极客大学架构师训练营

李智慧

《架构师训练营》

李智慧

长期从事大型网站架构、大数据的研发工作，Apache Spark 代码贡献者，曾担任阿里巴巴技术专家、Intel 亚太研发中心架构师、WiFi 万能钥匙 CTO ，著有畅销书《大型网站技术架构：核心原理与案例分析》。

超过 6 年的线下咨询、企业架构内训经验，曾经作为外聘教师为浙江大学硕士研究生开展《面向对象高级编程》课程。出品了极客时间专栏《从 0 开始学大数据》、《后端技术面试 38 讲》。

大厂 Offer 该如何获取？

后端架构师 / 25k-50k

职位诱惑：

六险一金,餐补

职位描述：

职位职责：
1、负责字节跳动企业级SaaS应用等后台业务的产品调研/讨论以及整体的架构设计；
2、承担业务重点、业内难点的技术攻坚，主导核心组件/服务的编码和上线；
3、分析和深入发掘现有系统的不足，定位系统瓶颈，提高系统性能/稳定性以及业务扩展性；
4、主导跨部门协作和复杂功能的调研、设计、协调、实施和落地。
职位要求：
1、本科及以上学历，7年及以上工作经验；
2、具备丰富的架构设计经验，能够准确、全面的理解业务，并根据业务发展设计合理的架构方案；
3、具备海量数据和大规模分布式系统的设计和开发经验
4、良好的产品意识，能够做到技术和产品相结合，从设计到实现始终对齐业内一流产品水准；
5、具备良好的沟通能力、组织能力及团队协作精神；
6、负责过多条业务线或整个产品线的业务架构工作，组织过中等以上规模项目者优先；
7、对多种数据库中间件、消息中间件及其他大规模分布式系统的基础架构组件有深入理解者优先；
8、熟悉公有云，私有云，虚拟化，容器化部署者优先。

工作地址

深圳 - 南山区 - 南海大道2163号来福士广场17层　　查看地图

Java架构师 / 40k-60k

职位诱惑：

免费三餐 长期激励

职位描述：

工作职责:
1．负责业务中台的建设规划、业务模型抽象、技术架构设计， 组织研发工作；
2．带领团队攻克高并发、高可用，业务模型复杂等挑战，持续打造、运营中台项目，沉淀技术架构能力和最佳方案；
3．深入思考研发过程中的各项问题，指导高级开发人员的开发工作，促进团队工作效率和研发质量的提升；
4．以自身良好的项目管理与协调沟通能力，负责跨团队的重点项目的推进工作；
任职资格:
1．统招本科及以上学历，具备5年以上软件架构、设计和开发经验；
2．擅长领域模型和敏捷开发的思想和方法论，具备微服务架构，设计和开发的项目经验；
3．擅长消息中间件、分布式事务等互联网应用架构，具备主导设计互联网应用的项目经验；
4．对Spring、MyBatis等常用开源框架应用经验丰富，对框架本身的体系有较为深厚的理解和应用经验；
5．对技术有强烈的兴趣，喜欢钻研，具有良好的学习能力，沟通技能，团队合作能力；
6．具备良好的沟通和协作技能、拿结果能力，能够有效的推进工作落地
7．有互联网业务背景，有大规模、高并发访问的Web应用开发经验。

工作地址

北京 - 海淀区 - 福道大厦 查看地图

Java架构师 / 30k-60k

职位诱惑：

生鲜电商交通补贴用餐补贴

职位描述：

工作职责:
1、负责公司电商平台的技术选型、架构搭建，完成系统整合过程的软件架构设计，解决开发中各种系统架构问题。
2、优化现有系统的性能，解决软件系统关键技术问题，核心功能的模块设计；
3、营造技术学习氛围，带领团队不断完善开发方法及流程，提升开发效率与质量，加强技术标准和规范；
任职资格:
1、5年以上JAVA开发经验，熟悉分布书系统的设计和应用。熟悉分布式、数据库、缓存、消息等机制，能对分布式常用的技术进行合理应用，具备实际架构经验。
2、扎实的计算机专业基础，对常用数据结构有深刻的理解和优秀的编码能力、算法设计能力和良好的开发习惯；
3、熟悉spring boot、spring cloud框架机制和实现原理，具有基于spring框架的大型分布式系统架构设计研发经验；
4、具有高并发、高可用、高性能的分布式后端系统设计和开发经验，有电商系统项目背景优先；
5、优秀的系统分析和问题解决能力，能够攻克复杂的系统难题。
6、具备业务需求、流程和模式的梳理能力，能把握核心问题，并进行梳理、归纳和抽象。
7、良好的沟通表达能力和团队协作能力，勤奋好学，能够快速适应变化。

工作地址

上海 - 浦东新区 - 盛夏路500弄　　查看地图

Java架构师（中间件） / 35k-50k

职位描述：

职位描述：

1、 提供高可靠高可用的基础组件

2、 解决基础架构线上线下问题

3、 基础架构的应用开发

4、 支撑和帮助业务团队架构服务的实施，支持运维团队的中间件部署

任资资格：

1、5年以上软件开发经验，精通Java语言，熟悉软件开发流程，熟悉常用项目构建工具，如Maven等

2、熟悉mysql，对其常用知识点理解深刻，熟悉数据库性能优化

3、熟悉spring,springmvc,spring boot,mybatis等常用框架的一种或多种

4、熟悉zookeeper, mq, redis，hbase, elastic search等常用中间件技术的一种或多种

5、熟悉负载均衡,大规模集群,故障处理,系统监控

6、良好的沟通能力、团队合作精神；认真负责、具有高度责任感；优秀的学习能力；快速解决技术问题的能力

架构师 / 45k-65k

职位描述：

工作职责：

负责有赞业务业务规划与需求讨论、业务领域建模、方案预研以及系统研发。

工作内容：

1.带领小组同学实现自己所负责域的产品技术目标，并且带领团队一起成长；

2.深入理解业务，围绕业务的发展、产品规划等做技术规划，并且能够驱动落地；

3.负责项目的设计方案评审，负责项目管理和技术难点攻关；

4.保持优秀的自我管理和持续学习能力，主导技术分享和培训，提升团队战斗力和个人影响力。

任职条件：

1.计算机相关专业本科或以上学历，五年以上Java开发经验，编程基础扎实；

2.精通Java开发语言，熟悉jvm、web、缓存、分布式架构、消息中间件等核心技术；

3.深入了解Java EE相关的主流框架，并熟知它的原理及机制，如Spring、Mybatis等；

4.熟悉MySQL，对数据库有较强的设计能力，同时熟悉大数据相关技术；

5.具备良好的面向对象的设计能力，熟悉面向对象设计原则，掌握设计模式及应用场景；

7.对于有过高并发、高可用、高性能、稳定性保障，以及大数据处理实际项目经验者优先；

8.具有高度的责任心与自驱力，以及良好的沟通协作、应急响应与处理问题的能力，能够通过技术能力贡献于业务成功。

什么是软件架构？

如何写一个架构设计文档，文档中应该包含哪些方面的内容？

子类 override 父类的方法后，想要修改抛出的异常，

那么子类方法抛出的异常类应该是父类方法抛出异常类的子类还是父类？

Spring 是如何实现单例的？

和设计模式中的单例实现方式有什么不同？

淘宝这样的大规模分布式互联网应用系统使用了哪些技术方案和手段，
主要解决什么问题？

什么是 CAP 原理？

请描述某个你熟悉的 NoSQL 产品是如何解决 CAP 问题的。

如何进行性能测试，性能测试的流程是什么？性能测试的主要关注指标有哪些？

为什么在系统性能测试的时候，随着并发请求数的逐渐增加，错误响应（或者响应超时）的比例快速增加？请从操作系统的线程与进程调度原理以及计算机内部资源使用角度进行分析。

为什么支持异步I/O 的 Web 服务器（比如 Nginx）要比阻塞式的 Web 服务器（比如 Apache）性能好很多，前者要比后者可以处理的并发连接请求多几十甚至数百倍？请从异步I/O 的线程阻塞特性进行分析。

给定一个 key，为什么可以在 Hash 表中快速查找到 value？

数据库索引是如何存储的？

Java 虚拟机的垃圾回收原理是什么？

你怎么理解领域驱动设计 DDD？

DDD 的优缺点是什么？

导致系统故障无法正常访问的原因有哪些？保障系统稳定高可用的方案有哪些？请列举并简述。

如何保护数据库中存储的用户密码，请用时序图用户密码加密存储与登录验证的过程。

Spark 为什么比 MapReduce 快？

淘宝，头条这些应用会针对不同用户推荐不同的商品和内容， 他们是如何做到的？用了哪些算法？

Google 搜索结果页面是如何排序的，正好使用户最想看到的页面排在前面？

区块链是如何保证数据无法被篡改的？

什么是边缘计算？

如果你觉得系统需要进行重构，但是老板和团队成员都觉得没必要，你如何说服大家？

架构师的主要职责

编写架构设计文档（week1）

开发编程框架（week2）

重构软件代码（week3）

设计系统架构（week4）

进行技术选型，解决技术应用中的问题（week5-6）

优化系统性能（week7-9）

模块分解与微服务架构重构（week10）

保障系统安全与高可用（week11）

大数据应用（week12-13）

技术创新（week14）

沟通管理（week15）

架构师主要能力

编程能力

基础技术掌握能力

常用技术产品的理解与应用能力

性能优化与分析故障的能力

常用架构模式和框架的理解与应用能力

建模以及设计文档的方法和能力

业务理解与功能模块及非功能模块拆解能力

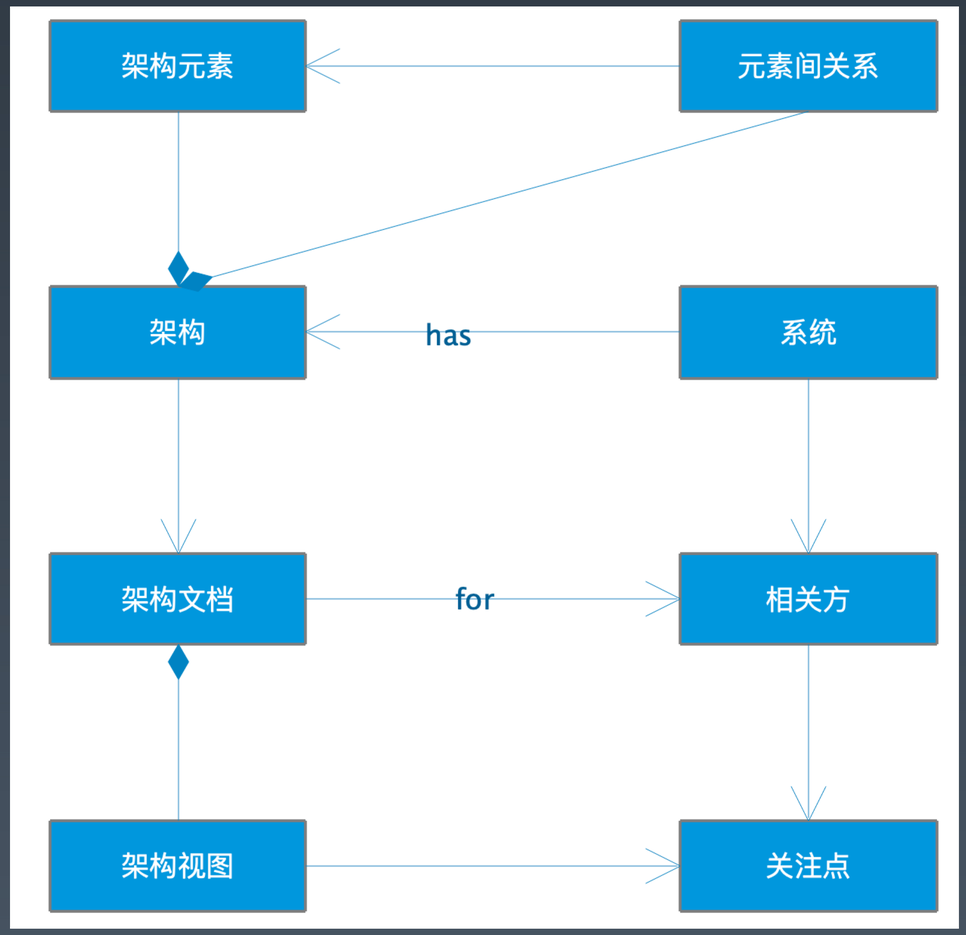
快速学习能力

沟通与领导能力

什么是软件架构？

软件架构，是有关软件整体结构与组件的抽象描述，用于指导大型软件系统各个方面的设计。

- 维基百科



关于软件开发的几个事实

软件技术的进步使得程序员不需要了解技术细节和原理就能开发出能用的软件。

让程序员关注更少的事情有助于提高软件开发效率和质量。

什么是架构师？

架构师是做架构设计、对系统架构负责的那个人。

架构师是一顶帽子，而不是一把椅子；架构师是一个角色而不是一个职位。

如何做软件架构

编写架构设计文档（week1）

开发编程框架（week2）

重构软件代码（week3）

设计系统架构（week4）

进行技术选型，解决技术应用中的问题（week5-6）

优化系统性能（week7-9）

模块分解与微服务架构重构（week10）

保障系统安全与高可用（week11）

大数据应用（week12-13）

技术创新（week14）

沟通管理（week15）

大家关心的一些问题

架构师与全栈工程师的区别是什么？两者之间是否有联系？

架构师应该怎么成长？哪些人适合做架构师？

技术的广度和深度怎么去选择和平衡？

面对一个陌生领域，或者复杂问题时，这种情况就好比您的工作经验比较少的领域，如何突破自我，做到驾轻就熟的？

有没有什么好的方式沉淀领域（行业）知识，以便构建个人中台？

学完之后，怎么应聘架构师？

感觉单单靠老师讲课还不够，希望老师推荐一些必备技能的书单，让我们在跟着老师学习过程中还可以有目标的去看一些书

如何通过训练营提高自己

架构师训练营，而不是架构训练营

- 架构方法、架构模式、关键知识点可以训练，但是架构一定要实践，一定要关注场景

课程中所有的技术都只是例子，通过学习例子训练架构思维，构建知识体系

- 通过例子，总结模式，通过模式，构建知识体系

THANKS!

极客大学架构师训练营

李智慧

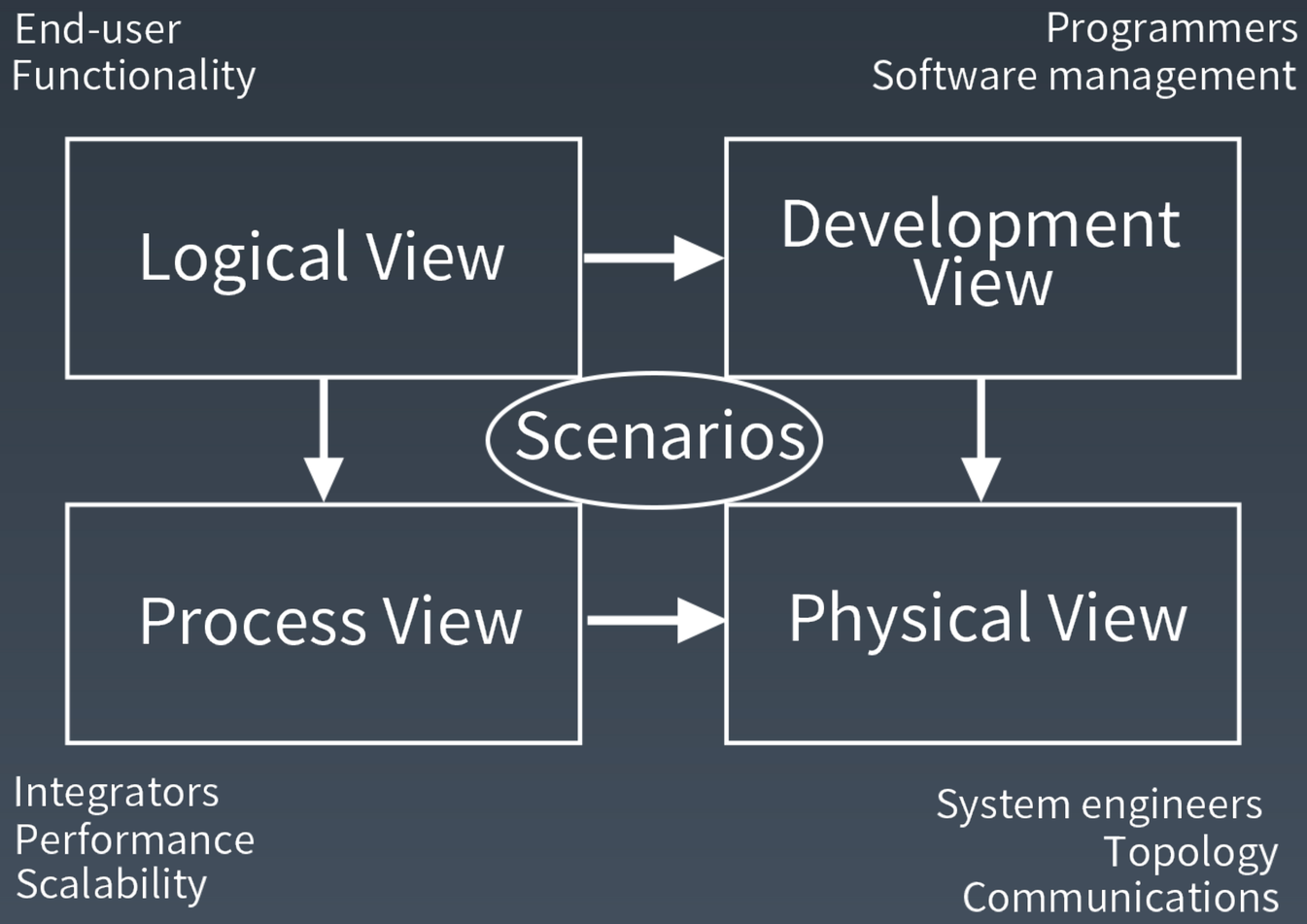
4+1视图模型：
软件开发的本质是什么？如何进行软件架构设计？

4+1 架构视图

软件架构 = {元素，形式，关系/约束}

单一的视图无法完整的表达架构，因此需要具备完整的视图集

- 逻辑视图（Logical View），设计的对象模型
- 过程视图（Process View），捕捉设计的并发和同步特征。
- 物理视图（Physical View），描述了软件到硬件的映射，反映了部署特性。
- 开发视图（Development View），描述了在开发环境中软件的静态组织结构。
- 场景视图(scenarios)，描述用例场景


End-user
Functionality
Programmers
Software management
Logical View
Development View
Scenarios
Process View
Physical View
Integrators
Performance
Scalability
System engineers
Topology
Communications

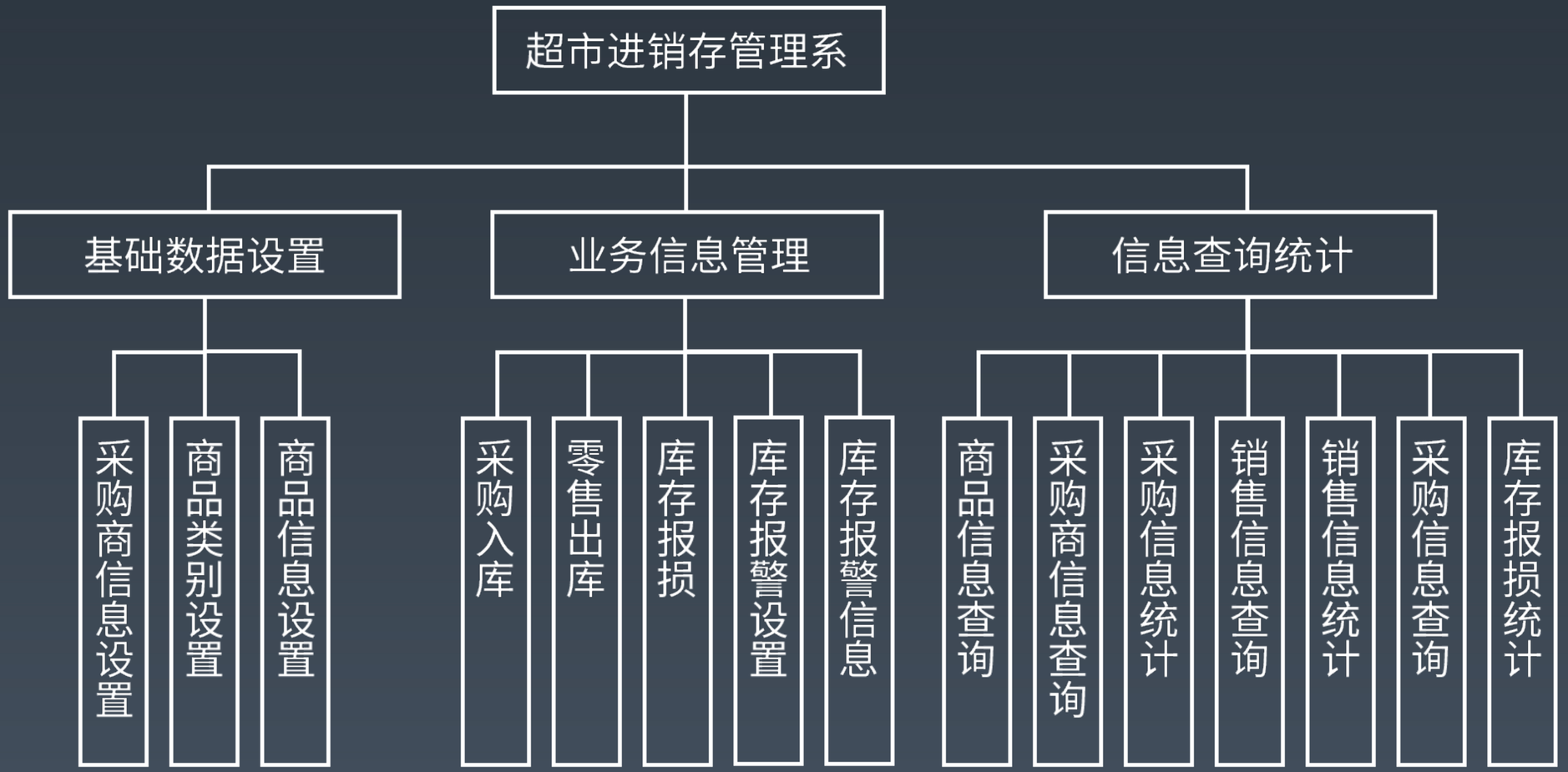
逻辑视图

相关方：客户，用户，开发组织管理者。

视角：系统的功能元素，以及它们接口，职责，交互。

主要元素：系统，子系统，功能模块，子功能模块，接口。

用途：开发组织划分，成本/进度的评估。


超市进销存管理系
基础数据设置
业务信息管理
信息查询统计
采购商信息设置
商品类别设置
商品信息设置
采购入库
零售出库
库存报损
库存报警设置
库存报警信息
商品信息查询
采购商信息查询
采购信息统计
销售信息查询
销售信息统计
采购信息查询
库存报损统计

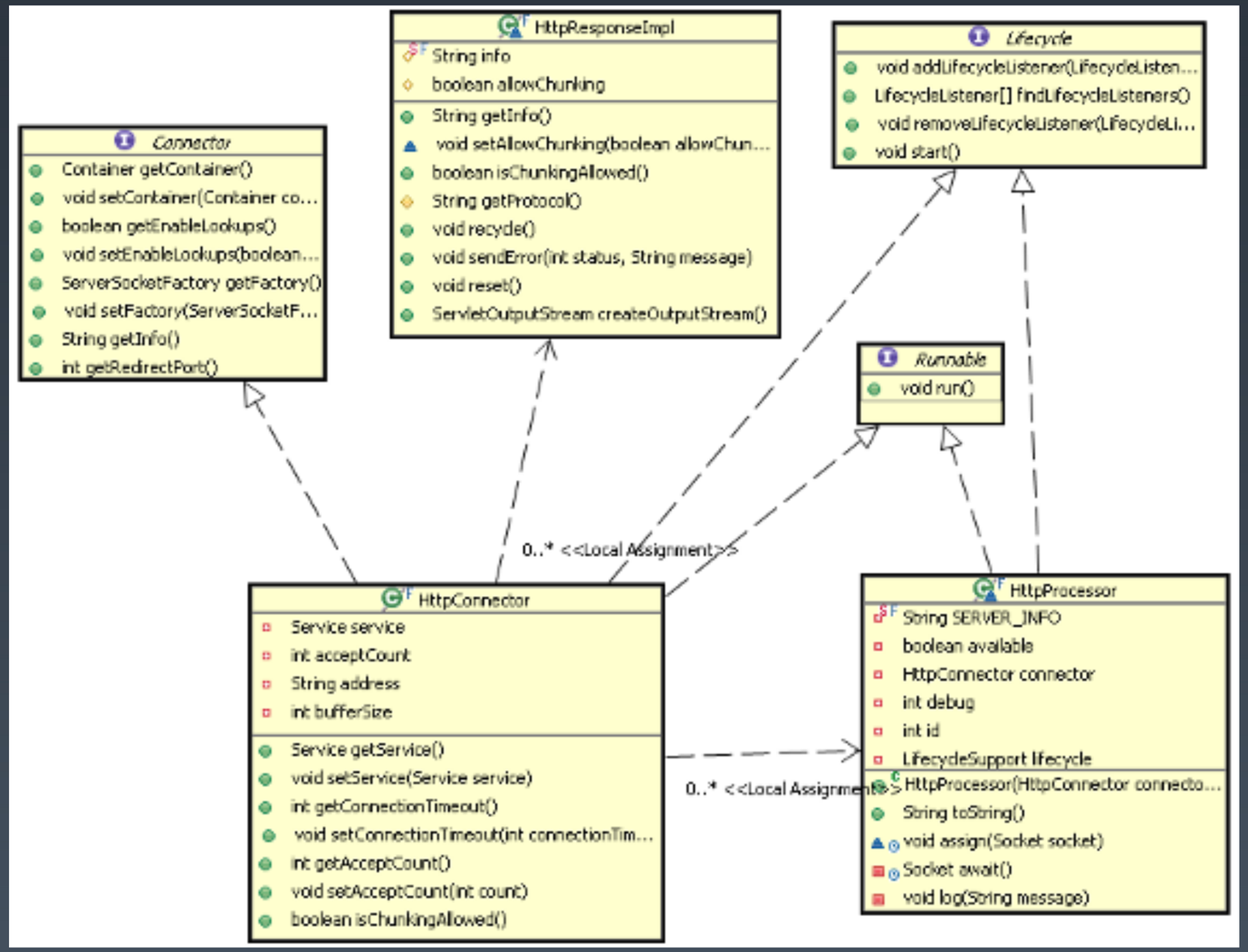
开发视图

相关者：开发相关人员，测试人员

视角：系统如何开发实现

主要元素：描述系统的层，分区，包，框架，系统通用服务，业务通用服务，类和接口，系统平台和相关基础框架。

用途：指导开发组织设计和开发实现

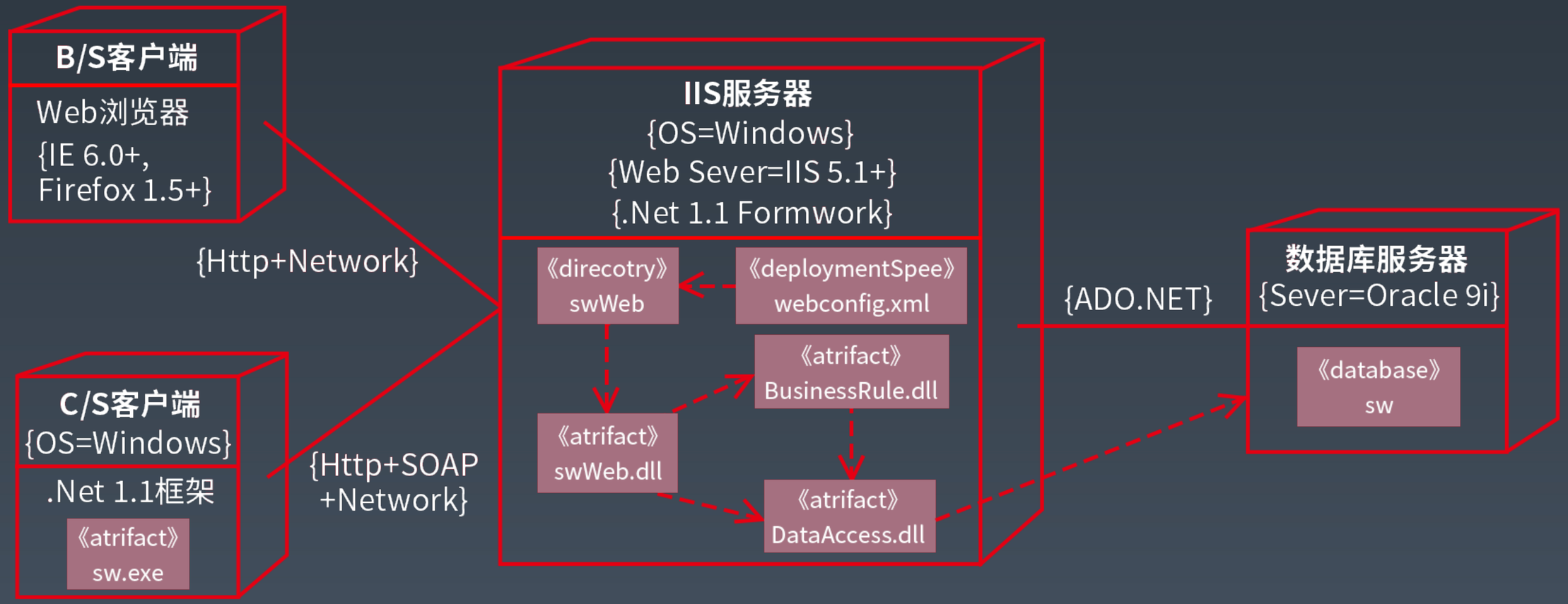

HttpResponseImpl
String info
boolean allowChunking
String getInfo()
void setAllowChunking(boolean allowChun...
boolean isChunkingAllowed()
String getProtocol()
void recycle()
void sendError(int status, String message)
void reset()
ServletOutputStream createOutputStream()
Lifecycle
void addLifecycleListener(LifecycleListen...
LifecycleListener[] findLifecycleListeners()
void removeLifecycleListener(LifecycleLi...
void start()
Connector
Container getContainer()
void setContainer(Container co...
boolean getEnableLookups()
void setEnableLookups(boolean...
ServerSocketFactory getFactory()
void setFactory(ServerSocketF...
String getInfo()
int getRedirectPort()
Runnable
void run()
0..* <<Local Assignment>>
HttpConnector
Service service
int acceptCount
String address
int bufferSize
Service getService()
void setService(Service service)
int getConnectionTimeout()
void setConnectionTimeout(int connectionTim...
int getAcceptCount()
void setAcceptCount(int count)
boolean isChunkingAllowed()
0..* <<Local Assignment>>
HttpProcessor
String SERVER_INFO
boolean available
HttpConnector connector
int debug
int id
LifecycleSupport lifecycle
HttpProcessor(HttpConnector connecto...
String toString()
void assign(Socket socket)
Socket await()
void log(String message)

物理视图

相关者：系统集成商，系统运维人员。

视角：系统逻辑组件到物理节点的物理部署和节点之间的物理网络配置。

主要元素：物理节点以及节点的通信。

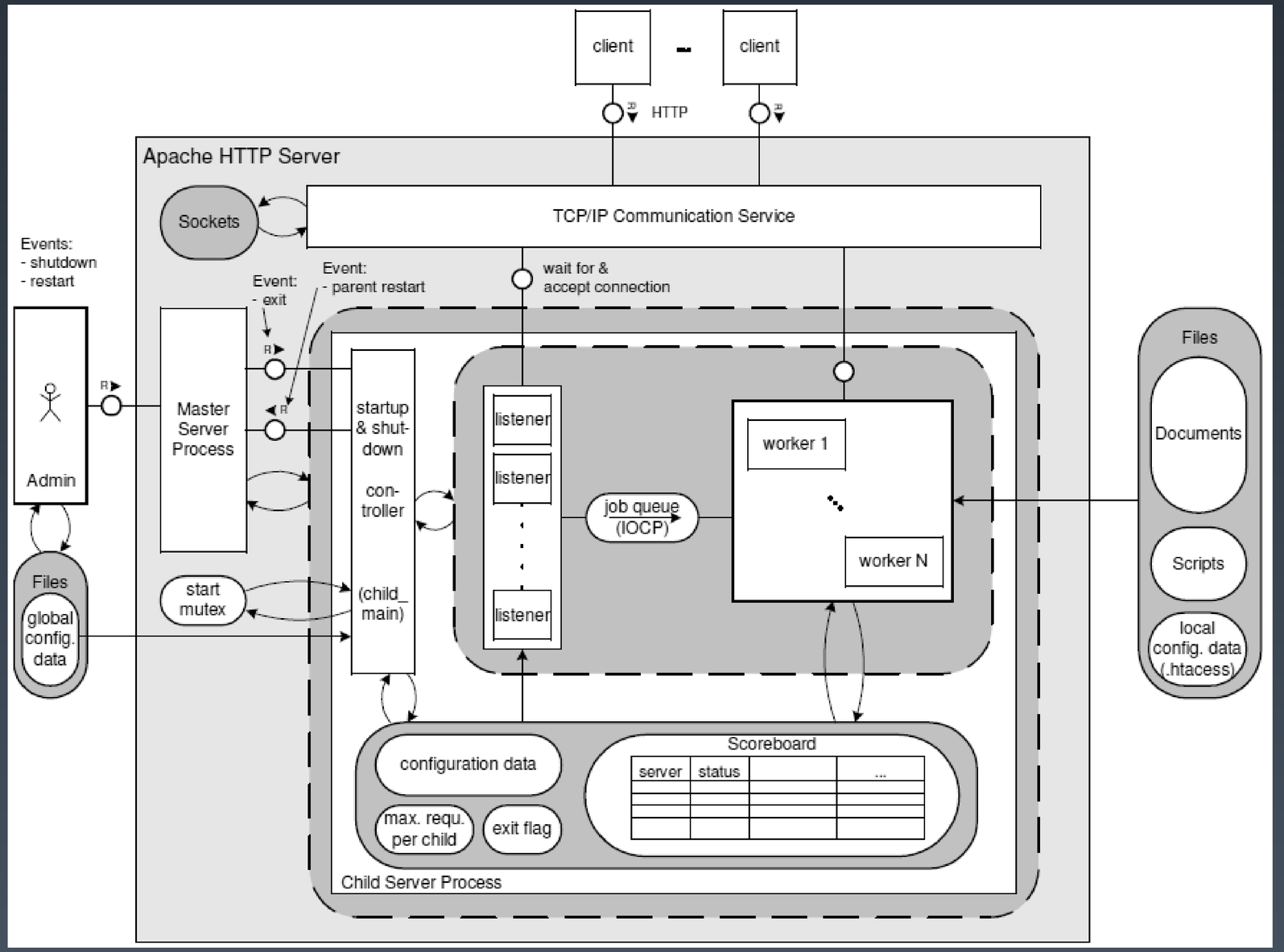

B/S客户端
Web浏览器
{IE 6.0+,
Firefox 1.5+}
{Http+Network}
C/S客户端
{OS=Windows}
.Net 1.1框架
《atrifact》
sw.exe
{Http+SOAP
+Network}
IIS服务器
{OS=Windows}
{Web Sever=IIS 5.1+}
{.Net 1.1 Formwork}
《direcotry》
swWeb
《deploymentSpee》
webconfig.xml
《atrifact》
BusinessRule.dll
《atrifact》
swWeb.dll
《atrifact》
DataAccess.dll
{ADO.NET}
数据库服务器
{Sever=Oracle 9i}
《database》
sw

过程视图

相关者：性能优化，开发相关人员。

视角：系统运行时线程，进程的情况。

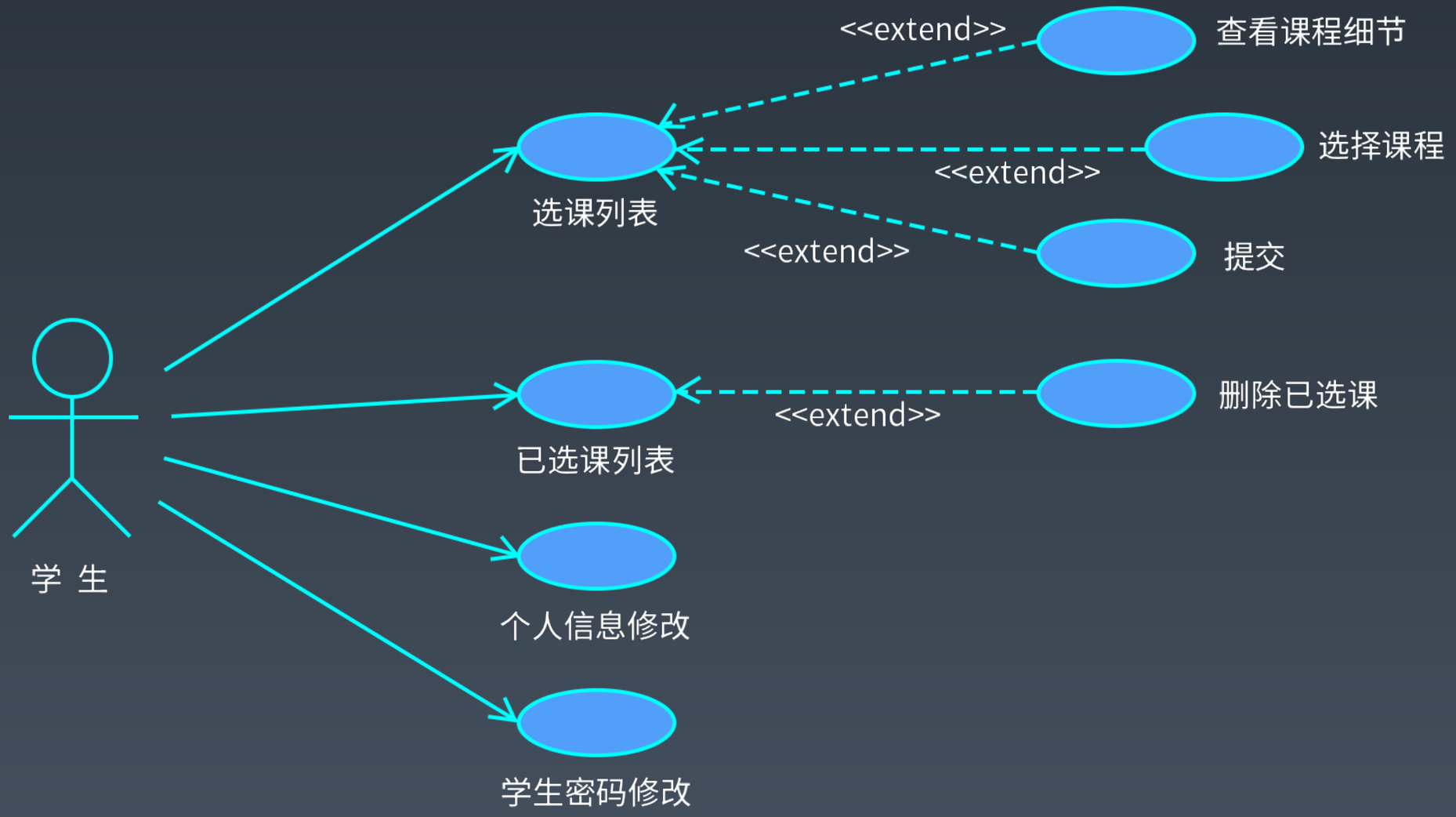
主要元素：系统进程，线程以及处理队列等。


client
client
HTTP
Apache HTTP Server
Sockets
TCP/IP Communication Service
Events:
- shutdown
- restart
Event:
- exit
Event:
- parent restart
wait for &
accept connection
Admin
Master
Server
Process
startup
& shut-
down
con-
troller
(child_
main)
start
mutex
listener
listener
listener
job queue
(IOCP)
worker 1
worker N
Files
global
config.
data
Files
Documents
Scripts
local
config. data
(.htacess)
configuration data
max. requ.
per child
exit flag
Scoreboard
server
status
...
Child Server Process

场景视图

相关者：用户，设计和开发人员。

视角：概括了架构上最重要的场景（最典型或者最有风险）及其非功能性需求，通过这些场景的实现，阐明了架构的广度或众多架构元素运行的方式。


<<extend>>
查看课程细节
选课列表
选择课程
<<extend>>
<<extend>>
提交
已选课列表
删除已选课
<<extend>>
学 生
个人信息修改
学生密码修改

软件建模语言

如何使用 UML 进行软件架构设计与建模？

什么是模型？

模型是一个系统的完整的抽象。人们对某个领域特定问题的求解及解决方案，对它们的理解和认识都蕴涵在模型中。

通常，开发一个计算机系统是为了解决某个领域特定问题，问题的求解过程，就是从领域问题到计算机系统的映射。

为什么要建造模型？

建造传统模型的目的

- 为了证明某件事物能否工作
- 前提：建造模型的成本远远低于建造实物的成本
  - 造飞机
  - 造高楼

建造软件模型的目的

- 为了与它人沟通
- 为了保存软件设计的最终成果
- 前提：除非模型比代码更说问题

何时、何处画图？

何时画图？

- 讨论、交流时
- 最终设计文档
  - 只保留少量的、重要的图
  - 避免涉及过多内容和实现细节

何处画图？

- 白板
- 绘图工具，如：Visio、Aastah
- draw.io

UML 简介

什么是 UML?

- Unified Modeling Language，或统一建模语言
- 以图形方式描述软件的概念

UML 可用来描述：

- 某个问题领域
- 构思中的软件设计
- 描述已经完成的软件实现

UML 图的分类 -静态图

静态图 - 通过描述类、对象和数据结构以及它们之间存在的关系，来描述软件要素中不变的逻辑结构。

- 用例图（Use Case Diagrams）
- 对象图（Object Diagrams）
- 类图（Class Diagrams）
- 组件图（Component Diagrams）
- 包图 (Package Diagrams)
- 部署图（Deployment Diagrams）

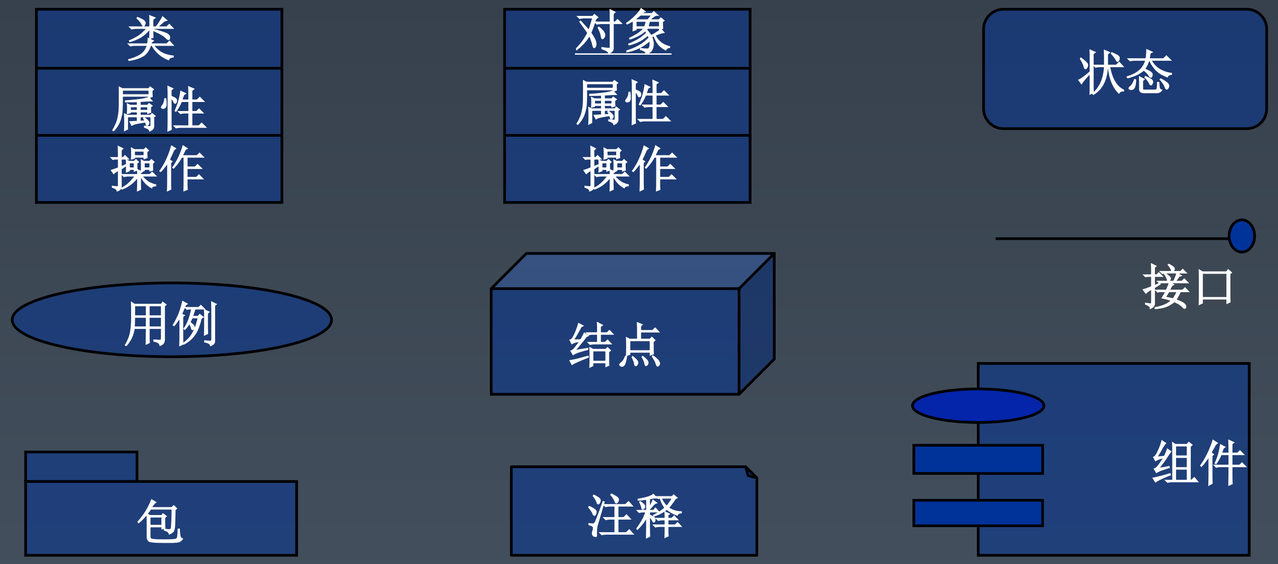
UML 图的分类 -动态图

动态图 - 通过描绘执行流程或者实体状态变化的方式，来展示软件实体在执行过程 中的变化过程。

- 协作图（Collaboration Diagrams）
- 序列图（Sequence Diagrams）
- 活动图（Activity Diagrams）
- 状态图（State Diagrams）

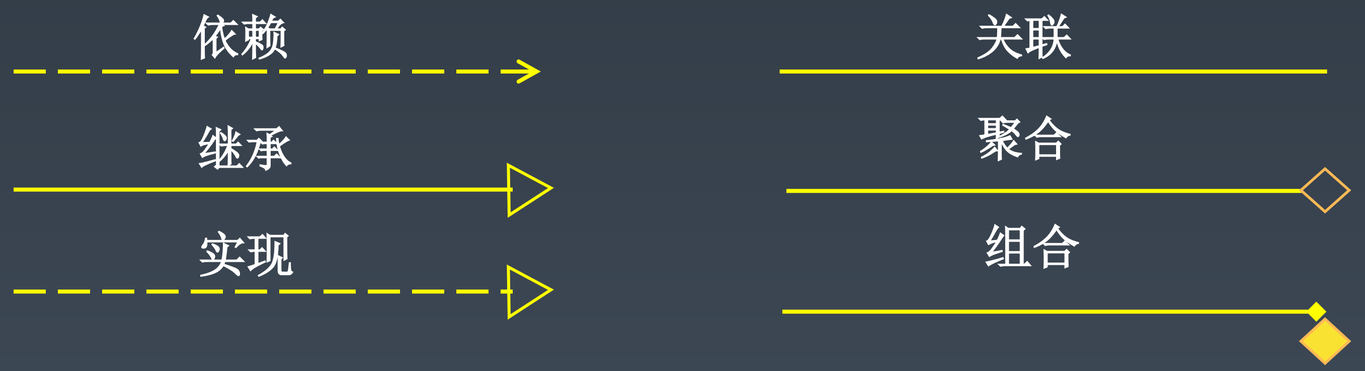
通用模型元素

可以在图中使用的概念统称为模型元素。模型元素在图中用其相应的视图元素（符号）表示，下图给出了常用的元素符号：类、对象、结点、包和组件等。



通用模型元素

模型元素与模型元素之间的连接关系也是模型元素，常见的关系有关联（association）、泛化（generalization）、依赖（dependency）和聚合（aggregation）。这些关系的图示符号如图所示。



关联：连接（connect）模型元素及链接（link）实例。

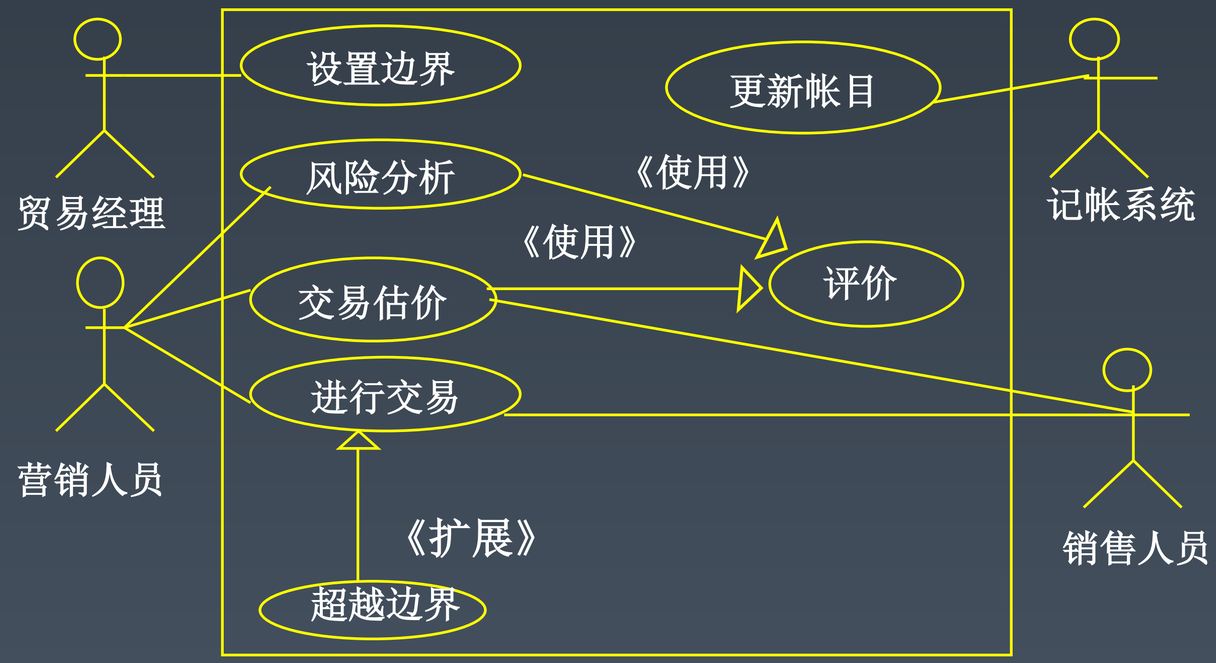
依赖：表示一个元素以某种方式依赖于另一种元素。

泛化：表示一般与特殊的关系，即“一般”元素是“特殊”关系的泛化。

聚合：表示整体与部分的关系。

用例建模

用例建模技术，用于描述系统的功能需求。在宏观上给出模型的总体轮廓。通过对典型用例的分析，使开发者能够有效地了解用户的需求。



用例模型描述的是外部执行者（Actor）所理解的系统功能。它描述了待开发系统的功能需求。

它驱动了需求分析之后各阶段的开发工作,不仅在开发过程中保证了系统所有功能的实现，而且被用于验证和检测所开发的系统，从而影响到开发工作的各个阶段和 UML 的各个模型。

用例模型由若干个用例图构成，用例图中主要描述执行者和用例之间的关系。在 UML 中，构成用例图的主要元素是用例和执行者及其它们之间的联系。

创建用例模型的工作包括：定义系统、确定执行者和用例、描述用例、定义用例间的关系、确认模型。

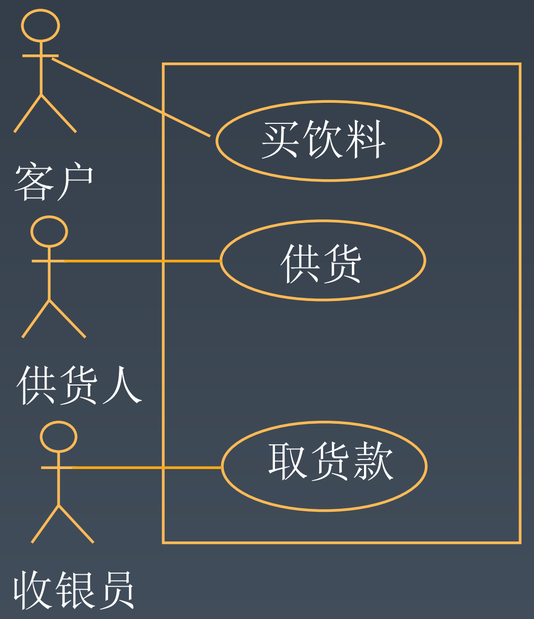
执行者（Actor）

执行者是指用户在系统中所扮演的角色。执行者在用例图中是用类似人的图形来表示，但执行者可以是人，也可以是一个外界系统。

注意：用例总是由执行者启动的。

如何确定执行者：

1. 谁使用系统的主要功能(主执行者)?
2. 谁需要从系统获得对日常工作的支持和服务?
3. 需要谁维护管理系统的日常运行（副执行者）?
4. 系统需要控制哪些硬件设备?
5. 系统需要与其它哪些系统交互?
6. 谁需要使用系统产生的结果（值）?

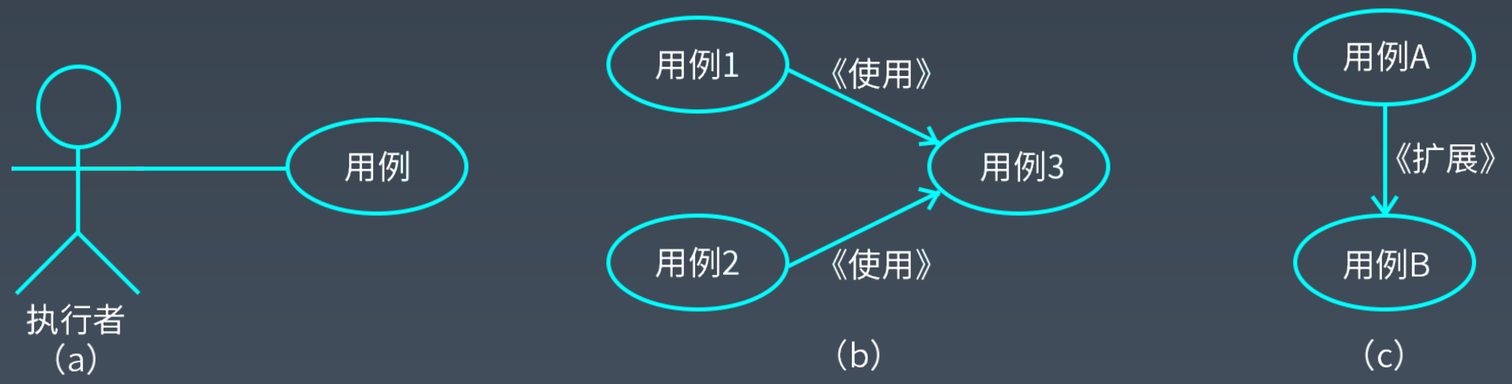


用例图描述了系统的功能需求，它是从执行者的角度来理解系统，用于捕获系统的需求，规划和控制项目；描述了系统外部的执行者与系统提供的用例之间的某种联系。

图中还有另外两种类型的连接，即《使用》和《扩展》关系，是两种不同形式的泛化关系。

《Use》表示一个用例使用另一个用例。

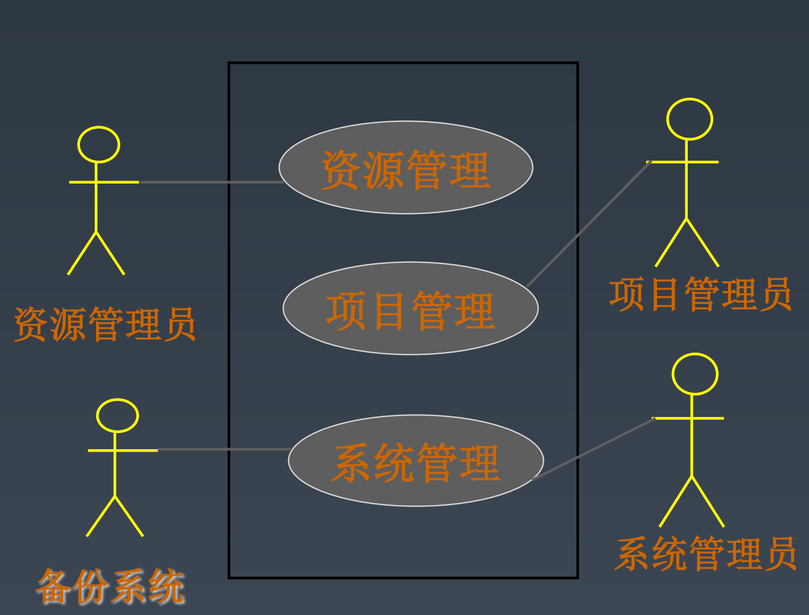
《Extend》通过向被扩展的用例添加动作来扩展用例。

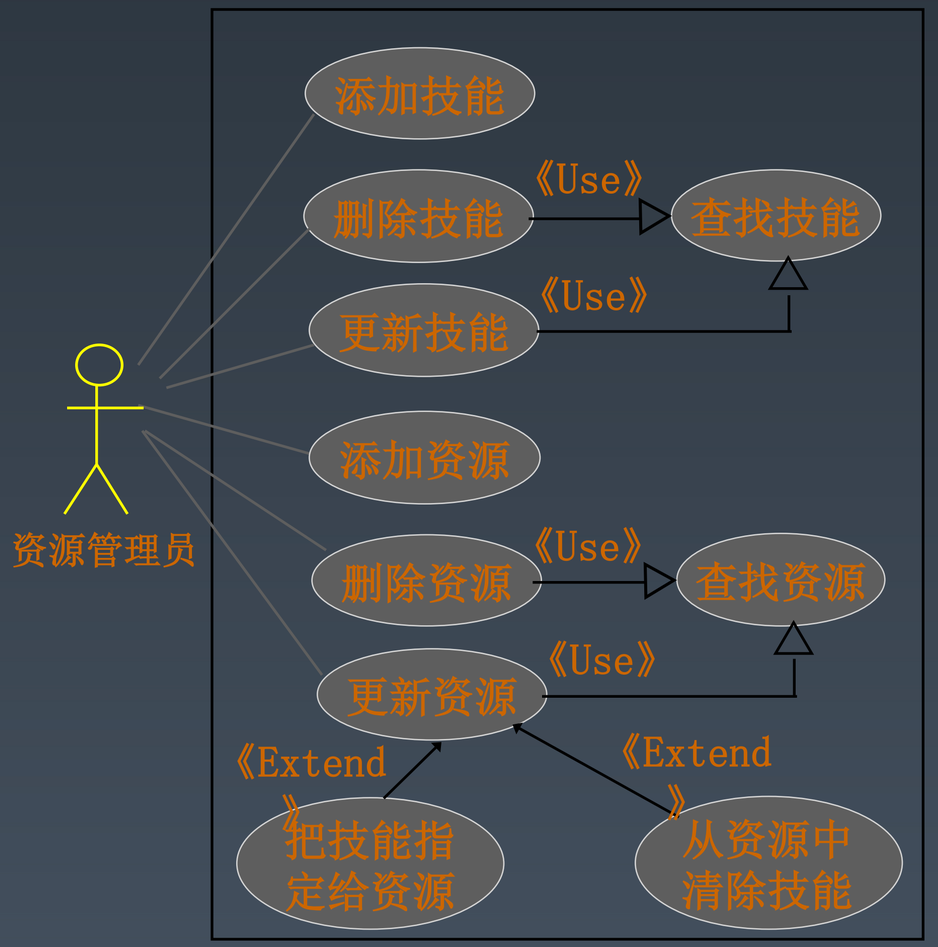


例 项目与资源管理系统的 Use case 图

系统的主要功能是：项目管理，资源管理和系统管理。项目管理包括项目的增加、删除、更新。资源管理包括对资源和技能的添加、删除和更新。系统管理包括系统的启动和关闭，数据的存储和备份等功能。

- 分析确定系统的执行者（角色）
  - 项目管理员、资源管理员、系统管理员、备份数据系统。
- 确定用例
  - 项目管理，资源管理和系统管理。
- 对用例进行分解，画出下层的 Use case 图
  - 对上层的用例进行分解,并将执行者分配到各层次的 Use case 图中。





Use Case图可以自顶而下不断精化，抽象出不同层次的Use Case图。

静态建模

任何建模语言都以静态建模机制为基础,标准建模语言UML也不例外。所谓静态建模是指对象之间通过属性互相联系，而这些关系不随时间而转移。

类和对象的建模，是 UML 建模的基础。UML 的静态建模机制包括：

- 用例图(Use case diagram)
- 类图(Class diagram)
- 对象图(Object diagram)
- 包图(Package diagram)
- 组件图(Component diagram)
- 部署图(Deployment diagram)

类与对象

面向对象的开发方法的基本任务是建立对象模型，是软件系统开发的基础。UML 中的类图（Class Diagram）与对象图（Object Diagram）表达了对象模型的静态结构，能够有效地建立专业领域的计算机系统对象模型。

属性（attribute）

属性用来描述类的特征，表示需要处理的数据。

属性定义：

visibility attribute-name : type = initial-value {property-string}

可见性 属性名：类型=缺省值{约束特性}

其中：可见性（visibility）表示该属性对类外的元素是否可见。

分为：

- public（+） 公有的。
- private（-） 私有的。
- protected（#） 受保护的。
- 默认（未声明）

操作

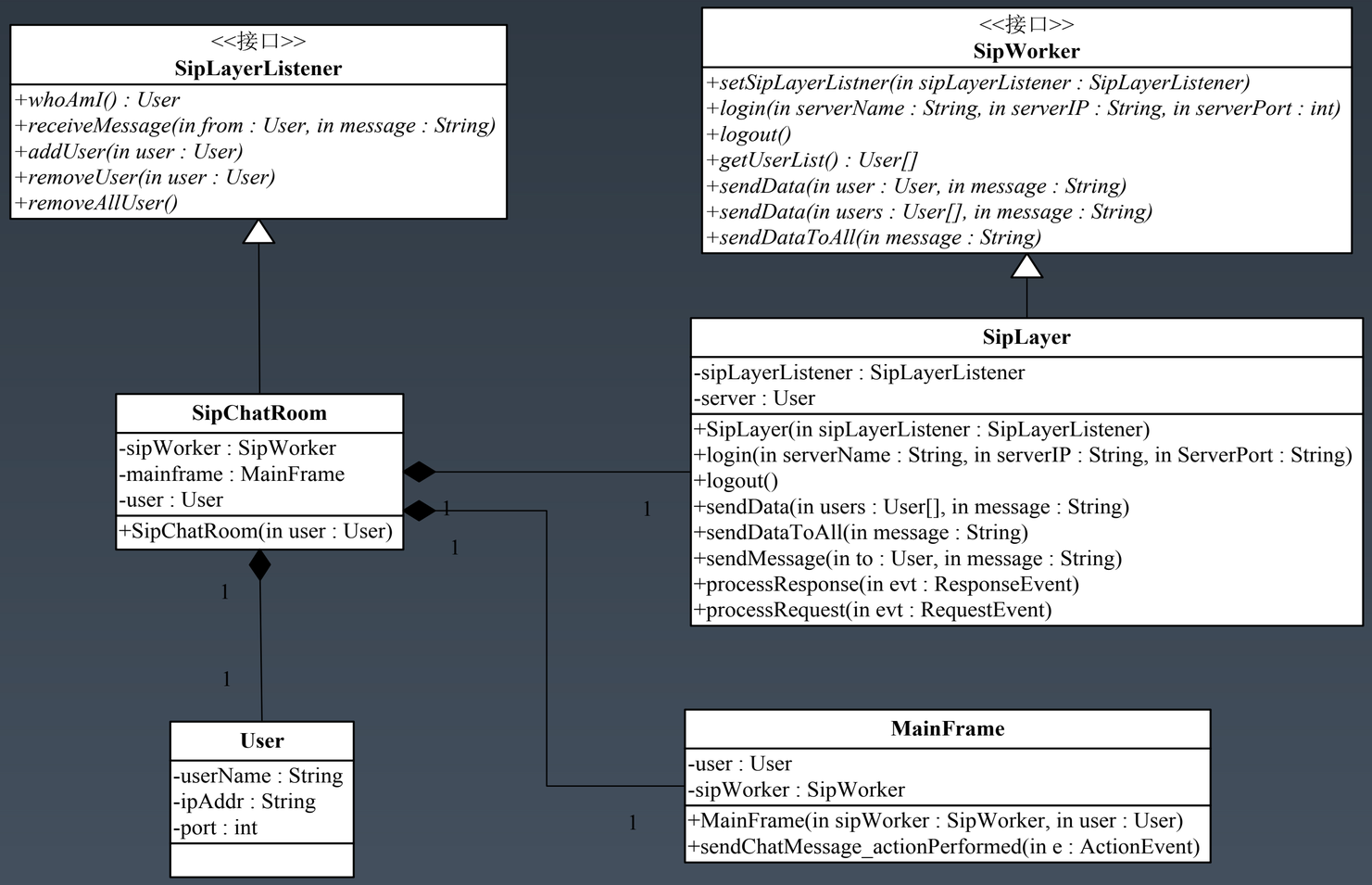
对数据的具体处理方法的描述则放在操作部分，操作说明了该类能做些什么工作。操作通常称为函数，它是类的一个组成部分，只能作用于该类的对象上。

操作定义：

visibility operating-name(parameter-list): return-type {property- string}

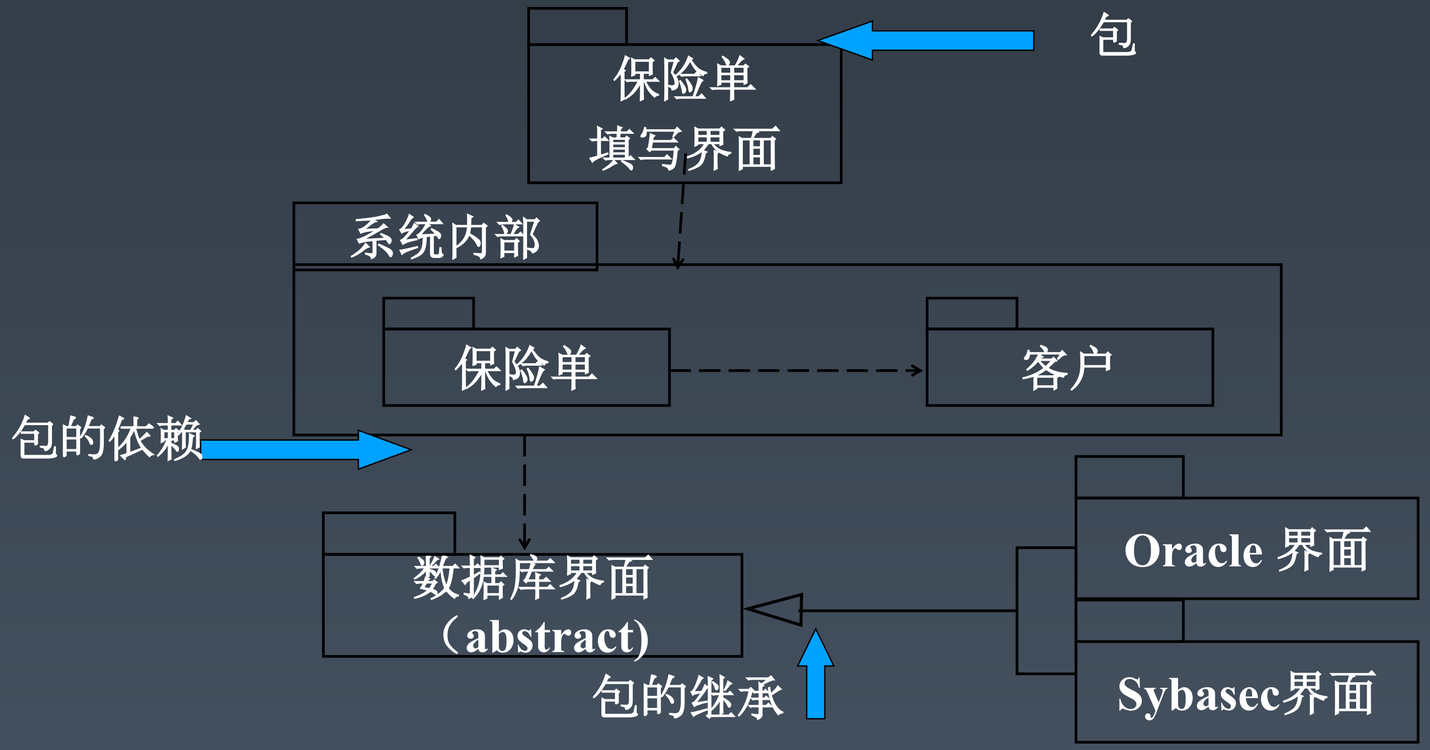
可见性 操作名（参数表）；返回类型{约束特性}

一个使用 Visio 绘制的类图



包图

一个最古老的软件方法问题是：怎样将大系统拆分成小系统。解决该问题的思路之一是将许多类集合成一个更高层次的单位，形成一个高内聚、低耦合的类的集合。UML 中这种分组机制叫包（Package）。引入包是为了降低系统的复杂性。



动态建模

动态模型主要描述系统的动态行为和控制结构。动态行为包括系统中对象生存期内可能的状态以及事件发生时状态的转移，对象之间动态合作关系，显示对象之间的交互过程以及交互顺序，同时描述了为满足用例要求所进行的活动以及活动间的约束关系。

在动态模型中,对象间的交互是通过对象间消息的传递来完成的。对象通过相互间的通信（消息传递）进行合作，并在其生命周期中根据通信的结果不断改变自身的状态。

动态模型

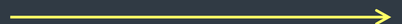
动态模型主要描述系统的动态行为和控制结构。

包括四类图：状态图、活动图、时序图、合作图。

- 状态图（state diagram）：状态图用来描述对象，子系统，系统的生命周期。
- 活动图（activity diagram）：着重描述操作实现中完成的工作以及用例实例或对象中的活动，活动图是状态图的一个变种。
- 时序图（sequence diagram)）：是一种交互图，主要描述对象之间的动态合作关系以及合作过程中的行为次序，常用来描述一个用例的行为。
- 合作图（collaboration diagram）：用于描述相互合作的对象间的交互关系，它描述的交互关系是对象间的消息连接关系。

UML 中的消息

简单消息（simple）

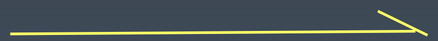


- 表示控制流，描述控制如何从一个对象传递到另一个对象，但不描述通信的细节。

同步消息（synchronous）

- 是一种嵌套的控制流，用操作调用实现。操作的执行者要到消息相应操作执行完并回送一个简单消息后，再继续执行。

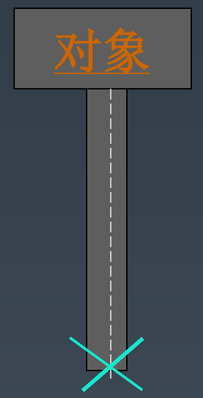
异步消息（asynchronous）



- 是一种异步的控制流，消息的发送者在消息发送后就继续执行，不等待消息的处理。

时序图

时序图（Sequence Diagram）用来描述对象之间动态的交互行为,着重体现对象间消息传递的时间顺序。



时序图存在两个轴：

- 水平轴表示一组对象
- 垂直轴表示时间

时序图中的对象用一个带有垂直虚线的矩形框表示，并标有对象名和类名。垂直虚线是对象的生命线,用于表示在某段时间内对象是存在的。

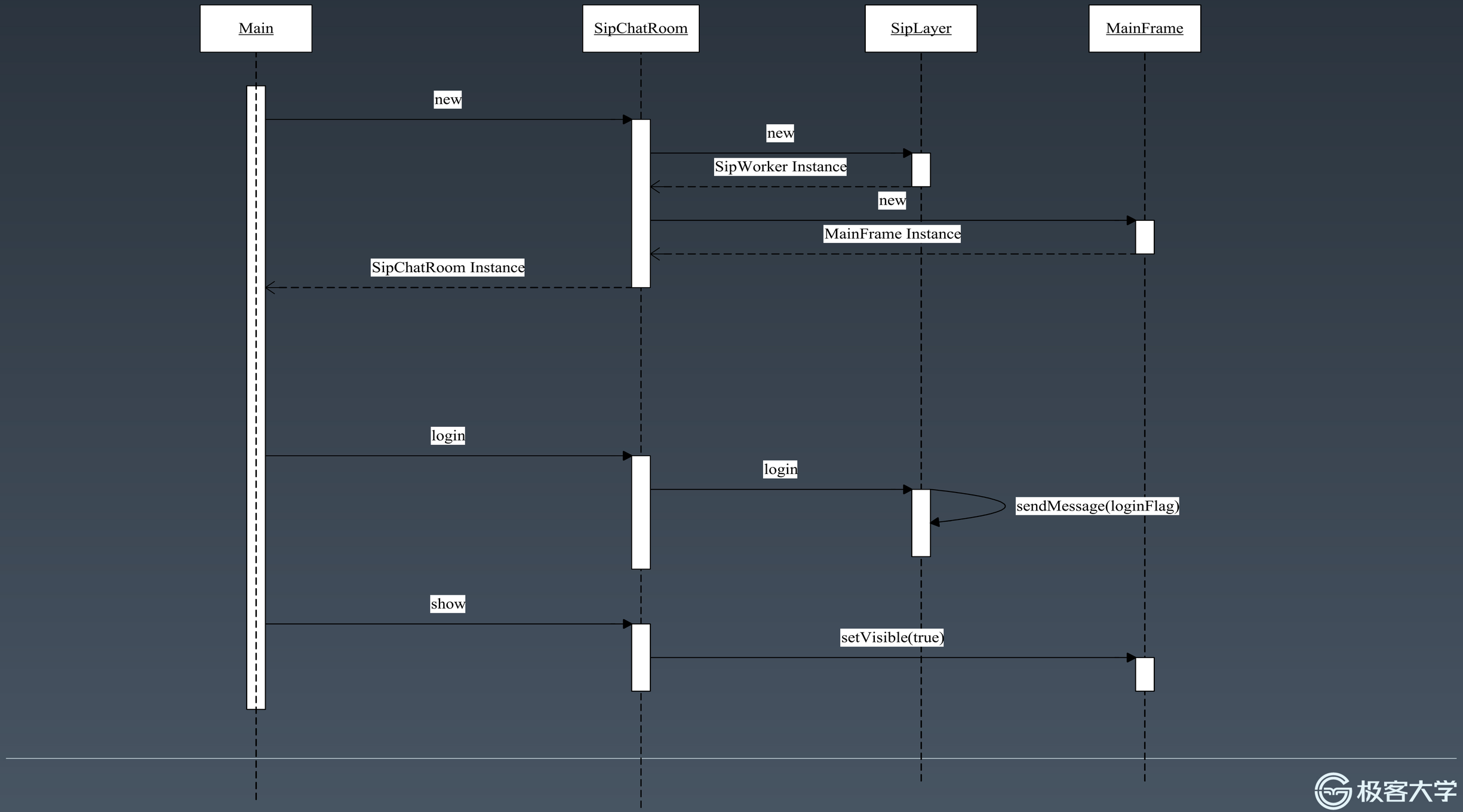
对象间的通信通过在对象的生命线之 间消息来表示，消息的箭头类型指明消息的类型。

顺序图的形式

有两种使用顺序图的方式：一般格式和实例格式。

实例格式详细描述一次可能的交互。没有任何条件和分支或循环，它仅仅显示选定情节（场景）的交互。

而一般格式则描述所有的情节。因此，包括了分支，条件和循环。


Main
SipChatRoom
SipLayer
MainFrame
new
new
SipWorker Instance
new
MainFrame Instance
SipChatRoom Instance
login
login
sendMessage(loginFlag)
show
setVisible(true)

活动图

活动图（Activity Diagram）的应用非常广泛,它既可用来描述操作（类的方法）的行为,也可以描述用例和对象内部的工作过程,并可用于表示并行过程。

活动图描述了系统中各种活动的执行的顺序。刻化一个方法中所要进行的各项活动的执行流程。

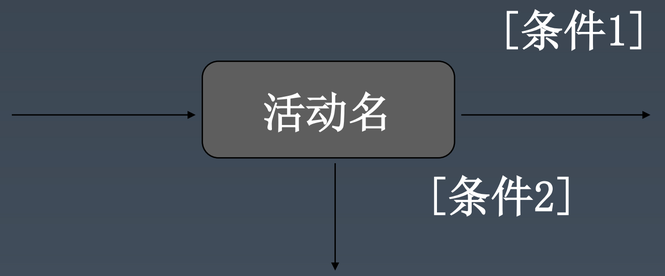
活动图中一个活动结束后将立即进入下一个活动（在状态图中状态的变迁可能需要事件的触发）。

活动图的模型元素

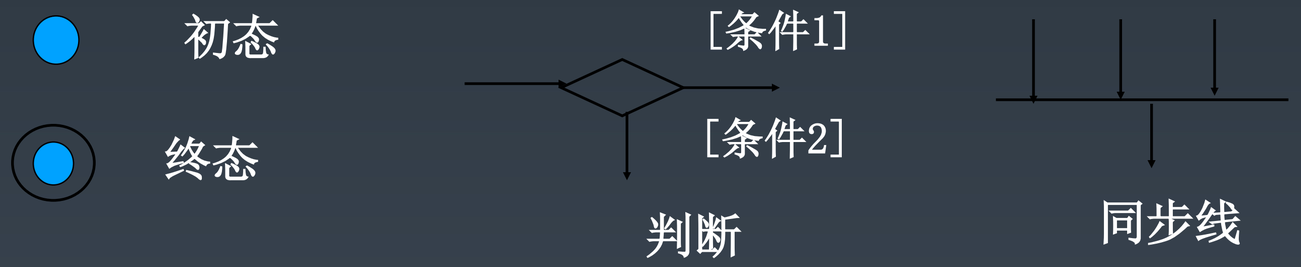
构成活动图的模型元素有：活动、转移、对象、信号、泳道等。

活动

- 是构成活动图的核心元素，是具有内部动作的状态，由隐含的事件触发活动的转移。
- 活动的解释依赖于作图的目的和抽象层次，在概念层描述中，活动表示要完成的一些任务；在说明层和实现层中，活动表示类中的方法。
- 活动用圆角框表示，标注活动名。
- 模型元素有：活动、转移、对象、信号、泳道等。

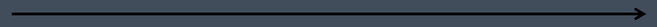


- 活动还有其它的图符：初态、终态、判断、同步。



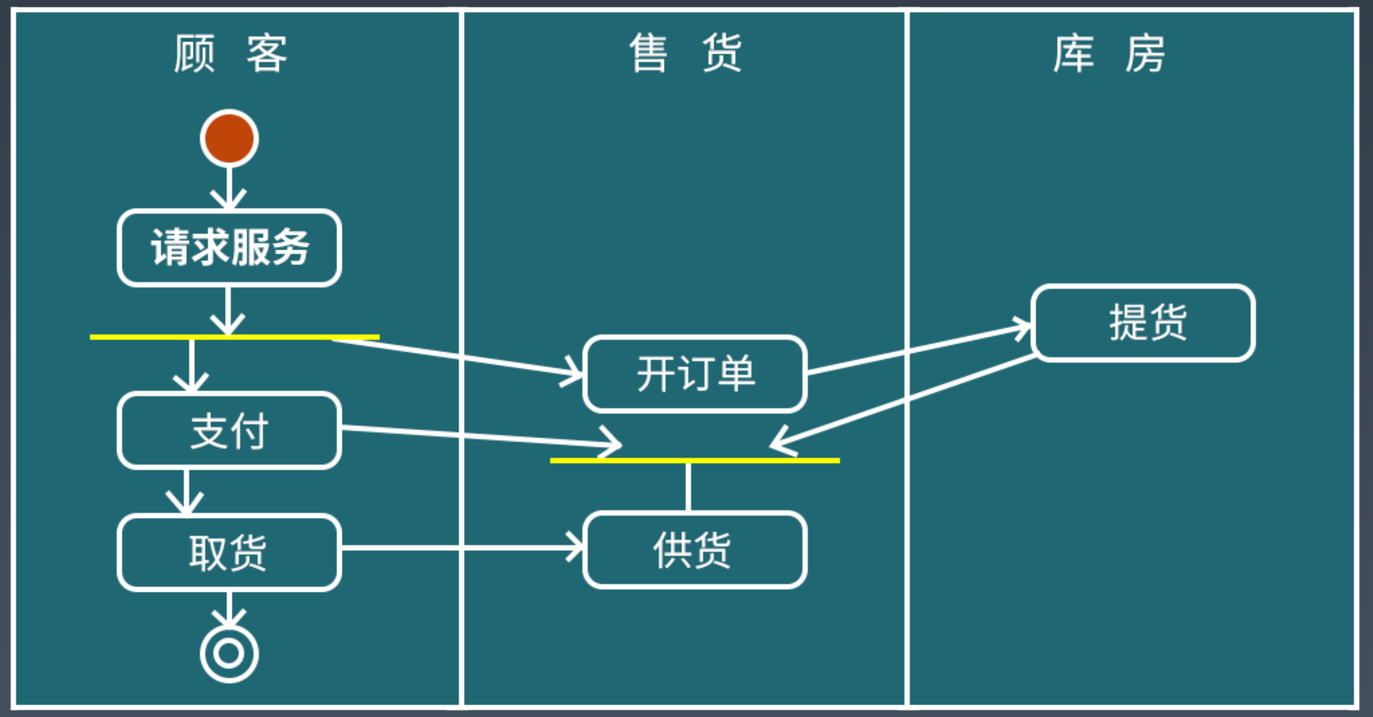
转移

- 转移描述活动之间的关系，描述由于隐含事件引起的活动变迁，即转移可以连接各活动及特殊活动（初态、终态、判断、同步线）。
- 转移用带箭头的直线表示，可标注执行该转移的条件，无标注表示顺序执行。



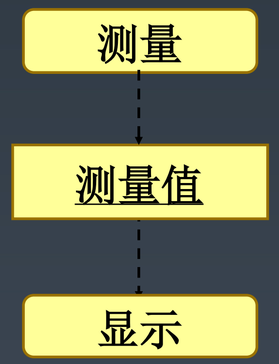
泳道

- 泳道进一步描述完成活动的对象，并聚合一组活动。活动图是另一种描述交互的方式，描述采取何种动作，做什么（对象状态改变），何时发生（动作序列），以及在何处发生（泳道）。
- 泳道也是一种分组机制。

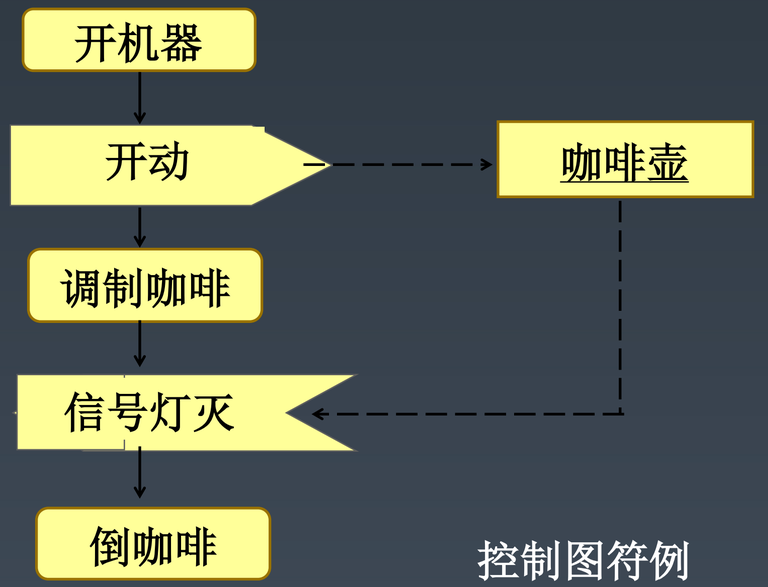


对象流

- 活动图中可以出现对象，对象作为活动的输入／输出，用虚箭头表示。



对象流



控制图符例

控制图符

- 活动图中可发送和接收信号，发送符号对应于与转移联系在一起的发送短句。接收符号也同转移联系在一起。

控制图符

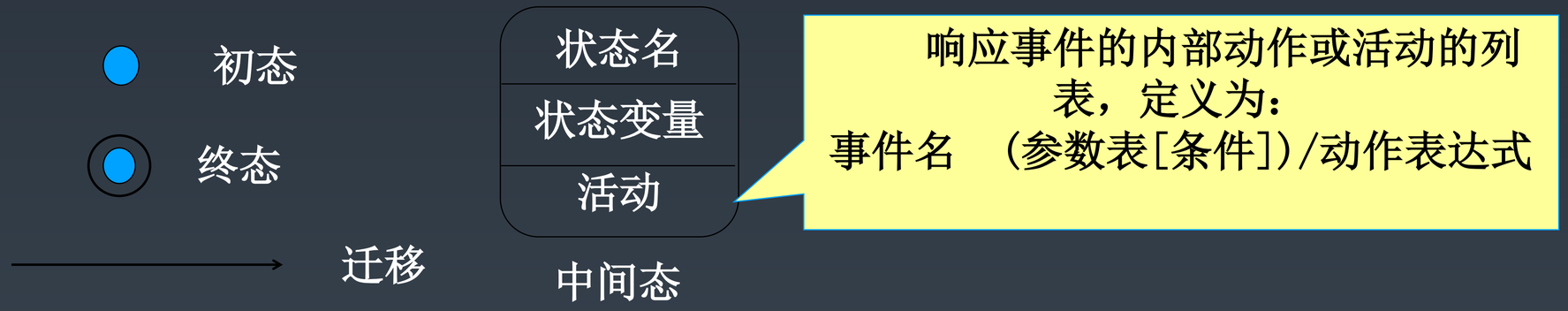
状态图

状态图（State Diagram）用来描述一个特定对象的所有可能的状态及其引起状态转移的事件。一个状态图包括一系列的状态以及状态之间的转移。

状态

所有对象都具有状态，状态是对象执行了一系列活动的结果。当某个事件发生后,对象的状态将发生变化。状态图中定义的状态有:

- 初态 - 状态图的起始点，一个状态图只能有一个初态。
- 终态 - 是状态图的终点，而终态则可以有多个。
- 中间状态 - 可包括三个区域:名字域、状态变量与活动域。
- 复合状态 - 可以进一步细化的状态称作复合状态。



状态变量 是状态图所显示的类的属性。

活动 列出了在该状态时要执行的事件和动作。有3个标准事件：

entry事件用于指明进入该状态时的特定动作。

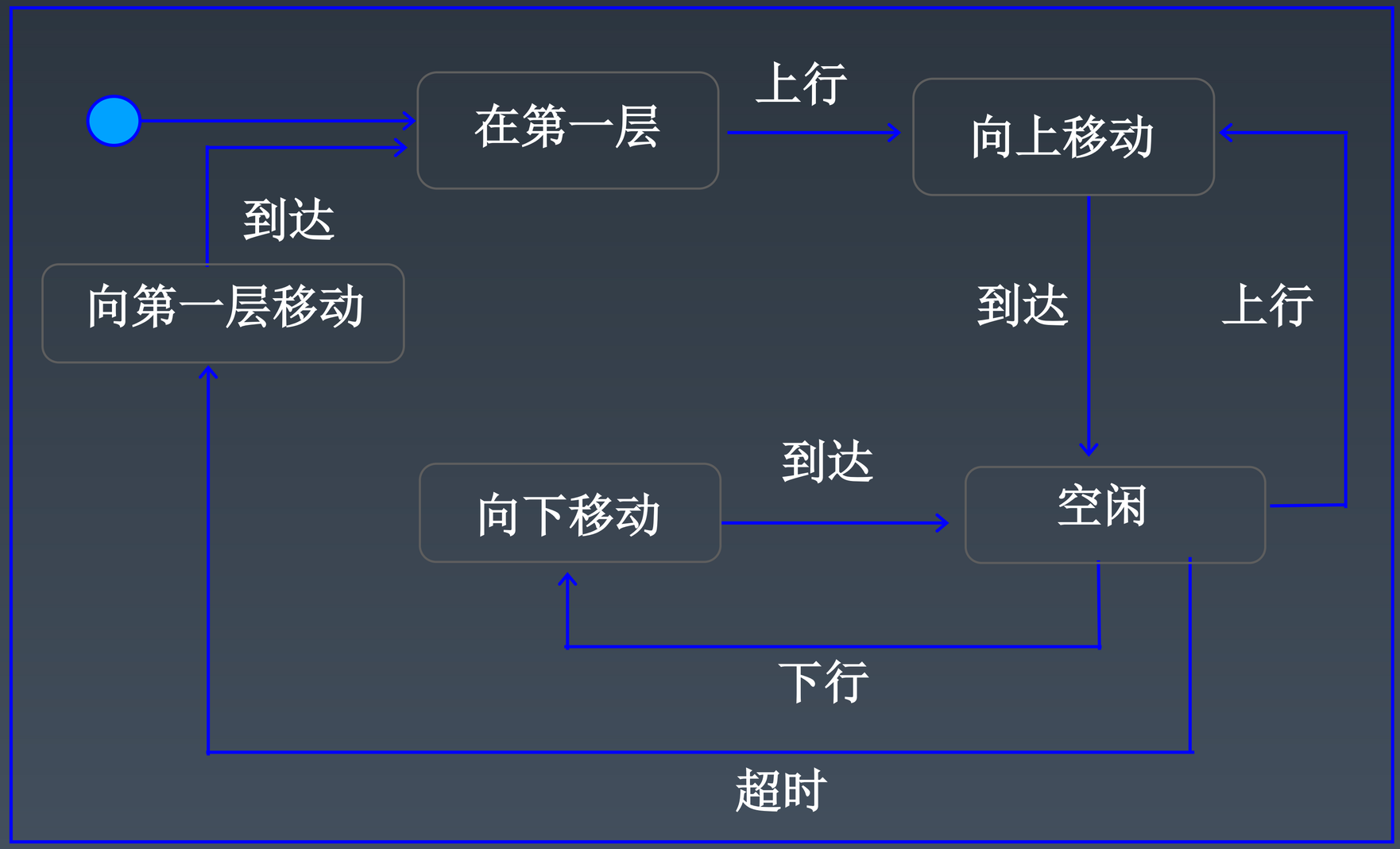
exit事件用于指明退出该状态时的特定动作。

do事件用于指明在该状态中时执行的动作。

无参数

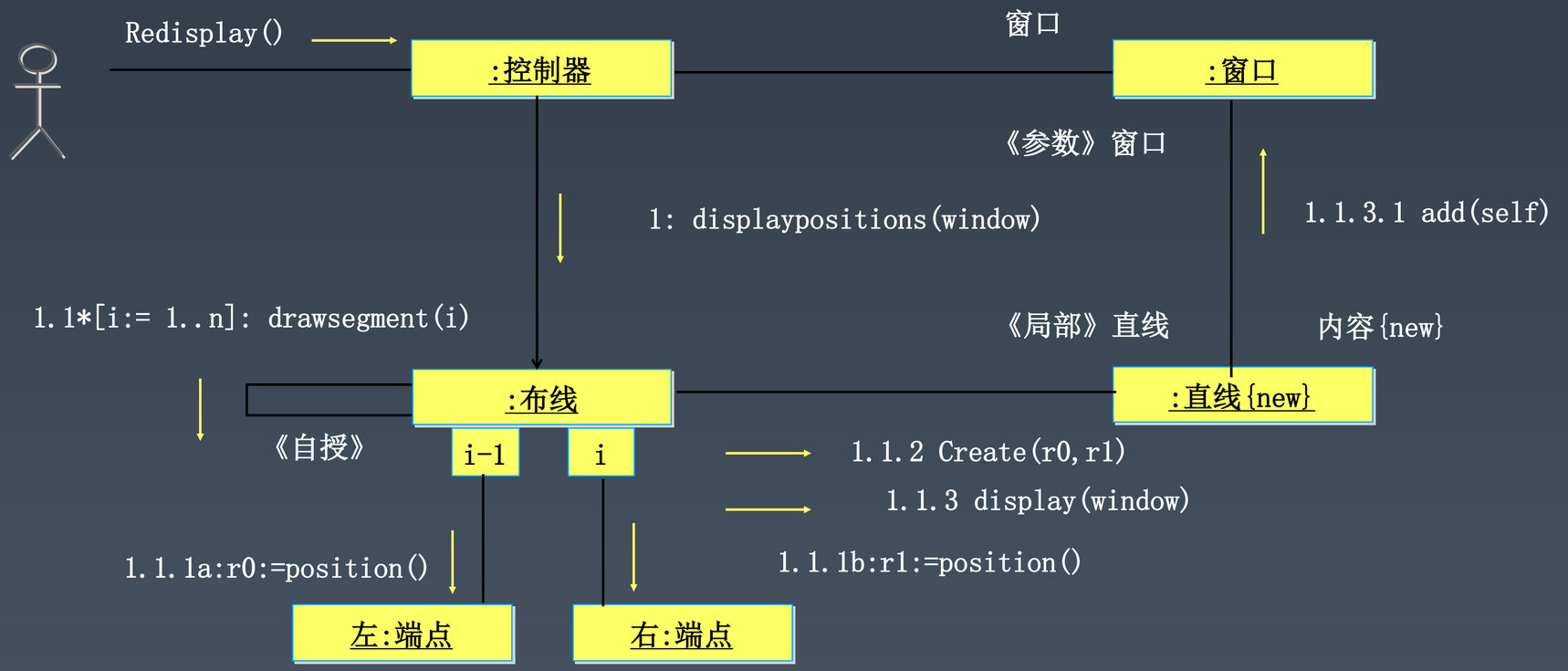
例：

login
login time=curent time
entry/type “login” do/get use name do/get password help/display help exit/login(use_name.password)


在第一层
上行
向上移动
到达
向第一层移动
到达
上行
到达
向下移动
空闲
下行
超时

合作图

合作图（Collaboration Diagram）,也称为协作图，用于描述相互合作的对象间的交互关系和链接（Link）关系。虽然顺序图和合作图都用来描述对象间的交互关系,但侧重点不一样。顺序图着重体现交互的时间顺序,合作图则着重体现交互对象间的静态链接关系。



实现模型

实现模型描述了系统实现时的一些特性，又称为物理体系结构建模。包括源代码的静态结构和运行时刻的实现结构。

实现模型包括：

- 组件图（Component diagram）显示代码本身的逻辑结构，它描述系统中存在的软构件以及它们之间的依赖关系。
- 部署图（Deployment diagram）描述了系统中硬件和软件的物理配置情况和系统体系结构。显示系统运行时刻的结构，部署图中的简单结点是指实际的物理设备以及在该结点上运行构件或对象。部署图还描述结点之间的连接以及通信类型。

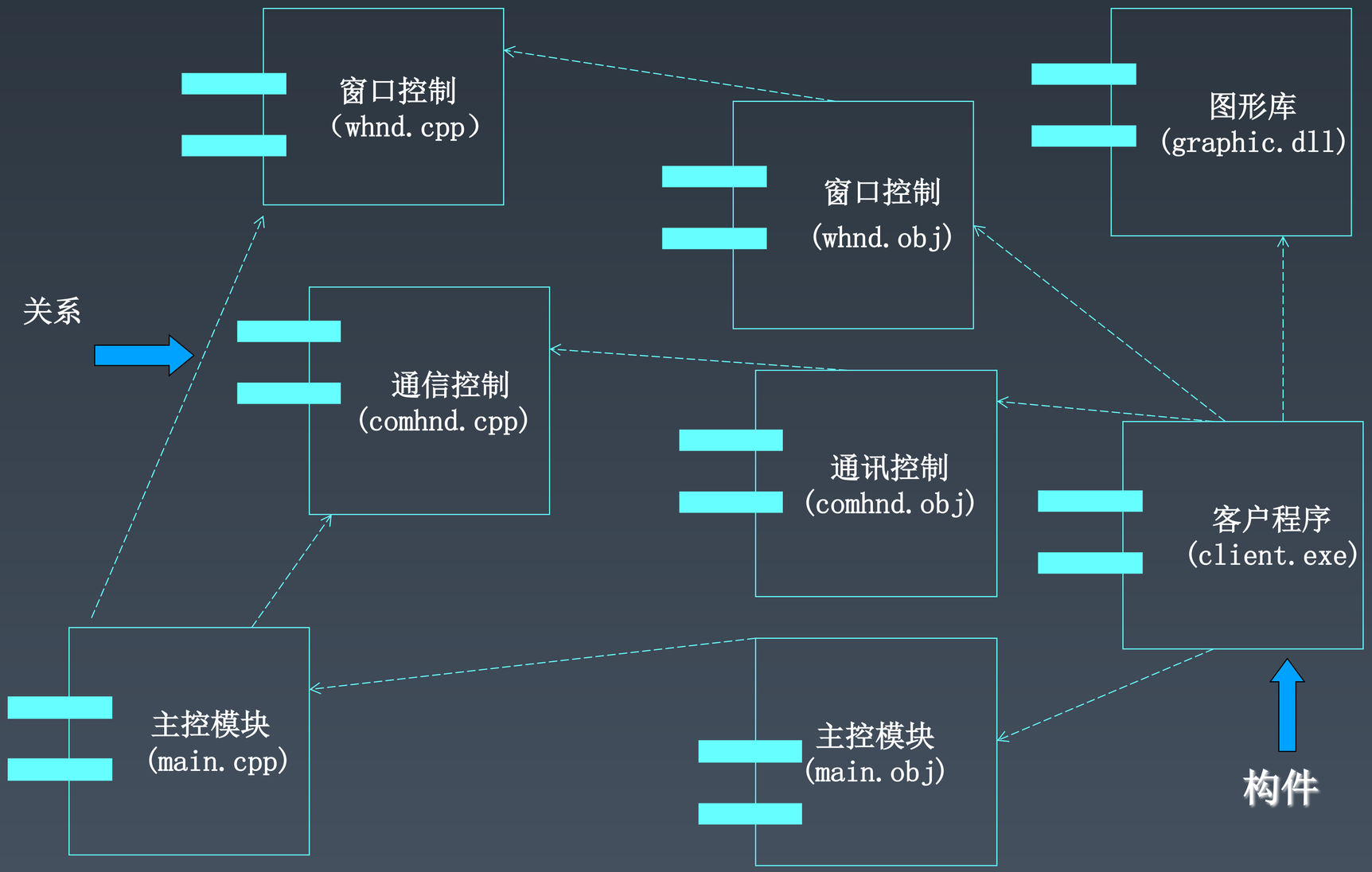
组件图

组件（component）
组件定义：系统中遵从一组接口且提供其实现的物理的、可替换的部分。对系统的物理方面建模时，它是一个重要的构造块。

组件可以看作包与类对应的物理代码模块，逻辑上与包，类对应，实际上是一个文件，可以有下列几种类型的构件：

- 源代码构件
- 二进制构件
- 可执行构件

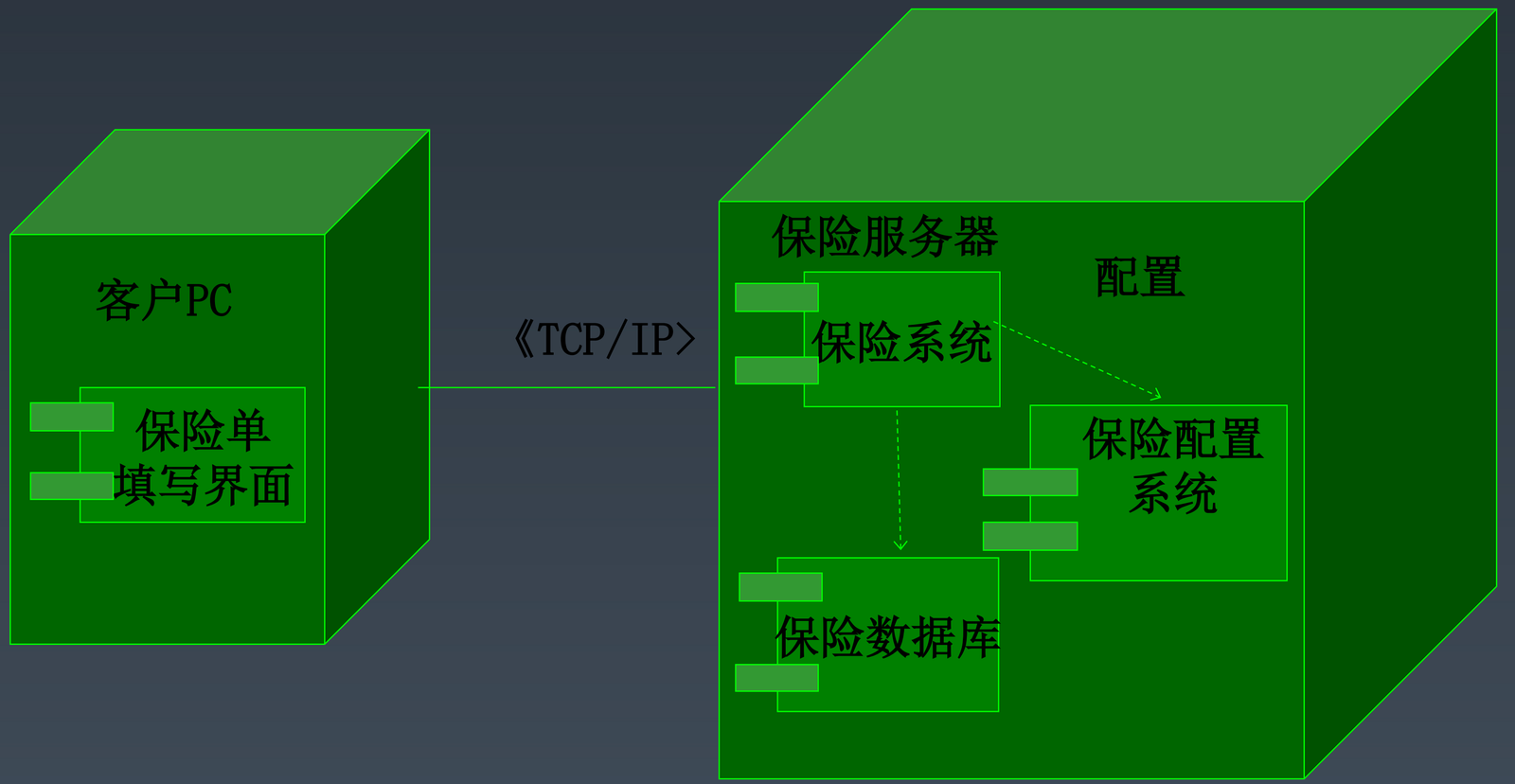
组件之间的依赖关系是指结构之间在编译，连接或执行时的依赖关系。用虚线箭头表示
组件图符：


窗口控制
(whnd.cpp)
窗口控制
(whnd.obj)
图形库
(graphic.dll)
关系
通信控制
(comhnd.cpp)
通讯控制
(comhnd.obj)
客户程序
(client.exe)
主控模块
(main.cpp)
主控模块
(main.obj)
构件

部署图

部署图用来描述系统硬件的物理拓扑结构以及在此结构上执行的软件，即系统运行时刻的结构。部署图可以显示计算机结点的拓扑结构和通信路径，结点上执行的组件，特别对于分布式系统，部署图可以清楚的描述系统中硬件设备的配置，通信以及在各硬件设备上各种软构件和对象的配置。因此，部署图是描述任何基于计算机的应用系统的物理配置或逻辑配置的有力工具，部署图的元素有结点和连接。

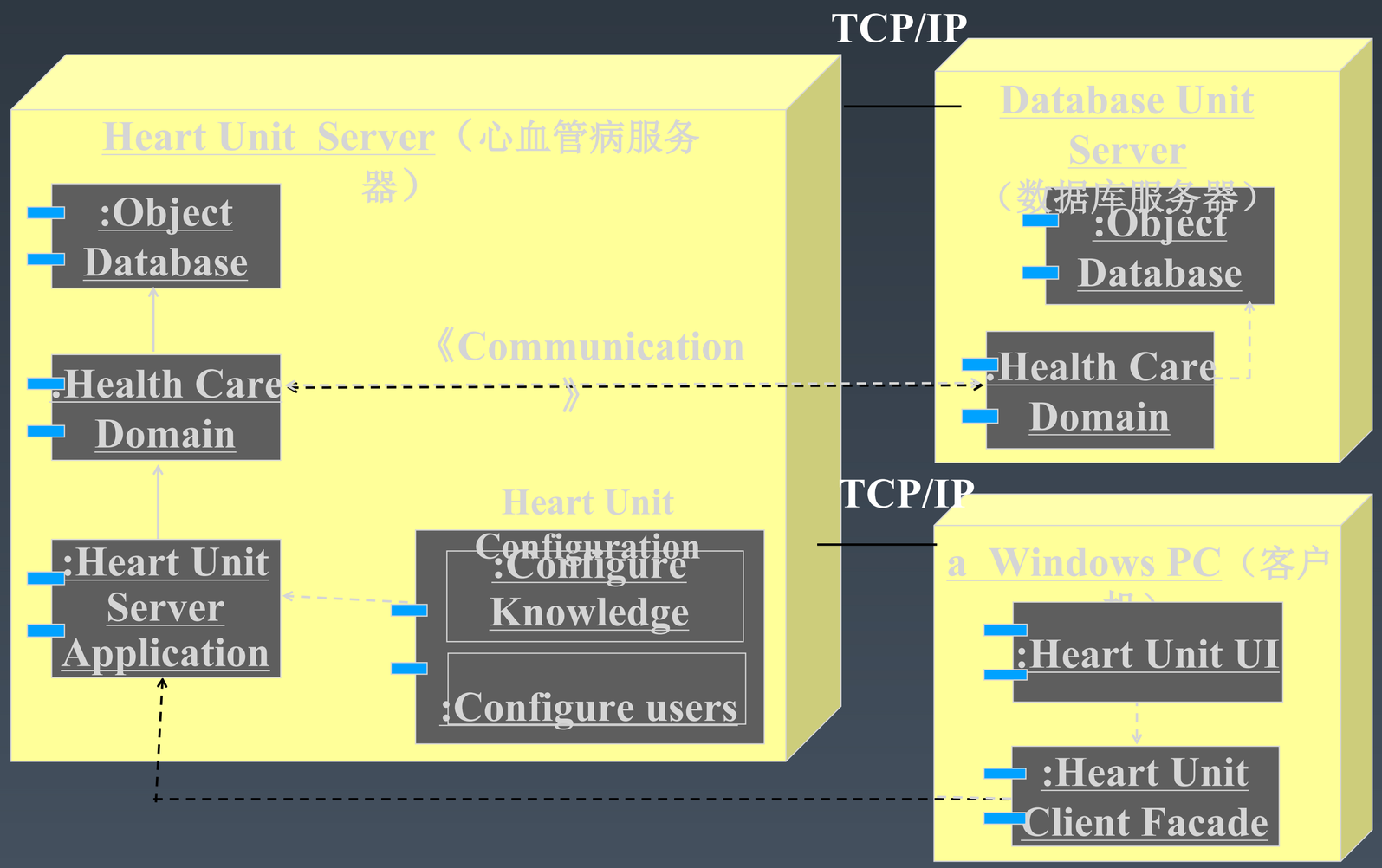
部署图中的结点代表某种计算机，通常是某种硬件。同时结点还包括在其上运行的软组件，软件组件代表可执行的物理代码模块。如一个可执行程序。 结点的图符是一个立方体。



保险系统的部署图

部署图各结点之间进行交互的通信路径称为连接，连接表示系统中的结点存在着联系，用结点之间的的连线表示连接，在连接的连线上标注通信类型。

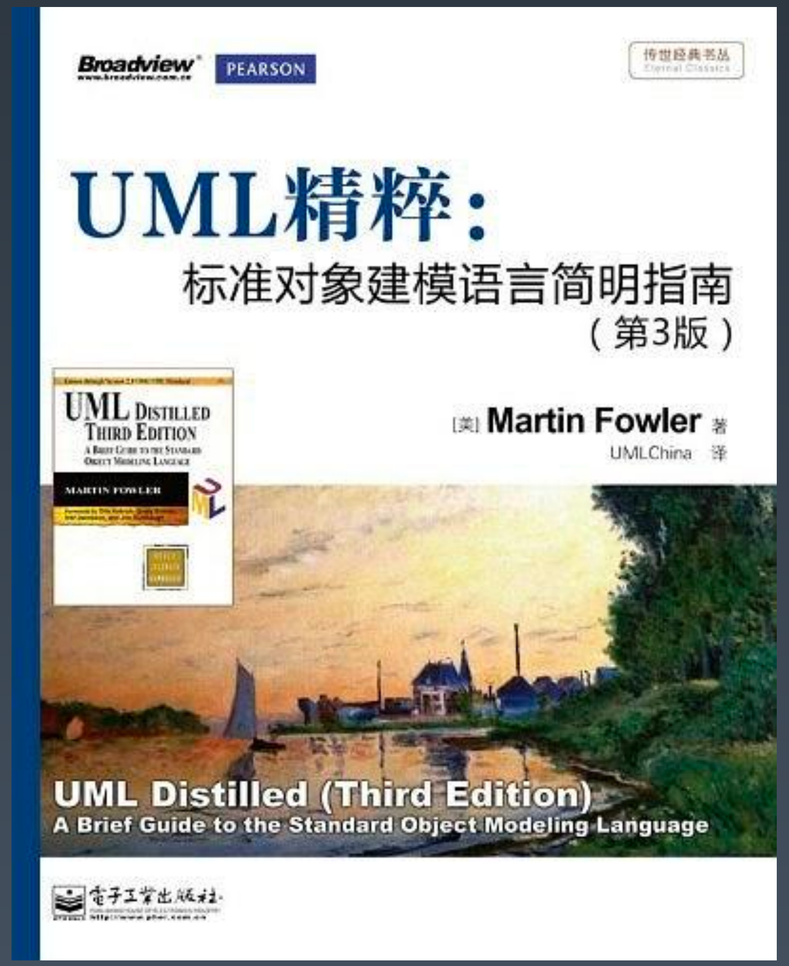
医院诊疗系统的部署图



没有设计文档就没有软件设计

没有软件设计就没有技术进步

参考书目


Broadview
PEARSON
传世经典书丛
UML精粹：
标准对象建模语言简明指南
（第3版）
[美] Martin Fowler 著
UMLChina 译
UML Distilled (Third Edition)
A Brief Guide to the Standard Object Modeling Language
電子工業出版社

THANKS!

《架构师训练营》