服务计算与服务生态系统 MOOC 习题答案

§ 服务、服务系统与面向服务的泛型

1. IT 服务和非 IT 服务都满足服务的基本要求，但他们也有区别。他们的区别包括：KPIs（关键绩效指标）不同、服务雇员不同、变化的步调不同。（✗）

解析：他们的区别在于 KPIs（关键绩效指标）不同、需求管理不同、变化的步调不同。

2. 铁三角发布 MSR7 限量版耳机。在普通 MSR7 耳机的基础上，限量版耳机听取了消费者的建议和需求，对模具和发声单元进行了大幅改进。4 月 1 日上市贩售，限量 10000 只，每只耳机均附带唯一编号的收藏证明。这个模式偏服务模式。（✗）

解析：为制造模式。

3. 服务模式（Service Mode）和制造模式（Manufacturing Mode）的最大差异在于：服务模式的产物是服务（Service），而制造模式的产物是货物（Goods）。服务是无形的、挥发的，并可能以消费者参与的方式定制化生产；而货物是有型的、可存储的，消费者不直接参与货物的生产过程。（✓）

4. 服务也可以以实物作为载体。如采用定制化的方式设计、裁剪和制作衣服。虽然衣服是有型的，但是满足了消费者的个性化定制需求，可以被认为是一个以服务为主的过程。（✓）

5. 可口可乐公司中的企业资源管理（ERP）偏服务模式。ERP 系统是一个服务系统。（✓）

6. 很多行业兼具制造模式和服务模式的特点，但随着时代的发展和竞争的要求，单纯的制造模式变得越来越少、服务模式的特点变得越来越显著。（✓）

7. 用来支持服务模式的 IT 系统被称为服务系统（Services System）。由于服务模式变的越来越普及，所有的软件系统均可以使用服务系统的概念进行抽象，进一步的，可以使用面向服务的方式进行分析和设计。（✗）

解析：不是所有的软件系统都是服务系统，也不是所有的服务系统都需要使用面向服务的方式进行分析和设计。

8. 要满足自治、开放、自描述、与实现无关的特点，无论是本地构件还是网络构件，都能够被认为是服务。（✗）

解析：服务必须是网络构件。

9. 面向对象的分析设计过程中间，如果将系统模块化，并采用统一的方式定义模块和模块之间的接口，如果这些接口是标准的，我们就可以把它们抽象为构件。与对象相比，构件的粒度较大，利于复用。（✗）

解析：构件是模块化的、可部署、可替换的软件系统组成部分。

10. 引入面向服务的泛型后，一方面，复用的范围从单个服务系统扩大到了多个服务系统；另一方面，软件的开发任务被分为服务的生产和应用的开发。这使得各个服务系统的开发工作能够有效利用现有的 IT 资源和人力资源。（✓）

§ 服务生态系统与面向服务的计算

1. 服务组合由多个装配在一起服务所构成，用以提供对业务任务或过程进行实现的功能。如果服务组合

能够进一步的被封装为服务，可以认为服务组合是服务的一种实现方式。（✓）

2. 只要在服务库存中存在，无论是应用服务、业务服务还是编排服务，都可以作为子服务被服务组合装配。（✓）

3. 根据是否直接满足服务消费者的需求，可以将服务生态系统（Services ecosystem）中的服务分为垂直（Vertical）服务和水平（Horizontal）服务。因此，在进行服务系统构建的时候，可以事先将服务分划为这两种类型，并按照其特点，应用不同的设计原则，分别进行分析和设计。（✗）

解析：分划不是排他性的，如果一个服务既能够被消费者直接调用，也能为其他服务所调用，那么该服务可以同时是垂直服务和水平服务。

4. 由于缺乏统一的标准化组织和标准体系，目前尚没有出现成熟的面向服务编程语言，但是随着面向服务的泛型和 Web Service 标准的发展，面向服务编程语言有望在将来出现，并提供面向服务实现的完全支持。（✗）

解析：面向服务着眼于对服务系统进行平台/技术/语言无关的分析和设计，不需要提供面向服务的编程语言来提供面向服务的实现。

5. 面向服务的目标主要包括技术上的：灵活性、扩展性、鲁棒性、重用性和效率、业务需求的满足。商业上的：组织灵活性、固有的互操作性、供应商多样化选择、联盟、业务和技术一致性、削减 IT 投入、提高投资回报率。（✓）

6. 从作用域角度出发，面向服务的计算中的服务作用域往往基于一个服务生态系统；在面向对象的计算中，对象的作用域往往基于一个软件系统。（✓）

7. 从耦合的角度出发，在面向对象的计算中，如果我们按照功能仔细划分，达成模块和模块之间的“相对无关性”，我们可以达成与面向服务计算中类似的松耦合特性。（✗）

解析：在面向服务计算中，松耦合还体现在运行时态动态绑定。即设计与实现分离、一个设计可以由多个实现者加以提供。

8. 服务组合提供了复用性和灵活性。但是由于特定子服务可以同时参与到多个业务流程，“瓶颈效应”和“单点失效”的问题使得以服务组合方式实现的业务功能可靠性差于采用面向对象实现的类似功能。（✗）

解析：“瓶颈效应”和“单点失效”问题可以通过提供服务的多个备选实现加以解决。

9. 在面向对象的计算中，代码复用通过类成员的继承和库函数加以实现。面向服务的计算中，代码在服务层次复用。（✓）

10. 由于服务是一个网络构件，采用面向服务的计算所构造出的服务系统，天然的是一个网络应用。（✓）

§ 面向服务的架构和 Web Service

1. 服务系统中的三要素包括：服务提供者、服务消费者和服务注册（Service Registry）。其中，服务注册通过支持服务的发布和查找，实现服务提供者和服务消费者之间的松耦合，从而实现服务系统灵活、可动态配置的特点。（✗）

解析：服务系统中没有关于服务注册的要求。

2. 由是否拥有中心协调者作为判断，服务组合（Composition）的方法可以分为编排（Orchestration）和

编导（Choreograph）。从能力上来说，它们各有不同，在实际使用时需要根据业务场景进行选择。（✗）
解析：从能力上来说他们是等价的。

3. 对应服务生态系统，SOA-RA（SOA 参考架构）中，我们使用垂直层来分层设计并实现直接满足消费者需求的垂直服务；我们使用水平层来分层设计并实现用以实现灵活、复用的水平服务。（✗）
解析：SOA-RA 中，水平层用以实现功能，垂直层用于提供支持和 QoS。他们和服务生态系统中的垂直服务和水平服务的分划方式没有关系。

4. 服务簇由服务层进行定义，它是一类从概念上服务于同一个业务功能的服务集合。他们可以由不同的服务提供者所发布，并在功能上可以相互替代。而业务过程层中，无论采用组合还是分解的方式，实质上都是面向服务簇来进行的。（✓）

5. 面向服务的架构并没有限定实现技术，常见的实现技术包括但不限于：Web Service、分布式对象、构件、RESTful Services、ESB 等。（✓）

§ XML 及相关协议

1. 信息交换的范围可以分为应用内部（Intra-Application）、应用之间（Inter-Application）、系统之间（Inter-System）和公司之间（Inter-Company）。XML 可以用以进行上述任意类型的信息交换。（✓）

2. XML 是一种以树结构交换文本信息的信息交换标准，无法被用以交换二进制数据。（✗）
解析：XML 可以用来交换二进制数据。

3. XML 既可以面向数据，也可以面向展现。在面向展现时，XML 可以使用预定好的元素和属性表达 Web 页面，在 Web 浏览器打开该页面时，如果发生结束标签缺失或标签大小写不匹配等问题时，与 HTML 类似，Web 浏览器可以自动纠正这些错误。（✗）
解析：XML 是面向数据的，他的语义和展现无关，亦没有任何面向页面的预定义标签，任何 XML 均应满足格式良好的限定条件。

4. 名称空间（Namespace），采用前缀（Prefix）+ 本地部分（Local part）表达元素和属性名称，从而解决命名冲突和全球化的问题。通过 import 机制可以把隶属在不同名称空间（前缀对应不同 URI）中的元素（属性）组装为一个 XML 文档。（✗）
解析：无需额外机制，即可通过 QName 在单一 XML 文档中融合来源于不同 Namespace 的元素或者属性。

5. 只要满足 5 + 1 条基本原则（单根原则、元素（Element）标签原则、元素嵌套原则、元素原则、属性原则和 XML 声明原则），XML 文档就可以被称为良好格式化（Well-formed）。在此基础上，如有需要，还可以使用 Schema 脚本等对 XML 文档进行数据类型检查，这个过程称为验证（Validation）。（✓）

§ Web Service 核心

1. SOAP 定义了在分布式异构环境中进行信息交换的消息框架和单向、无状态的通信范例。它不是一个网络传输协议，因此，必须通过绑定机制将 SOAP 消息与实际的网络消息格式进行关联。（✓）

2. 在我们使用 SOAP 来传递 XML 信息时，由上层的应用来解释 SOAP 所携带 XML 消息的语义语法。虽不是必须，一般来说，必选的核心的相关信息被放置在消息体（Body），可选的辅助性的相关信息被放

置在消息头（Header） （✓）

3. WSDL 以 XML 的方式描述了一个服务的功能、调用地点和调用方式。WSDL 中所定义的操作（Operation）按照风格可以分为面向文档（Document-Oriented）和面向过程（Procedure-Oriented）。其主要区别在于所使用的 SOAP 绑定风格不同：前者使用 Message-exchange 风格的 SOAP 消息，而后者使用 RPC 风格的 SOAP 消息。 （✗）

解析： 主要的区别在于：是接受一个用 XML Schema 约定的 XML 文档作为输入输出，还是采用远端的过程接口加上映射方式决定输入输出。

4. WSDL 模型被分为抽象部分（Abstract Section）和具体部分（Concrete Section）。抽象部分包括 types、interface 元素；具体部分包括 binding、service 元素。抽象部分和具体部分可以由不同的设计人员/团队加以定义，并加以组装。拥有相同抽象部分的服务被称为服务簇（Services Cluster），它们拥有相同/相似的服务能力，在满足其他条件（如非功能性需求）的前提下，可以相互替代。 （✓）

5. WSDL 中的 binding 元素和 SOAP 中的绑定机制是类似的。因此，在使用 WSDL+SOAP 对服务进行描述时，binding 元素可以省去。只有采用 WSDL+HTTP 对服务进行描述时，binding 元素才是必须的。 （✗）

解析： WSDL 中的 binding 元素和 SOAP 中的绑定机制解决不同的问题。即使是使用 WSDL+SOAP，binding 元素也是必须的。

§ Web Service 扩展

1. UDDI 存放的信息可以分为白页（White Pages）、黄页（Yellow Pages）和绿页（Green Pages）。在使用 UDDI 发布 Web Services 时，使用白页记录 WSDL 的抽象部分；使用绿页记录 WSDL 的具体部分；使用黄页记录服务合约中的其他信息。 （✗）

解析： 白页存放基本信息、黄页存放分类信息、绿页存放技术信息。

2. BPEL 是一种使用编排风格实现服务组合的规范。一方面 BPEL 要求组合成员在 WSDL 中以 partnerLinkType 元素说明它在组合中的作用和地位；另一方面使用 BPEL 文档描述该组合的逻辑流程。BPEL 公布了服务实现的细节。 （✓）

3. WS-Addressing 被用以完成服务实现中的寻址和消息路由。WS-Security 被用以实现 Web Service 安全相关的非功能性需求。 （✓）

4. WSRF 被用以完成服务相关资源的定义和管理。无论是有状态服务（Stateful）还是无状态服务（Stateless）都可以使用它来定义和管理资源。 （✗）

解析： 一般认为，携有资源的服务为有状态服务。

5. WS-Coordination 被用以支持服务之间的复杂协作，它定义了协作服务和协作上下文，只能使用原子事务（Atomic Transactions）和业务活动（Business Activity）两种协议分别完成短时间协作和长时间协作。 （✗）

解析： 亦可自己扩展相关的协作协议。

§ 服务生态系统的构建

1. 面向服务分析的过程主要包括：定义流程自动化需求，识别现有的自动化系统，服务候选建模。其中，服务候选建模是面向服务分析的主要步骤，以建立服务操作候选，并将其合理分组到服务候选为目的。（✓）

2. 在服务建模的过程中，需要避免服务候选之间的业务逻辑相互重叠。这一原则要求在任何情况下同一个业务逻辑单元仅能用一个服务操作候选进行封装。（✗）

解析：大多数服务候选有这样的需求，但是编排和组合服务有意设计为其他服务的组合。封装遗留系统的服务也可能违反这一要求。为了安全性和粒度需求，有时候我们也会采用冗余接口的方式加以设计。

3. 根据业务分析方法及其自身特点，在对服务进行设计时，可以将服务分为应用服务、以实体为核心的业务服务、以任务为核心的业务服务和编排服务。各种不同类型的服务在面向服务原则的应用上各有不同，因此需要在分析和设计时区别对待，为每种类型的服务分别制定分析和设计标准。（✓）

4. 在提炼和应用面向服务原则的过程中，不应局限于产生该服务的初始流程和上下文。一方面，需要综合考虑服务生态系统中所有可能复用该服务的流程和上下文；另一方面，应当充分考虑服务的进一步演化和全新流程的构建。（✓）

5. 以任务为核心的业务可以使用服务组合加以实现，而以实体为核心的业务服务不能使用服务组合加以实现。（✗）

解析：两者均可。

§ 服务设计原则

1. 标准化服务合约原则要求制定服务生态系统中的服务合约标准。越是标准化的服务合约越利于服务的可理解性和可复用性。此外，标准化的服务合约还倾向于产生标准粒度的调用数据，从而避免不必要的数据转换，减少服务实现的复杂度。（✓）

2. 服务松散耦合原则允许的积极耦合包括：逻辑—合约耦合、消费者—合约耦合。（✓）

3. 服务的抽象原则要求服务合约对外隐藏非必要信息。这些信息抽象包括：技术信息抽象、功能抽象、程序逻辑抽象和服务质量抽象。越是抽象的服务合约，越倾向于较高的可复用性。（✗）

解析：完全抽象的服务合约可能需要额外的其他元信息交换渠道，反而不利于服务的可复用性。

4. 服务的可复用性原则要求将服务生态系统中的所有服务均设计为无关服务（Agnostic service），从而确保最大程度的可复用性。（✗）

解析：不是所有的服务均是无关服务，垂直服务在大多数情况下是和消费者有关的。

5. 服务自治原则，将运行时自治的情况分为共享自治、服务逻辑自治和完全自治。虽然高自治的服务倾向于高性能、高可靠性和可预测性，但是由于高自治服务往往需要较多的软硬件资源，在设计服务的时候需要慎重考虑。（✓）

6. 服务的无状态性原则，鼓励利用调用消息或状态数据库，在必要时卸载/还原服务的状态信息，将服务的实例转换为无状态/少状态，从而减少资源的消耗，提升服务的可扩展性。（✓）

7. 服务的可发现性原则要求服务不仅能够被发现，而且要求服务合约所提供的信息使得服务的潜在调用

者能完全了解该服务的目标、能力和限制等。良好的可发现性是服务具备较高可复用性的前提条件。（✓）

8. 服务的可组合性原则，要求服务应尽量设计为使用服务组合来加以实现。作为组合控制器（Composition Controller）的服务调用的组合成员（Composition Member）越多，该服务组合越复杂，担任组合控制器的服务的可组合型越好。（✗）

解析：服务的可组合性主要面对非功能性需求，由其他服务设计原则的综合表现进行评级。