题型：

简答题、问答题、分析设计题

英文题目、中文或英文大题

个别题目可能需要画图

基础内容 70%

高阶内容 30%

期末考试 40% 作业和 project 占到 60%

Software Architecture in General

1.什么是软件架构？

Structure,Element,Relationships,Design

2.软件架构做什么？

架构决定了结构：接口，传输（数据传输机制（函数调用，远程调用，同步信息），流控制）和依赖，责任

3.软件架构如何而来？（依据）

NFRs（how well）[技术约束，逻辑约束，质量属性],ASRs,Quality Requirements,Stakeholders,Organizations,Technical Environments

4.架构视图 4+1 ViewModel

逻辑视图 logical、过程视图 process、物理视图 physical、开发视图 development

+

Architecture use cases

5.架构活动和过程

（包含几个步骤，四大步骤，有图）

Stakeholder 发现重要的需求（ASRs）‘

使用 Pattern\tactic 解决问题

视图、文档

评估

Quality Attributes

1.选件需求

功能需求、质量需求（NFRs)，约束

2.质量属性

两大类：

1、 执行时观察到的：

性能、安全、有效性、可用性、

Performance.security,availability,usability

2、 观察不到的：

可修改性、移动性、重用性、测试性

Modifiability,portability,reusability,testability

质量属性必须全程关注，高于系统功能性
一旦质量属性不过关就要重新设计
不能忽视
-->引入了体系结构来处理质量问题

场景： Modeling quality attributes:Source,Stimulus,Artifact,Environment,Response,Measure
两大类：
普通的即系统独立 scenarios
具体的 （是 general scenarios 的例子）

PPT 上提过的质量属性
3.Architecturaly Significant Requirements
How to gather and identify ASRs:Requirements,Intervies,Business goals,Utility tree

Architecture Patterns
Architecture Patterns
描述: Context,Problem,Solution:elements+relations+constraint

（DSSA） Domain-Specific Software Architecture
针对一个领域的，

Module Patterns
分层 Layered pattern

中介模式？Broker Pattern
a runtime component---broker to mediate the communication
元素：Client,Server,Broker,Client-side proxy,Server-side proxy
流程：Client->(proxy)->broker->(proxy)->server
缺点：•由于转发带来了效率的降低和延迟，客户端和服务器之间的联系可能存在瓶颈
•本身会带来错误
•增加了复杂度
•易导致安全攻击
•测试难度大

MVC 模式：
优点：
•拓展性良好：易于增加功能，不必改变程序结构

•部署快：由于模型独立于视图，也因此可以直接移植到新平台，只要修改视图和控制器

•速度快：界面与后台分离，提高了加载速度

•易于维护：同样是因为视图和逻辑分离，易于定位维护

•高重用性：不同的视图可以访问同一个逻辑代码

缺点：

复杂度高，不适用于简单的用户接口

•这种抽象不适用于部分用户接口

Pipe-and-filter 模式:

以数据流的形式啊

优点：

•降低了复杂度：一个过滤器只要实现单一的功能

•降低了耦合度：过滤器之间依赖很小

•灵活：可以通过组合的方式实现新的功能

缺点：

•不适用于交互系统

•大量独立的 filter 带来了计算的负担

•也不适合长时间运行的计算

--看 word

Component-Connector Patterns

Broker, MVC, Pipe-andfilter pattern,Client-Server,Peer-to-Peer , Service-orientated,Publish-subscribe,Share-data

Allocation Pattern

Map-reduce pattern, Multi-tier pattern

Patterns vs. Tactics

Tactics 是比 Pattern 密度更小的解决方案，Pattern 会包含一定的 Tactics，Pattern 本身可以包含其他的 Pattern，Pattern 之间存在交叉，看的角度不同

Designing Architecture

Design Strategy

Decomposition and iteration, Generation and test

分解和迭代，将设计放到一个更大的全局的系统中去测试

Attribute-Driven Design

Choose a part to design

Marshal all ASRs for that part

Create and test a design for that part

Inputs to and outputs of AD

详细的是 8 个步骤

1.confirm requirements

2.choose an element to decompose

3.identify ASRs

4.choose a design satisfying ASRs *********

5.instantiate elements & allocate responsibilities

……..

Documenting Architecture

Views and Beyond

Views:

Styles,patterns and views (区分三个概念）

Structural vies: module views(强调静态） cac(component-and-connector) views（强调动态) allocation views（周围环境）

Quality Views

Documenting views：

1.build stakeholder/view table

不同的 stakeholder 对于不同 views 的关注点，（重要性）

2.combine views

以 1 为依据合并 view，减少 View 的数量

3.prioritise & stage

优先级

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

将不同的视图联系起来

Documentation package：views and beyonds

Analysis:

Completeness external+internal

Consistency: internal

Name

Interface

Behavior
Interaction
Refinement

Compatibility: external

adhere to guidelines and constraints(style,reference architecture,architectural standard)

Correctness: external

fully realize a system specification
implementation realize the architecture

Scope of Analysis:

component and connector
subsystem and system
data exchange
different abstraction level
comparison?

type of analysis:

static
dynamic
scenario-driven

(can be both static and dynamic)

Evaluating Architecture

Why: late and over-budget; rework often;alleviate problem;对于 bug 早发现 早治疗；等等
Why early: enough time; inexpensive;assure quality;

**ATAM:Architecture Tradeoff Analysis Method

Stakeholders involved in ATAM

在 ATAM 不同阶段有不同的 Stakeholder 来参与

Inputs:架构文档
Outputs:evaluation plan{

presntation of the architecture
质量属性（场景）的优先级表
utility tree
risk and nonrisks
sensitivity points and tradeoff points

}

Phase 0: Partnership & preparation
Phase 1:Evaluation — 1

1.present ATAM

2.present business drivers

3.present architecture

4.identify architectural approaches

5.generate utility tree

6.analyse architectural approaches

Phase 2 :Evaluation-2

Phase 3:Follow up

Software Product Lines

基本思想：把一个产品分为两部分

Product = core assets + custom assets

Reusability and Modifiability 重用性、可修改性

产品线中的每个产品公用 core assets

Product Line Architecture

Reuse:find,understand,and use(invoke)

体现的是 core assets

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

考虑到不同产品的变化

6 种形式的变化，变化发生的位置（3 个层次））

变化发生的时间（5 个不同的时间点）

6*3*5=90 种不同变化形式

Architecture:variation points

结合、链接

SPL Practice Ares and Patterns

29 practice areas,12(22)patterns

29 个实践、 分组（每个不同的 Pattern 应对与不同的问题，12 个 pattern，其中衍伸出 10 个不同的变种，一共 22 个 patterns）

Model Driven Architecture

Model Driven Development(MDD)

Separate the specification of functionality and the specification of implementation

功能定义和功能实现分离开

追求高的 Reusability,Interoperability, and Portability

可移植性

互用性

Abstraction Levels

抽象层次

CIM(computation independent model)

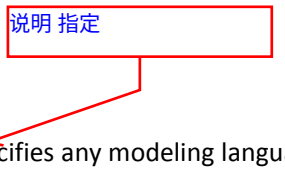
PIM(platform independent model) 平台无关

PSM(platform specific model) 平台相关

OMG Standards

MDA 遵循 OMG 标准

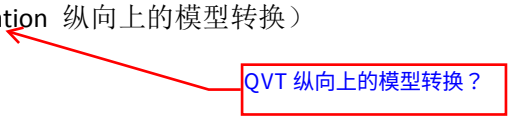
MDA是什么



UML,MOF(meta-object facility)specifies any modeling language（不同建模语言之间的横向转换，遵循相同标准）

XMI

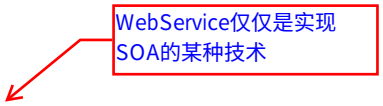
QVT(query-view-transifomation 纵向上的模型转换）



Service Oriented Architecture

Service Find-Bind_execute Paradigm

（三角关系，图）

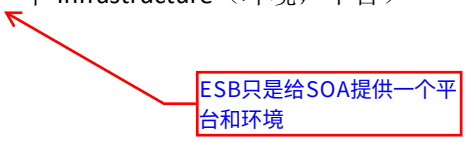


SOA vs. WebService

WebService 仅仅是实现 SOA 的某种技术，两者不等价

SOA vs.Enterprise Service Bus(ESB)

ESB 只是给 SOA 提供一个 Infrastructure（环境，平台）



实现

上-下 下-上

SA03 设计模式们

Pattern	Strength	Weakness
Layered	•便于开发：开发人员可以只关注整个结构中的某一层 •易于修改：修改时只要修改一层即可 •有利于标准化：分好层次易于展示达到统一标准 •利于复用：相似的逻辑不用重复写代码，都在逻辑层实现，其它层调用	•层数的增加带来系统的成本和复杂度的增加 •多层会减弱性能
Broker	•组件的拓展性好：server 可以动态的增删改 •耦合低：组合的方式，减少了依赖 •隔离：将与通信有关的代码和应用程序隔离 •简单：易于分工合作	•由于转发带来了效率的降低和延迟，客户端和服务器之间的联系可能存在瓶颈 •本身会带来错误 •增加了复杂度 •易导致安全攻击 •测试难度大
MVC	•拓展性良好：易于增加功能，不必改变程序结构 •部署快：由于模型独立于视图，也因此可以直接移植到新平台，只要修改视图和控制器 •速度快：界面与后台分离，提高了加载速度 •易于维护：同样是因为视图和逻辑分离，易于定位维护 •高重用性：不同的视图可以访问同一个逻辑代码	•复杂度高，不适用于简单的用户接口 •这种抽象不适用于部分用户接口
Pipe-filter	•降低了复杂度：一个过滤器只要实现单一的功能 •降低了耦合度：过滤器之间依赖很小 •灵活：可以通过组合的方式实现新的功能•	•不适用于交互系统 •大量独立的 filter 带来了计算的负担 •也不适合长时间运行的计算•
Client-server	•速度快：服务器通常采用高性能的机器，运算速度极快 •负荷轻：客户端可承担一部分的运算 •数据管理独立：客户无法通过客户端修改不应该被修改的数据，保护了	•服务器可能会造成性能瓶颈 •服务器一旦出错就整个都崩了 •把功能实现放在客户端还是服务器的决定很难，而一旦系统建成后修改的成本很大

	数据的稳定	
P2P	•摆脱了对服务器的依赖：即使服务器没有资源，可以从别的用户获得资源 •高扩展性：可以容易的增加、查找节点 •速度快：用户之间直接共享，不受带宽的限制	•管理安全、数据持久化、数据和服务的可用性、备份和恢复复杂度很高 •小型的 P2P 系统不能实现如性能和可用性等质量目标
Service-oriented	•敏捷：缩短了产品的开发时间 •数据统一：通常定义好了 SCHEMA 的格式，有着统一的规范	•SOA 系统的建立很复杂 •难以控制独立服务的演化 •服务会带来性能瓶颈，不能保证性能
Publish-subscribe	•松耦合：被观察者不必了解任何一个观察者，只要实现接口即可 •灵活：观察者可以轻松决定关注或取消关注，而改变的仅仅是观察者列表里的用户 ID	•增加了延迟，对于信息传送的范围和预测性有负面影响 •信息的有序化难以控制，而且不能保证信息一定发送出去
Share-data	•数据独立管理：由 data store 来管理数据的修改 •保证了数据的安全：数据的连接器都会被验证是否有读写资格	•可能带来性能瓶颈 •可能会一个部件失败导致整个系统失败 •生产者和消费者是数据会被捆绑
Multi-tier	•松耦合：各层之间相互独立 •易于维护：各组件相互独立，更换简单 •适应性强：通过增减层次可以适应不同的业务需求 •提高并发：分层带来了并发性的可能 •提高安全：数据与数据控制、程序、逻辑等分开，减少了异常访问数据的可能，提高了安全性	•潜在的成本和复杂度
Map-reduce	•简化开发：自动并行、负载均衡、灾备管理，简化了开发工作 •伸缩性好：每增加一台服务器，就能将差不多的计算能力接入到集群中	•大型数据集会带来额外负担 •数据需要划分为小子集，不然就没有并发优点了 • Operations that require multiple reduces are complex to orchestrate

数据共享

自动并行 负载均衡

软件生产线

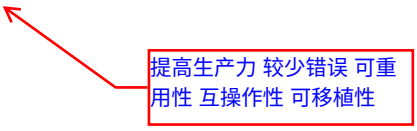
可重用的核心组件装配不同的产品

SA12 模型驱动架构 MDA

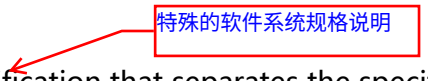
模型驱动开发 MDD:

提高生产力，减少错误，可重用性，互操作，移动性

可以得出特殊问题的通解



MDA:



Definition: “an approach to IT system specification that separates the specification of functionality from the specification of the implementation”

CIM,把计算独立开来，

- The three primary goals of MDA

– Portability

– Interoperability

– Reusability

SA14 SOA 面向服务的架构

SOA的优点

优点：

– Adapt applications to changing technologies.

使得应用程序不断适应新技术

– Integrate applications with other systems.

– Leverage existing investments in legacy applications.

将应用程序集成到其他系统上

将遗留的应用程序应用到现有的开发项目上

– Create a business process from existing services.

从当前的服务创建一个新的业务流程

特点：

– A service has a network-addressable interface.

每一项服务都有一个网络可寻址接口

– A service stresses interoperability 互用性

– A service may be dynamically discovered and used.

互用性

可以被动态的发现和使用

需要的支持技术：

-XML

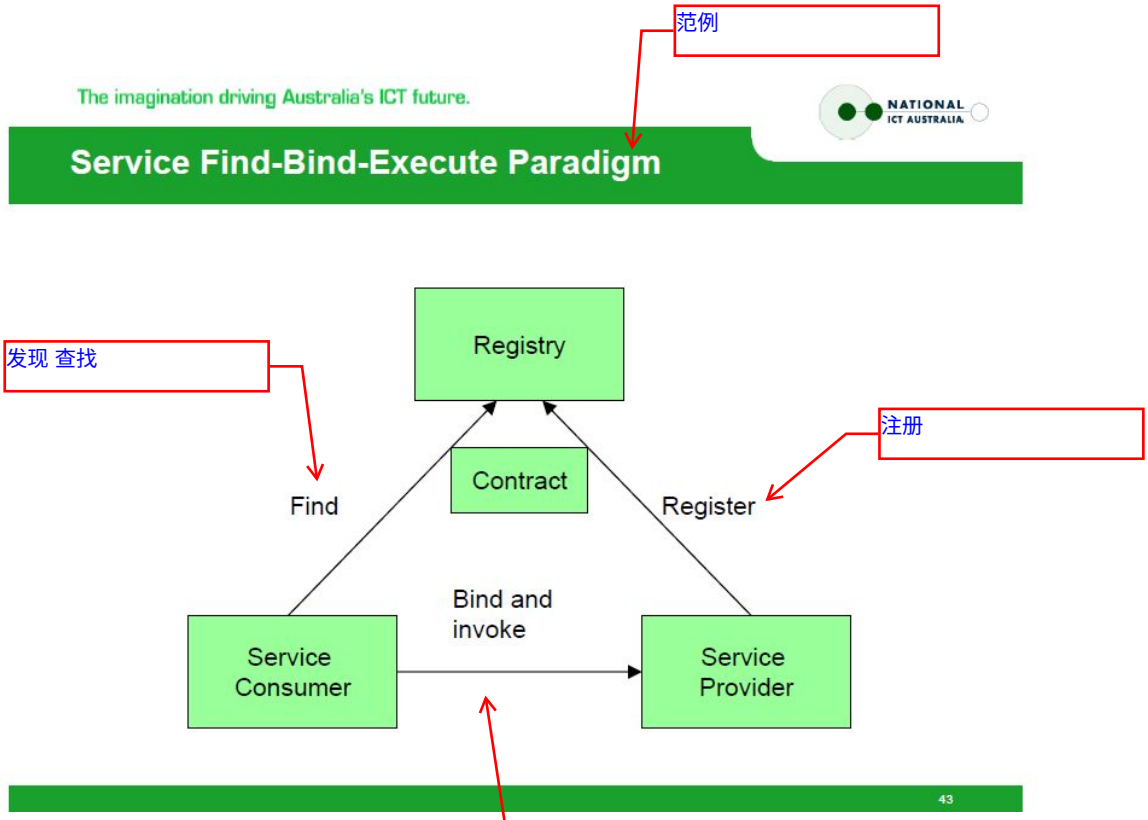
-网络服务

-服务总线

XML

网络服务

网络总线



服务提供者在注册处注册服务，使用者在注册处查找到服务后，向服务提供者调用

绑定和调用

一些澄清：

网络服务和SOA不一样，SOA是一个设计原则，网络服务是一种应用技术。

面向服务的应用的构建可以不使用网络服务（JNI RMI等）

网络服务带来了与平台无关的标准

网络服务允许多种技术的互用性

JNI 还有RMI也可以实现SOA

平台无关的标准

允许多种技术的互用性

面向服务的多种层次：

展示层 处理层，构成了服务层

逻辑层

持久层

处理层

处理层：BPEL语言 SOA提供商会提供工具

Service层和Business层

partnerLinks标签 variables标签 Flow Logic标签（receive, invoke, assign, pick, wait, throw, terminate, flow,sequence, switch dcope, fault handlers etc.）
