实验九

实验目的：

1. 深入理解UML
2. 了解计算机学科中的逻辑
3. 学习对比软件体系结构设计GB和IEEE最新SAD (Software Architecture Document)的标准
4. 研究经典软件体系结构案例
5. 完成自己项目的SRS

实验内容：

1. 阅读“The Unified Modeling Language Reference Manual”，进一步学习UML知识，理解如何应用UML对系统进行建模

UML是一种用于软件系统建模的标准化语言，提供了一套丰富的图形符号和语法规则，用于描述系统的结构、行为和交互。下面是使用UML建模的几个步骤：

1、明确系统建模的目的和范围，了解要建模的系统是什么，并确定需要关注的方面，如系统结构、行为、交互等。

2、据建模目的和需求，选择适当的UML图表来描述系统的不同方面。常用的UML图表包括用例图、类图、时序图、活动图、状态图等。

3、使用用例图描述系统的功能需求，识别系统的参与者和用例，以及它们之间的关系。

4、使用类图描述系统的静态结构，识别系统中的类和它们之间的关系。

5、使用行为图描述系统的动态行为。时序图展示对象之间的消息传递和交互顺序，活动图展示系统的业务流程和操作步骤，状态图展示对象的状态和状态转换。

6、使用构件图描述系统的组成部分和构件之间的关系。构件图帮助识别系统的模块、库、组件等，以及它们之间的依赖关系和接口。

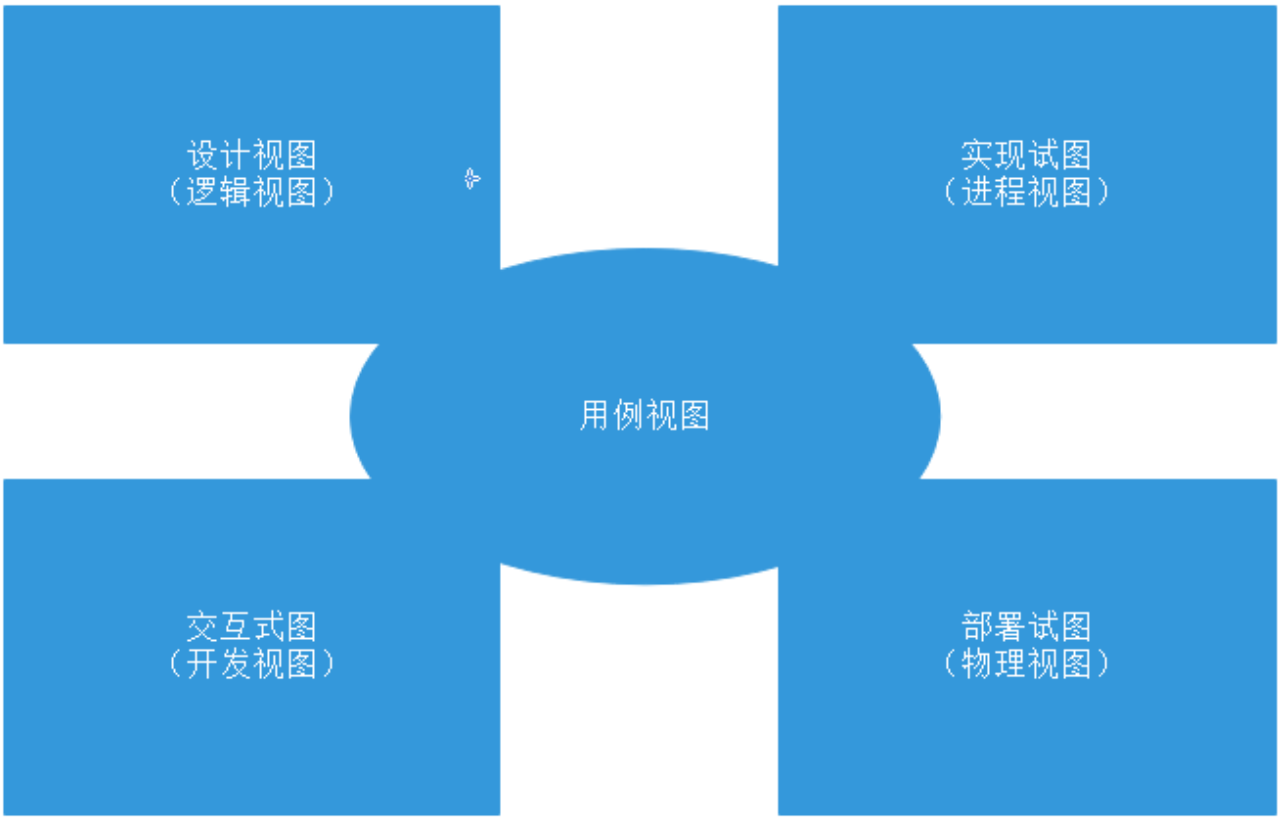
7、使用部署图描述系统的物理部署结构，包括硬件设备、软件组件、网络连接等。部署图帮助理解系统在不同物理节点上的部署情况和通信方式。

8、对建立的UML模型进行验证，确保模型与系统需求的一致性和正确性。根据反馈和需求变更，对模型进行迭代和改进。

9、将建立的UML模型文档化，并与团队成员或利益相关者进行沟通和共享。模型文档应清晰描述系统的结构、行为和交互，以便他人理解和使用

详细分析：

Kruchten 提出了一个"4+1"视图模型，从5个不同的视角包括包括逻辑试图、进程视图、物理视图、开发视图、场景视图来描述软件体系结构。每一个视图只关心系统的一个侧面，5个试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：逻辑视图主要是用来描述系统的功能需求，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、功能分解与功能分析，这些主要来自问题领域（Problem Definition)。 在面向对象技术中，通过抽象、封装、继承,可以用对象模型来代表逻辑视图，可以用类图（Class Diagram）来描述逻辑视图。

开发视图：开发视图主要用来描述软件模块的组织与管理（通过程序库或子系统）。服务于软件编程人员， 方便后续的设计与实现。它通过系统输入输出关系的模型图和子系统图来描述。要考虑软件的内部需求：开发的难易程度、重用的可能性，通用性，局限性等等。开发视图的风格通常是层次结构，层次越低，通用性越好（底层库：Java SDK，图像处理软件包）。

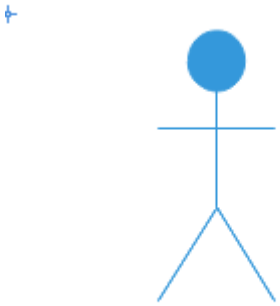
进程视图：进程试图侧重系统的运行特性，关注非功能性的需求（性能，可用性）。服务于系统集成人员，方便后续性能测试。强调并发性、分布性、集成性、鲁棒性（容错）、可扩充性、吞吐量等。定义逻辑视图中的各个类的具体操作是在哪一个线程（Thread）中被执行。

物理视图：物理试图主要描述硬件配置。服务于系统工程人员，解决系统的拓扑结构、系统安装、通信等问题。主要考虑如何把软件映射到硬件上，也要考虑系统性能、规模、可靠性等。可以与进程视图一起映射。

用例图、参与者、用例的基本概念：

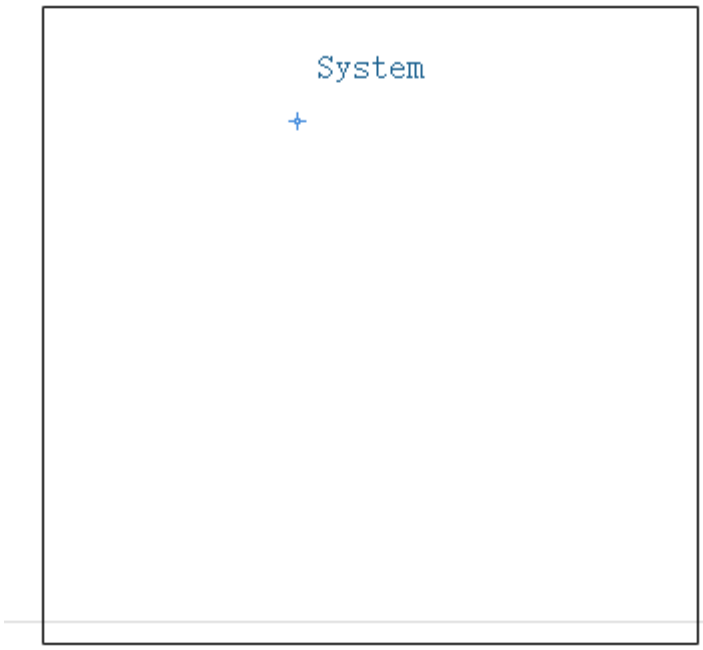
用例图的作用：是需求分析中的产物，主要作用就是描述参与者和用例之间的关系，帮助开发人员了解系统的功能。用例图的构成元素有四个：参与者（角色）、用例、系统边界和元素。

参与者：在系统外部并且直接与程序或者系统接触的人、系统、子系统火类的外部实体的抽象，在UML中使用一个小人来进行表示



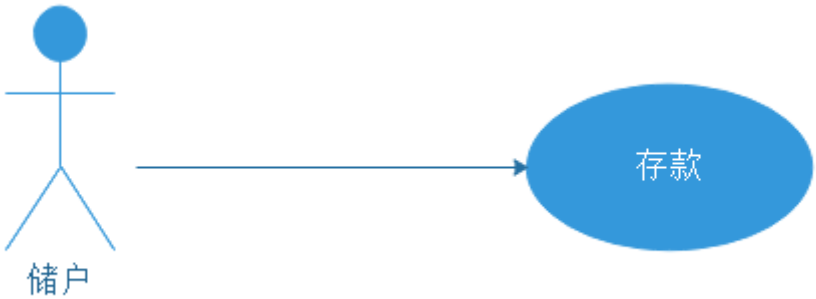
用例：用例即使外部可见的系统功能，对系统提供的服务来进行描述，使用椭圆来进行表示

系统边界：指系统与系统之间的界限，使用方形框+系统名称来表示



元素间的关系有四个：关联、泛化、包含

关联：参与者与用例之间的关系，，任何一方都可以接收或者发送信息（箭头指向信息接受方）



泛化：是类元一般描述和具体描述之间的关系，具体描述建立在一般描述的基础之上，并且对其进行了扩展。具体描述完全拥有一般描述的特性、成员和关系，并且包含补充信息（类似于Java这继承关系）。

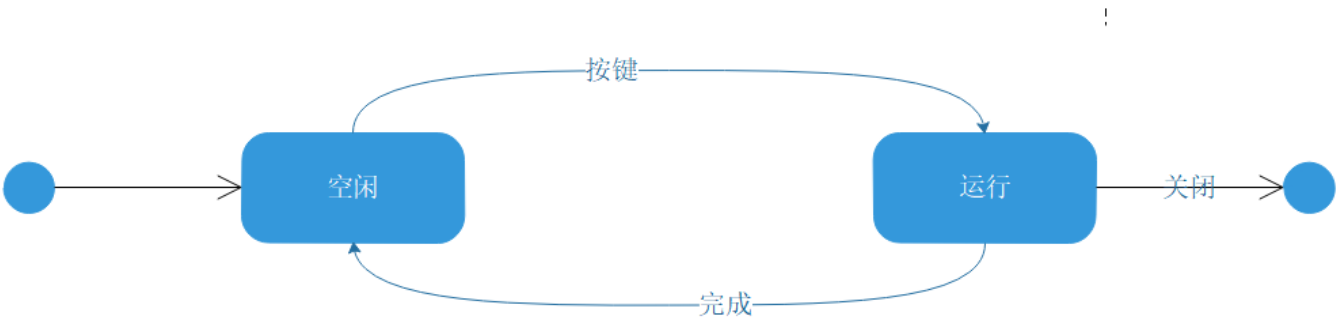
依赖：类A的实现需要使用到B，这就是依赖，这种使用关系是具有偶然性、临时性、非常弱的，并且B类的的变化会影响到A，则A与B就是存在依赖关系，依赖关系是弱的关联关系，例如：人们依赖计算机去做软件的开发。

箭头家虚线不仅仅可以代表依赖关系，还可以代表扩展、包含关系，扩展就是对用例功能的延申，包含就是将一个复杂的功能分解成数个小的功能

状态图：

状态图就是开关，是描述状态变化的图形。描述了一个对象状态与状态的转变并且象所经历的状态序列，引起状态转移的事件，以及因状态转移而伴随的动作状态图主要用于描述一个对象在其生存期间的动态行为，表现为一个对一般可以用状态机对一个对象的生命周期建模，状态图用于显示状态机，重点在于描述状态图的控制流。

一个机器的状态图如下：



2. 浏览“LOGIC IN COMPUTER SCIENCE--Modelling and Reasoning about Systems”，了解常用逻辑及其在计算机学科中的应用

1、命题逻辑（Propositional Logic）：命题逻辑是一种用于研究命题（逻辑语句）之间关系的形式系统。它使用逻辑运算符（如与、或、非）来组合命题，并确定它们的真值。命题逻辑在计算机科学中的应用广泛，包括：

- 布尔代数和逻辑电路设计：命题逻辑提供了运算规则和代数性质，与布尔代数密切相关。它在逻辑电路设计中被广泛应用，用于构建和分析数字电路和逻辑门电路。
- 程序验证和形式化验证：命题逻辑用于描述和验证程序的正确性。通过将程序的逻辑规范转化为命题逻辑表达式，可以应用形式化验证技术进行自动化验证，确保程序满足预期的行为。
- 推理引擎和专家系统：命题逻辑提供了推理规则和推理机制，用于构建推理引擎和专家系统。这些系统利用命题逻辑的推理能力，根据已知事实和规则，生成新的推论和结论。
- 自然语言处理：在自然语言处理中，命题逻辑被用于对句子的逻辑结构进行分析和推理。通过将自然语言句子转化为命题逻辑形式，可以进行语义解析、推理和问答系统的构建。
- 知识表示和语义网：命题逻辑用于表示和推理关于世界的知识。它在知识表示和语义网的构建中发挥重要作用，支持知识图谱的建立和推理。
- 数据库查询语言：命题逻辑提供了查询语言中的逻辑基础。关系数据库中的SQL查询语言使用命题逻辑的语义和推理规则来操作和查询数据库中的数据。

2、一阶逻辑（First-Order Logic）：一阶逻辑引入了个体和谓词的概念，允许对个体进行量化，并进行更复杂的逻辑推理。一阶逻辑在计算机科学中的应用包括：

- 人工智能和知识表示：一阶逻辑被广泛用于表示和推理关于世界的知识。它为人工智能系统提供了一种形式化的表示方式，用于描述实体、关系、属性和规则。专家系统、语义网络和语义推理系统都依赖于一阶逻辑来表示和推理知识。
- 形式化验证和模型检查：一阶逻辑用于规约系统的性质和行为，以进行形式化验证和模型检查。通过将系统的行为和约束转化为一阶逻辑表达式，可以应用自动化验证技术来验证系统是否满足指定的规范和属性。
- 自动推理和定理证明：一阶逻辑提供了推理规则和推理机制，可用于自动推理和定理证明。它被用于构建自动推理系统和定理证明器，以推导新的逻辑结论和证明给定的数学定理。
- 数据库查询和信息检索：一阶逻辑在数据库查询语言和信息检索中发挥重要作用。它可以用于描述查询条件和约束，以及推理和推断查询结果。关系数据库的查询语言如SQL使用一阶逻辑的语义和规则来操作和查询数据库中的数据。
- 自然语言处理：在自然语言处理中，一阶逻辑被用于对自然语言句子的语义进行解析和推理。通过将句子转化为一阶逻辑形式，可以进行语义解析、语义角色标注和语义推理等任务。
- 软件工程和形式化方法：一阶逻辑在软件工程中被用于形式化描述软件系统的行为和属性。它可以用于规约系统的规范和约束，以进行形式化验证、模型检查和软件测试。

3、模态逻辑（Modal Logic）：模态逻辑用于描述命题在不同的世界或情境中的可知性、必然性、可能性等性质。在计算机科学中的应用包括：

- 形式化系统规约和验证：模态逻辑用于形式化描述系统的规范和约束条件，以验证系统的正确性和安全性。它可以描述系统中的可知性、必然性、可能性等性质，帮助分析和验证系统的行为和属性。
- 安全性验证：模态逻辑可用于描述和分析系统的安全属性和策略。它可以帮助识别和分析系统中的安全漏洞、访问控制规则和安全策略，以确保系统对不同的攻击和威胁具有适当的响应。
- 知识表示与推理：模态逻辑用于表示和推理关于世界的知识。它可以描述知识的可知性、可能性和必然性，以支持智能代理的决策和推理过程。
- 模型检查：模态逻辑在模型检查中发挥重要作用。模型检查是一种自动化验证技术，用于对系统的状态转换模型进行分析和验证。模态逻辑提供了描述系统状态和转换之间时序关系的能力，用于验证系统是否满足特定的时序性质。
- 软件工程和形式化方法：模态逻辑在软件工程中被用于形式化描述软件系统的行为和属性。它可以描述系统中的时序性质、状态转换和约束条件，以进行形式化验证和建模。
- 认知建模和人机交互：模态逻辑用于认知建模和人机交互领域。它可以描述用户的知识、信念、意图和目标，并支持智能系统与用户之间的推理、对话和决策。

4、时序逻辑（Temporal Logic）：时序逻辑用于描述系统中事件或状态之间的时序关系，包括顺序、并发、前后关系等。在计算机科学中的应用包括：

- 并发系统建模与验证：时序逻辑用于建模和验证并发系统中的时序性质，例如同步、死锁和竞争条件等。它可以描述系统中的事件顺序、时间约束和时序关系，以验证系统在并发环境下的正确性和安全性。
- 硬件验证：时序逻辑在硬件验证中被广泛应用。它可用于描述和验证电子系统中的时序行为，例如时钟同步、状态转换和数据通路。通过时序逻辑的形式化规约和验证，可以发现和解决硬件设计中的逻辑错误和时序问题。
- 实时系统分析与调度：时序逻辑可用于实时系统的分析和调度。它可以描述系统中的时间约束和时序要求，并帮助分析系统的实时性能和可调度性。通过时序逻辑的建模和分析，可以优化实时系统的任务调度和资源分配。
- 通信协议验证：时序逻辑在通信协议验证中发挥重要作用。它可用于描述和验证协议中的时序约束和通信行为，以确保协议的正确性和安全性。通过时序逻辑的规约和验证，可以发现和解决通信协议中的死锁、冲突和数据丢失等问题。
- 软件规约：时序逻辑可用于规约软件系统的行为和属性，以进行形式化验证和测试。它可以描述系统中的事件序列、状态转换和时序关系，并支持对软件系统的时序性质进行验证和推理。

5、类型理论（Type Theory）：类型理论是一种逻辑体系，用于描述计算对象的类型和类型之间的关系。它在计算机科学中的应用包括：

- 函数式编程语言：类型理论为函数式编程语言提供了类型安全性和推断能力，帮助开发人员编写可靠和可维护的代码。
- 证明助理：类型理论在交互式证明助理中发挥重要作用，支持形式化证明的正确性和可靠性。

- 形式化验证：类型理论可用于形式化验证系统的属性和规约，以确保系统满足特定的类型约束和规则。

除了以上提到的逻辑，还存在其他一些在计算机科学中有应用的逻辑。例如：

6、高阶逻辑（Higher-Order Logic）：高阶逻辑扩展了一阶逻辑，允许量化谓词或函数。它在形式化验证、形式化语义、形式化数学等领域中得到应用。

7、非经典逻辑（Non-Classical Logic）：非经典逻辑是对经典逻辑的扩展或修正，以处理更复杂的情境和推理形式。例如，模糊逻辑处理模糊性信息，多值逻辑处理多值推理，弱三段论逻辑处理不完全信息等。

8、知识表示逻辑（Knowledge Representation Logic）：知识表示逻辑是用于表示和推理知识的逻辑系统。它在人工智能、专家系统、自然语言处理等领域中用于表示和推理关于世界的知识。

9、归纳逻辑编程（Inductive Logic Programming）：归纳逻辑编程是逻辑编程和机器学习的结合，用于从观察数据中归纳出逻辑程序。它在机器学习、数据挖掘、概念学习等领域有应用。

3. 分工协作，参考国标“13 - 软件(结构)设计说明(SDD)”等资料，对比参考SAD最新标准IEEE-42010.pdf，针对自己的项目设计SAD初稿。

1、 引言部分需要准确描述该软件(结构)设计说明文档的标识、系统概述、文档概述等重要信息。

2、 在引用文件中应列出本文档所引用的所有相关文件，以保证本文档的准确性和可靠性。

3、 CSCI级设计决策部分需要详细记录与计算机软件组件相互作用相关的决策、问题和限制。

4、 CSCI体系结构设计部分需要明确定义整个博客网站系统的层次结构，并划分程序(模块)并说明程序(模块)之间的关系。

5、 全局数据结构说明部分需要清晰地描述常量、变量及各种数据结构在该系统中的使用方式。

6、 CSCI部件部分需要介绍软件(结构)设计中所设计的各项具体部分，并详细描述其功能和实现。

7、 执行概念需要描述该系统的整体执行流程，从而帮助用户更好地了解系统的功能和特点。

8、 接口设计部分需要明确各个组件之间的接口标识和接口图，以确保各个部分能够有效地协同工作。

9、 在CSCI详细设计中，需要具体说明系统的各个设计方面。

10、需求的可追踪性是本文档整个设计过程中的重要部分。需要确保每一个需求都有明确的追踪方法，并能够有力地支撑整个软件开发过程。

4. 分工协作，学习、检索研究经典软件体系结构案例。

On-the-Criteria-To-Be-Used-in-Decomposing-Systems-into-Modules.pdf

http://www.cs.cmu.edu/~ModProb/index.html

1.客户端-服务器体系结构：一种常用的体系结构，其中客户端应用程序通过网络与服务器进行通信。客户端应用程序通常是轻量级的，只负责处理用户界面和显示数据。服务器则负责处理业务逻辑和数据存储。适用于需要中心化数据管理和处理的应用程序，例如Web应用程序和企业应用程序。

2.分层体系结构：用于将应用程序分为不同的层。每个层都有自己的职责和功能，例如表示层、业务逻辑层和数据访问层。这种体系结构使得应用程序更易于维护和扩展。适用于需要清晰的代码组织和模块化，易于维护和扩展的大型企业应用程序。

3.Model-View-Controller (MVC)体系结构：MVC是一种常用的体系结构，用于开发Web应用程序。模型(M)负责处理应用程序的数据和业务逻辑，视图(V)负责显示数据，控制器(C)则负责协调模型和视图之间的通信。适用于开发Web应用程序，具有清晰的分层结构，易于维护和扩展。

4.微服务架构：微服务架构是一种分布式体系结构，其中应用程序被分解为多个小的、自治的服务。每个服务都有自己的数据存储和业务逻辑，可以独立部署和扩展。这种体系结构具有高可用性和可扩展性。适用于需要高可用性和可扩展性的应用程序，例如电子商务网站和在线游戏。

5.事件驱动体系结构：这种体系结构基于事件和消息传递，应用程序通过发布和订阅事件来进行通信。这种体系结构具有松耦合性和可伸缩性。适用于需要处理大量事件和消息的应用程序，例如物联网应用程序和实时数据分析应用程序。

5. 完成软件需求规格说明SRS