时，对系统其他视图的影响最小。

设计：大型系统的物理视图可能会变得十分混乱，因此可以与进程视图的映射一道，以多种形式出现，也可单独出现。

* 场景视图

场景可以看作是那些重要系统活动的抽象，它使四个视图有机联系起来，从某种意义上说场景是最重要的需求抽象。在开发体系结构时，它可以帮助设计者找到体系结构的构件和它们之间的作用关系。同时，也可以用场景来分析一个特定的视图，或描述不同视图构件间是如何相互作用的。

场景可以用文本表示，也可以用图形表示。

* 设计思想

逻辑视图和开发视图描述系统的静态结构，进程视图和物理视图描述系统的动态结构。

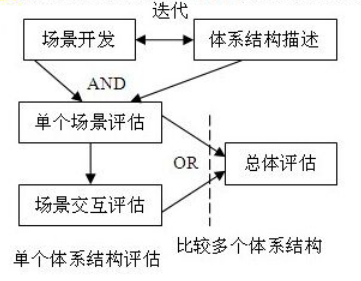
对于不同的软件系统来说。侧重的角度也有所不同。例如，对于信息管理系统来说，比较侧重于从逻辑视图和开发视图来描述系统，而对于实时控制系统来说，则比较注重于从进程视图和物理视图来描述系统。

1. 详细了解ATMA,SAAM两种评估方法。

ATMA以及SAAM都是属于基于场景的评估方式。

ATMA是指体系结构权衡分析方法，评估的过程主要由九个步骤组成，包括：

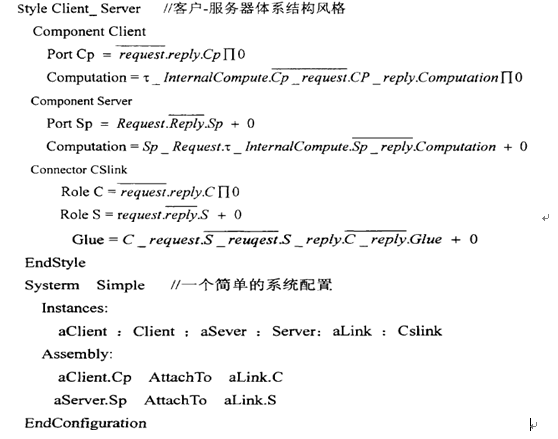
1. 描述ATAM方法：评估小组负责人向参加会议的风险承担者介绍ATAM评估方法。关键是要使每个人都知道要收集哪些信息，如何描述这些信息，将要向谁报告等。
2. 描述商业动机：参加评估的所有人员必须理解待评估的系统，在这一步，项目经理要从业务角度介绍系统的概况。
3. 描述体系结构：首席设计师或设计小组要对体系结构进行详略适当的介绍，这一步很重要，将直接影响到可能要做的分析及分析的质量。
4. 确定体系结构方法：由设计师确定体系结构方法，由分析小组捕获，但不进行分析。
5. 生成质量属性效用树：评估小组、设计小组、管理成员和客户代表一起确定系统最重要的质量属性目标，并对这些质量目标设置优先级和细化。这通常是通过构建效用树的方式来实现的，效用树的输出结果是对具体质量属性需求（以场景形式出现）的优先级的确定
6. 分析体系结构方法：评估小组根据效用树的结果对实现重要质量属性的体系结构方法进行考察，完成与该方法有关的质量属性的初步分析，从而产生一个体系结构方法或风格的列表，与之相关的一些问题，以及设计师对这些问题的回答。通常产生一个风险列表、敏感点和权衡点列表。
7. 讨论和分级场景：风险承担者需进行两项相关的活动那个：集体讨论用例场景（描述风险承担者期望使用系统的方式）和改变场景（描述风险承担者所期望的系统在将来变更的方式）
8. 分析体系结构方法：是第六步的重复，设计师把最高级别的场景映射到所描述的体系结构中，把新的到的最高优先级场景与尚未得到的体系结构工作产品对应起来。
9. 描述评估结果：把ATAM分析中所得到的各种信息进行归纳，并反馈给风险承担者。这种描述一般要采用辅以幻灯片的形式，但也可以在ATAM评估结束之后，提交更完整的书面报告。

SAAM指的是基于场景的软件体系结构分析方法，

总的来说，SAAM评估分为6个步骤：

1. 场景开发。通过集体讨论，风险承担者提出反映自己需求的场景。
2. 体系结构描述。SAAM定义了功能性，结构和分配三个视角来描述SA。功能性指示系统做了些什么，结构由组件和组件间的连接组成，而从功能到结构的分配则描述了域上的功能性是如何在软件系统中实现的。场景的形成与SA的描述通常是相互促进的，并且需要重复地进行。
3. 场景的分类。在分析过程中需要确定一个场景是否需要修改该体系结构。不需要修改的场景称为直接场景，需要修改的场景则称为间接场景。另一方面需要对场景设置优先级以保证在评估的有限时间内考虑最重要的场景。
4. 单个场景的评估。主要针对间接场景，列出为支持该场景所需要对体系结构做出的修改，并估计出这些修改的代价。而对于直接场景只需弄清体系结构是如何实现这些场景的。
5. 场景交互的评估。两个或多个间接场景要求更改体系结构的同一个组件就称为场景交互。对场景交互的评估，能够暴露设计中的功能分配。
6. 总体评估。按照相对重要性为每个场景及场景交互设置一个权值。
7. 能使用ADL或化学抽象机对一个简单系统的体系结构进行描述。

C/S体系结构：πADL描述：



代码解释如下：

客户-服务器风格Client-Server定义了客户构件Client、服务器构件Server、 连接件CSlink。配置Simple定义了具有一个客户构件、服务器构件的简单系统。

客户构件Client具有一个端口Cp，它的行为交互协议是：发出服务请求 request，收到回答reply后，继续执行进程cp或决定结束服务请求而执行进程 O。Client的计算行为是：首先进行内部计算f—internaiCompute，然后通过端口C≯ 发出服务请求印一request，等到端口cp得到回答cp—reply后，Computation继 续进行该过程或决定结束。 运用端口名称修饰、限定行为名字，防止各个端口的行为同名，因此 Cp．request和C≯reply表示端12 Cp发出服务请求request和得到回答

reply。 服务器构件Server具有一个端口sp，它的交互行为协议是：接受服务请求 Request，并给出回答Reply，然后根据是否还有服务请求决定继续执行该过程 或是结束。它的计算行为是：通过端口．s扫接受服务请求sp 后，执行内 部计算f ，然后通过端口．跏回答服务请求．．RequestInternalCompute sp reply，再根据 是否还有服务请求决定继续执行计算或结束。 连接件CSLink具有两个角色c、S，客户角色c的行为协议是：发出服务请 求request，等待回答reply，然后继续执行该过程或自主决定结束。服务器角 色S的行为协议是：接受外部服务请求Request，给出回答Reply，然后根据外 部情况决定继续执行或结束。它的粘接规约部分Glue的交互连接协议是：看到 角色c发出的服务请求C request后，向角色S发出服务请求S—reuqest，得到 角色S的回答S—reply后，向角色C做出回答C—reply，然后根据角色C的执行 情况决定继续执行该交互协议或结束。

1. 详细了解设计模式并在实际项目中如何运用模式.

设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。用于解决在特定环境下、重复出现的、特点问题的解决方案。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。

**基本设计模式：**

总体来说设计模式分为三大类：

创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式

1. 体系结构的测试是什么，熟悉使用抽象化学机进行测试的方法

软件体系结构测试是**检查**软件设计的**适用性**，**不考虑**软件的**实现代码**。与传统的软件测试一样，基于体系结构的软件测试也需要研究测试**内容**、测试**准则**、测试**用例**、测试**充分性**及测试**方法**等问题。

化学抽象机是一种对**动态**软件体系结构的分析、测试非常有用的**形式化描述技术**。CHAM描述的软件体系结构**规格说明**包含四部分：

（1）表达系统构件和连接件的**语法描述**，构件分为3类：数据元素、处理元素和连接元素。

（2）系统的**初始状态**

（3）一套**反应规则**

（4）一组表示系统**最终状态**的**溶液集**

基于CHAM的软件体系结构测试过程：

（1）**建立**基于CHAM的软件体系结构**测试模型**。

（2）根据CHAM描述的软件体系结构规格说明**导出标号迁移系统**（LTS），利用LTS**描述**系统的动态**行为**。

（3）得到LTS后，需要**定义**基于LTS的**测试覆盖准则**，然后根据选取的覆盖准则，**生成**基于此准则的**测试路径**