**软件体系结构说明书 [SAD]**

目录

[1 引言 1](#_Toc72962834)

[1.1系统概述 2](#_Toc72962835)

[1.2背景 2](#_Toc72962836)

[1.3 文档概述 2](#_Toc72962837)

[1.4基线 2](#_Toc72962838)

[2引用文件 3](#_Toc72962839)

[3总体设计 3](#_Toc72962840)

[3.1基本功能 3](#_Toc72962841)

[3.2运行环境 3](#_Toc72962842)

[3.3基本概念和处理流程 4](#_Toc72962843)

[4各模块结构 5](#_Toc72962844)

[4.1 UI模块 5](#_Toc72962845)

[4.2逻辑处理模块 6](#_Toc72962846)

[4.3网络模块 7](#_Toc72962847)

[5程序设计说明 8](#_Toc72962848)

[5.1 程序系统总述 8](#_Toc72962849)

[5.2运行概念 10](#_Toc72962850)

[6注解 12](#_Toc72962851)

[6.1参考资料 12](#_Toc72962852)

[附录 12](#_Toc72962853)

[1、 软件体系结构学习报告(笔记)： 12](#_Toc72962854)

[2、 保证项目质量属性的设计： 27](#_Toc72962855)

[3、 针对自己的项目，描绘故障树，分解为割集树 27](#_Toc72962856)

# 1 引言

互联网技术的飞速发展和广泛普及，对传统游戏方式产生了深远的影响，相比于传统游戏方式受限于地理位置、表达方式单一的影响，网络游戏以其信息丰富、打破地域限制、互动性强等特点而受到游戏开发人员的关注。

## 1.1系统概述

网上斗地主系统是为了满足广大玩家的游戏需求，能够打破传统斗地主形式的空间限制，满足玩家在不同地理位置能够进行斗地主游戏的需求，实现邀请玩家进入房间以及踢出玩家的功能，并对玩家的的个人信息包括胜负次数，积分等一些信息的维护。

## 1.2背景

开发软件名称：网上斗地主

项目的提出者：山东大学计算机学院18级pog小队

开发方名称：山东大学计算机学院18级pog小队

开发者：山东大学计算机学院18级pog小队

开发负责人：山东大学计算机学院18级pog小队队长

用户：网络玩家

## 1.3 文档概述

通过本项目的体系结构说明，可以深入了解该软件系统的开发内容和开发过程。

## 1.4基线

1.可行性报告 V1.0

2.产品需求规格说明书 V1.0

# 2引用文件

1. 中华人民共和国国家标准GB T-8567-2006

# 3总体设计

## 3.1基本功能

a) 登录功能: 对用户输入的用户名，密码进行验证，验证通过后，该用户可以进入游戏平台，加入游戏房间进行游戏

b) 用户信息维护：对用户的用户名、密码、胜场信息、积分等信息进行维护更新，以便用户登录后进行显示

c) 斗地主游戏：三个玩家根据游戏规则进行自己的选择，具体到自己的出牌阶段，玩家可以选择出牌和不出，系统对玩家的选择进行显示以告知所有玩家。按照出牌规则依次循环至每位玩家的出牌阶段，直至游戏结束。

d) 用户交流：玩家在房间内可以进行闲聊或是针对于出牌情况进行评价，增强玩家之间的交互性。

## 3.2运行环境

硬件环境

A CPU：2.3GHz及以上

B 内存：4G及以上

C 硬盘：10G及以上

D 显示：1280x1024分辨率及以上

软件环境

A Windows 10操作系统

B Visual Studio 2017/2019

## 3.3基本概念和处理流程

该斗地主游戏包括以下几大模块：

1. UI模块

表现游戏内容

注册登录界面

游戏过程展现

给逻辑处理和网络发送信息

接收处理逻辑处理和网络的信息

1. 逻辑处理模块

判断游戏流程正确

抢地主是否成功

出牌是否正确

判断输赢

接收处理UI与网络信息

1. 网络模块

注册登录功能

游戏匹配机制

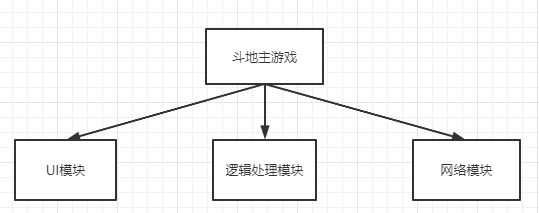
游戏进度保存

用户信息保存

给UI与逻辑发送信息

处理UI与逻辑的信息

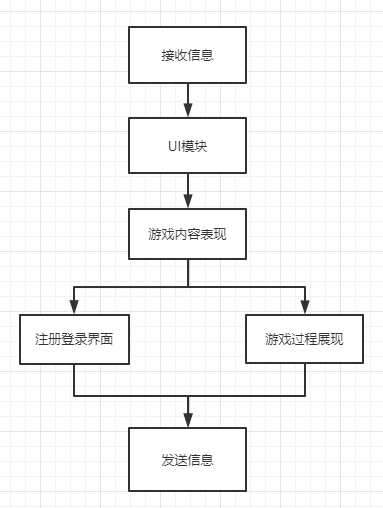
整个项目结构如下图所示：



# 4各模块结构

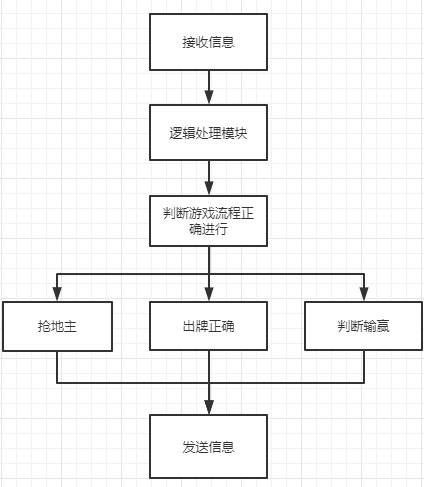
## 4.1 UI模块

这一模块主要是处理游戏与用户交互的代码，它主要负责的是接收逻辑和网络的信息，然后将对应的游戏内容表现出来，例如出牌显示以及输赢显示，以及接收用户的输入。如下图所示：



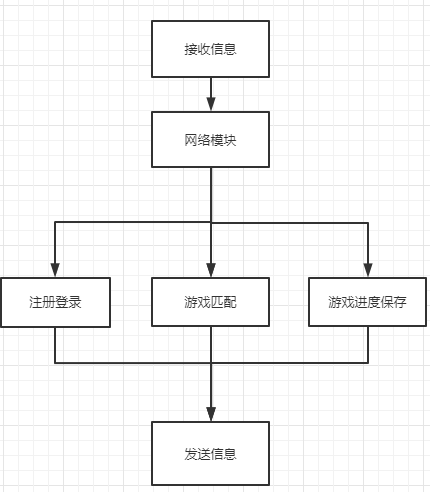
## 4.2逻辑处理模块

这一模块主要是处理游戏业务的代码，比如说整个游戏流程的流程顺序，判断输赢，牌出的对不对，能不能这样出等等。对于这一模块来说，它只是从UI部分和网络部分得到玩家的输入，然后进行游戏的逻辑判断，然后告诉UI或者网络到底发生了什么。如下图所示。



## 4.3网络模块

这一部分主要是考虑网络通信，如何联机的代码，例如注册登录功能、匹配机制等。对于这一模块来说，它从UI部分和逻辑处理部分得到玩家的输入，然后进行一个在线储存，然后告诉UI或者逻辑处理是否执行成功。如下图所示。



# 5程序设计说明

在我们的游戏运行过程中，为便于开发，实现各方面极大程度上的解耦，简化游戏运行模型的设计，同时使得整个系统的运行秉承一个线性的逻辑，我们利用帧的概念，以一帧为游戏运行过程中的最小单位，在时间维度上，帧与帧的切换便展现了游戏的整个运行过程和状态。

## 5.1 程序系统总述

在我们的程序中，我们提出了系统的概念，各类系统属于我们程序运行过程中的运行组件，换句话讲，各类系统正是我们在游戏运行过程中的控制器，在我们的设计概念中，我们共抽象出了六个系统：

input: 一个输入系统，可以为检查按键状态和输入状态。

rebot: 重启系统的集合，开发者编写的重启系统可以通过组件加载到程序中去。

render: 渲染系统的集合，开发者编写的渲染系统可以通过组件加载到程序中去。

resource: 资源系统的集合，开发者可以通过资源系统添加新的资源或者是获取其他资源（例如其他系统的资源），也可以实现多系统之间的交互；

update: 更新系统的集合，开发者编写的更新系统可以通过组件加载到程序中去。

windows: 窗口系统的集合，开发者可以获取窗口的一些信息。

整个程序的架构图如下：

Runtime-service

## 5.2运行概念

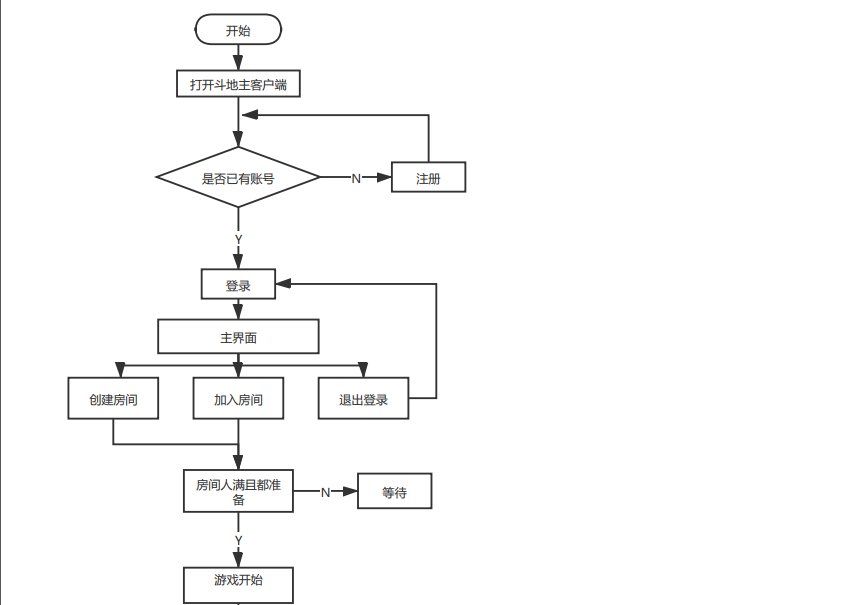
框架的运行本质是顺序运行多个系统，例如运行重启系统加载程序初始化的资源，例如运行更新系统去检查当前的程序状态，更新程序状态(比如说处理游戏的逻辑)，运行渲染系统将我们的图形界面渲染到窗口。

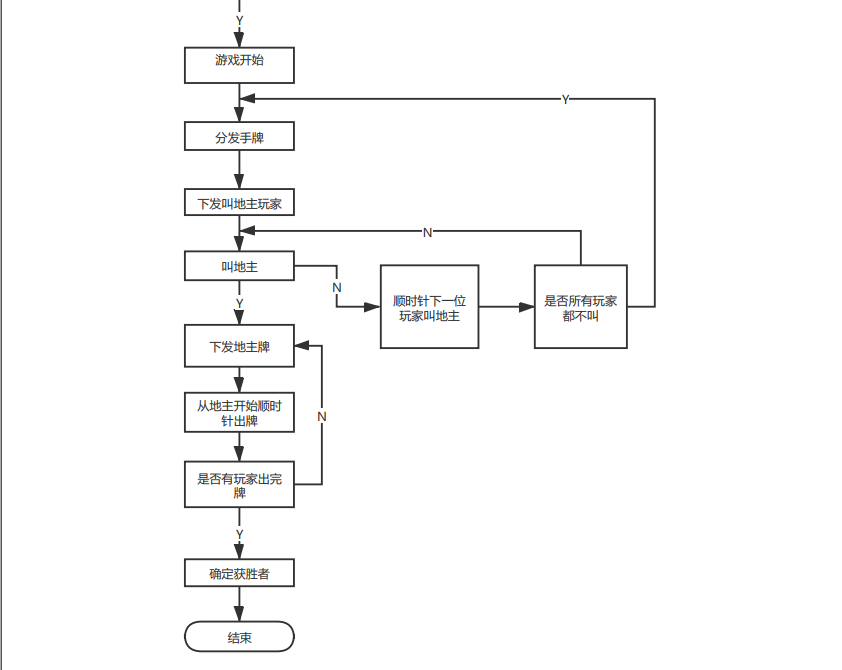
在每一帧中，框架会通过运行时组件按照顺序和优先级调用更新系统和渲染系统，即它们的update函数。而对于重启系统，将会在程序进入主循环之前调用，用来加载程序(对于重启系统来说，最简单的实例就是通过重启系统读取配置文件加载需要的系统)。

对于我们游戏的运行，核心部分在于更新系统，更新系统是驱动程序运行的核心组件，在这个过程中，我们实现逻辑和ui上的解耦，优先执行的是我们的游戏逻辑，用来接收我们的输入以及判断该如何去做，以及告诉我们的UI系统该如何显示我们的界面，然后执行的是我们的UI系统，接收来自逻辑系统的指示，调用ImGui渲染对应的图形界面，UI系统也可以接收用户的输入，然后告知逻辑系统。在这两种系统之间，我们可以通过运行时组件，例如资源系统来存放资源进行交互，比如说消息队列。

值得注意的是，我们上面所描述的六类系统是从整个结构上概括来讲的，而对于每一类系统，其中包含着属于自己的子系统，准确地讲，上述六类系统分别属于一个系统集合，其中包含了整个程序中所有属于该类性质系统的子系统。Runtime-service是我们的一个运行实例，属于一个runtime\_service类型，作为参数传递给开发者（当开发者 开发一个system的时候）。在整个开发模式中，我们主要采用面向对象式的编程思想，对于每一种系统，我们提供了不同的成员函数对其所有数据进行操作，以实现不同的功能。

运行时的大致流程如下：





# 6注解

本文档为《软件工程》课程的实验项目文档，希望通过《网上斗地主系统》的设计开发，了解软件文档的相关标准与编写原则，训练并掌握软件各类文档的写作技巧。

## 6.1参考资料

《GB-T-8567-2006计算机软件文档编制案例》

《软件(结构)设计说明(SDD)》

# 附录

1. **软件体系结构学习报告(笔记)：**

**杨成豪：**

**软件体系结构的研究意义学习报告**

软件体系结构是软件系统的高级抽象，往往体现了系统开发中最早做出的决策，它体现了根本性的系统设计思路，对系统起着最为深远的影响。体系结构在明确了系统的各个组成部分的同时，也限定了各部分间的交互方式，这将进一步影响到开发资源的配置和开发团队的组织等其他方方面面的开发活动，并影响着最终的软件产品质量。在大型软件系统中，软件体系结构是决定系统能否顺利实现的关键因素之一，不当的体系结构会给整个系统带来灾难性的后果。良好的体系结构对于软件系统的重要意义在软件生命周期中的各个阶段都有体现。

对系统分析的意义：

在系统分析阶段，软件体系结构发挥着巨大作用。一方面借助于软件体系结构进行描述，可以使问题得以进一步抽象，使整个系统更易于被系统分析设计人员把握，更清晰地认识系统，完善对系统的理解。除此之外，体系结构对于系统分析带来的优势还体现在，它为系统分析设计人员提供了新的思路。比如，在更高层次上进行系统一致性检查、使用成熟的软件体系结构风格等。另一方面，它能够帮助软件系统的各有关权益方(客户、用户、项目管理人员、设计开发人员以及测试人员等)形成统一认识，互相交流。

交流是软件开发的重要组成部分。在软件开发活动中，各有关权益方承担着不同角色，关注同一软件系统不同的侧面，这要求他们要从不同的角度交流对共同面对的软件系统的认识。 例如用户关心系统是否满足可用性及可靠性需求，客户关心此结构能否按期按预算实现，管理人员担心在经费支出和进度条件下，按此体系结构能否使开发组成员在一定程度上独立开发，并有条不紊地有序地交互，开发人员关心达到全部目的的策略。

对于软件开发的意义：

软件体系结构代表了系统早期的设计决策：与开发、设计、编码或运行服务及维护阶段相比，早期设计决策的处理难度最大，对系统的生命期的影响也最大同时，软件体系结构也难于改变，会对整个系统开发活动造成深远影响；软件体系结构是系统实现的基本约束：即系统的后继开发工作要遵循体系结构所描述的设计决策。每个构件或连接件必须满足体系结构规格说明中指定的功能、语义和接口，并且按体系结构配置中所规定的方式完成交互。这样，构件或连接件的开发人员在体系结构给定的约束下进行工作，他们既不关心其他构件或连接件的开发，也不会对其产生影响。而体系结构设计者也不必设计算法或精通编程语言；软件体系结构对于软件质量控制有重要意义：软件质量特性可分为两类，第一类是可以通过运行软件并观察其效果来度量的特性，如功能、性能、安全性及可靠性等，第二类是指那些无法通过观察系统来度量，只能通过观察开发活动或维护活动来考察的特性，包括各种可维护性问题，如可适应性、可移植性、可重用性等,(例如，可重用性依赖于系统中的构件与其他构件的联系情况)。软件体系结构在很大程度上确定了系统是否能达到其需求的质量特性。

对软件重用的意义：

重用是提高软件开发效率、保证软件质量的重要手段：软件开发经历了从机器语言、汇编语言、结构化程序设计语言、面向对象程序设计语言、形式化(非形式化)规格说明语言(如体系结构描述语言)的发展过程，越来越适合开发人员的思维活动模型，代码重用的级别也在不断地提升。体系结构技术的研究，使软件重用从代码重用发展到设计重用和过程重用，实现多层次的软件重用；构件的重用是体系结构良好的软件系统最基本的一点：面向体系结构的开发方法常常注意构件的组合与装配，而不一定把编程作为主要活动。有效地利用标准构件，或是识别并重用系统内部的构件，或是购买第三方构件，只要这些构件与当前体系结构相容，都能减少开发中的重复劳动和系统中的重复代码。体系结构起了组织产品的构件、接口及运行的作用。这里要着重指出的是标准构件的应用。应用标准构件库的关键在于要能够从整体上对库中构件进行把握。一旦做到了这点，就可以快速、灵活地在构件库中选择出合适的构件应用到系统中去；反之，构件的选取就只能通过反复地浏览来寻找，这实际上影响到了体系结构所带来的优势，造成了不必要的资源浪费。

对系统演化的意义：

对软件系统的演化过程中，维护人员需要不断地进行调整、修改、增加新的功能或构件等工作：通常情况下，软件系统的开发成本中，有80%是在初次投入使用之后产生的，因此，解决好系统演化阶段的开发问题具有重要意义；软件体系结构决定着系统构件的划分和交互方式，一方面在设计系统的体系结构之初，就应当充分考虑到将来可能的系统演化，另一方面在进行系统演化阶段的开发时，由于体系结构充分地刻画了当前系统，清晰地描述了构件及其相互关系和整个系统的框架，所以应当充分利用。

**侯旋：**

软件体系结构学习报告(笔记)

**软件体系结构：**

**概念：**是在处理算法与数据结构之上的，关于整体系统结构设计和描述方面的问题。是软件系统的一个结构、行为和熟悉的高级抽象。

**组成部分：**

1. 构件：一组代码，或程序块或一个独立的程序。
2. 连接件：关系的抽象，用以表示构件之间的相互作用。（如过程调用、管道等）
3. 配置：用于对构件与连接件的语义说明。
4. 端口：表示构件与外部环境的交互点。
5. 角色：连接件的接口。

**软件体系结构的意义：**

1. 是系统开发中不同参与者进行交流和信息传播的媒介。
2. 是早期设计决策的体现。
3. 可以反映软件的的质量以及瓶颈。
4. 本身也是一种可传递可重用的模型。

**软件体系结构模型：**

1. 结构模型：最直观最普遍的建模方法，通过体系结构的构件、连接件和其他概念来刻画结构；力图通过结构，来反映系统的重要语义内容。
2. 框架模型：结构模型类似，但不侧重结构的细节，而侧重整体的结构。主要以一些特殊的问题作为目标，建立只针对和适应这类问题的结构。
3. 动态模型：对结构或框架模型的补充，研究系统的“大粒度”的行为及其性质。如系统总体结构的配置或演化、通信通道或计算过程的建立或拆除等方面的行为。
4. 过程模型：研究构造系统的步骤与过程。
5. 功能模型：体系机构的一组功能构件，按照层次组成，下层向上层提供服务，可以看作为一种特殊的框架模型。

**常见软件体系结构风格**

1. 管道过滤器风格：主要包括过滤器和管道两种元素。在这种结构中，构件被称为过滤器，负责对数据进行加工处理。每个过滤器都有一组输入端口和输出端口，从输入端口接收数据，经过内部加工处理之后，传送到输出端口上。数据通过相邻过滤器之间的连接件进行传输，连接件可以看作输入数据流和输出数据流之间的通路，这就是管道。
2. 面向对象风格：在这种体系结构中，数据表示和相关原语操作都被封装在抽象数据类型中。在这种风格中，对象是构件，也成为抽象数据类型的实例。对象与对象之间，通过函数调用和过程调用来进行交互。
3. 事件驱动体系结构风格：系统对外部的行为表现可以通过它对事件的处理来实现。在这种体系结构中，构件不再直接调用过程，而是声明事件。系统其他构件的过程可以在这些事件中进行注册。当触发一个事件的时候，系统会自动调用这个事件中注册的所有过程。因此，触发一个事件会引起其他构件的过程调用。
4. 分层体系结构风格：在分层风格中，系统将划分为一个层次结构。每一层都具有高度的内聚性，包含抽象程度一致的各种构件，支持信息隐藏。分层有助于将复杂系统划分为独立的模块，从而简化程序的设计和实现。通过分解，可以将系统功能划分为一些具有明确定义的层，较高层是面对特定问题，较低层具有一般性。每层都为上层提供服务，同时又利用了下层的逻辑功能。在分层体系结构中，每一层只对相邻层可见。层次之间的连接件是协议和过程调用。用以实现各层之间的交互。
5. 客户端-服务器风格：这种模式由两方组成;一个服务器和多个客户端。服务器组件将向多个客户端组件提供服务。客户端向服务器请求服务，服务器向这些客户端提供相关服务。此外，服务器继续侦听客户端请求。

**许振科：**

界面设计是为了满足软件专业化标准化的需求而产生的对软件的使用界面进行美化优化规范化的设计分支。具体包括软件启动封面设计，软件框架设计，按钮设计，面板设计，菜单设计，标签设计，图标设计，滚动条及状态栏设计，安装过程设计，包装及商品化。

软件界面设计的三大原则是：置界面于用户的控制之下、减少用户的记忆负担、保持界面的一致性。一般来讲，软件界面设计在工作流程上可分为结构设计、交互设计、视觉设计三个部分。

**一、结构设计（Structure Design）**

结构设计也称概念设计 (Conceptual Design)，是界面设计的骨架。通过对用户研究和任务分析，制定出产品的整体架构。基于纸质的的低保真原型(Paper Prototype)可提供用户测试并进行完善。在结构设计中，目录体系的逻辑分类和语词定义是用户易于理解和操作的重要前提。如西门子手机的设置闹钟的词条是“重要记事”，让用户很难找到。

**二、交互设计（Interactive Design）**

交互设计的目的是使产品让用户能简单使用。 任何产品功能的实现都是通过人和机器的交互来完成的。因此，人的因素应作为设计的核心被体现出来。交互设计的原则如下：

* 有清楚的错误提示。误操作后，系统提供有针对性的提示。
* 让用户控制界面。“下一步”、“完成”，面对不同层次提供多种选择，给不同层次的用户提供多种可能性。
* 允许兼用鼠标和键盘。同一种功能，同时可以用鼠标和键盘。提供多种可能性。
* 允许工作中断。例如用手机写新短信的时候，收到短信或电话，完成后回来仍能够找到刚才正写的新短信。
* 使用用户的语言，而非技术的语言。
* 提供快速反馈。给用户心理上的暗示，避免用户焦急。
* 方便退出,如手机的退出，是按一个键完全退出，还是一层一层的退出。提供两种可能性。
* 导航功能。随时转移功能，很容易从一个功能跳到另外一个功能。

**三、视觉设计（Visual Design）**

在结构设计的基础上，参照目标群体的心理模型和任务达成进行视觉设计。包括色彩、字体、页面等。视觉设计要达到用户愉悦使用的目的。视觉设计的原则如下：。

* 界面清晰明了。允许用户定制界面。
* 减少短期记忆的负担。让计算机帮助记忆，例：User Name,、Password、IE进入界面地址可以让机器记住。
* 依赖认知而非记忆。如打印图标的记忆、下拉菜单列表中的选择。
* 提供视觉线索。图形符号的视觉的刺激;GUI(图形界面设计)：Where, What, Next Step。
* 提供默认(default)、撤销(undo)、恢复(redo)的功能
* 提供界面的快捷方式。
* 尽量使用真实世界的比喻。如：电话、打印机的图标设计，尊重用户以往的使用经验。
* 完善视觉的清晰度。条理清晰;图片、文字的布局和隐喻不要让用户去猜。
* 界面的协调一致。如手机界面按钮排放，左键肯定;右键否定;或按内容摆放。
* 同样功能用同样的图形。

色彩与内容。整体软件不超过5个色系，尽量少用红色、绿色。近似的颜色表示近似的意思。

**李子任：**

**体系结构的应用场景**

首先需要承认的是，在软件设计的不同阶段需要不同的设计方法。面对不同粒度，不同复杂度的系统，你需要使用不同的思考框架。

体系结构设计与算法设计就需要不同的思考框架。

在**算法设计**中，动态规划、贪心算法等这些算法思想是非常重要的思考方式/思考框架，他们在一定程度上能够保证快速得到计算的结果。它们面临的场景更多类似于“快速在某个字符串中快速寻找某个子串”，这样一些小问题。

算法虽然能够攻克粒度较小的问题，但是在构建系统的时候，往往面临的问题是：**软件系统规模大且复杂性高**。这时对系统的全局结构设计和规划更加重要。比如说，给你一个需求，设计一个图书管理系统，这时候动态规划这些思考框架就显得有些力不从心了。这时候，**体系结构设计**就应运而生了，**体系结构设计是一种构建复杂系统时的思考框架**，注意它服务的对象是粗粒度的复杂系统，所以你也不要奢望它能对你的算法设计有太大的帮助。（正如你不要期待国家的高层，用它的思考方式去解决 如何耕地的细节问题）

**理解软件体系结构**

软件体系结构（Software Architecture）包括构成系统的设计元素的描述、 设计元素

之间的交互、 设计元素的组合模式以及在这些模式中的约束。

理解上述抽象概念并不是一件容易的事情，需要一定的工程经验。这里简单谈谈自己的理解。

理解的角度1：（集合角度）

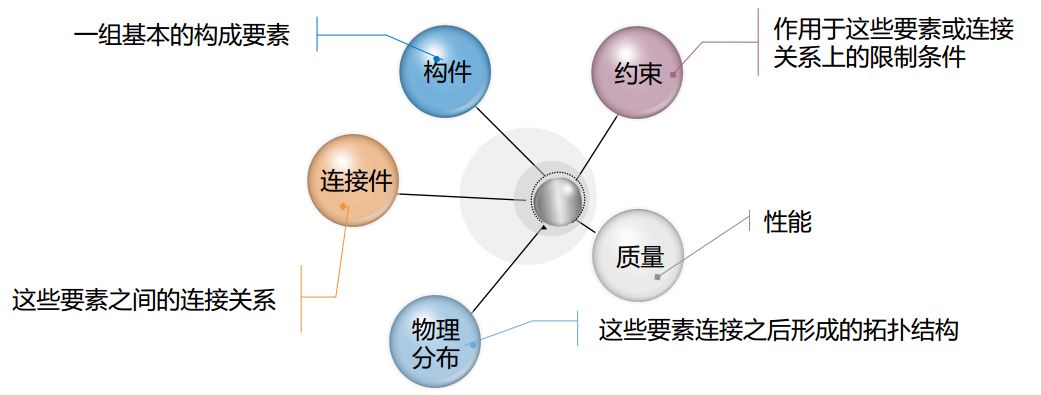
在计算机中，集合是我们一个有利的工具，我们定义事物，理解事物都可以从中获得启发。在集合中，你需要规定基本的元素（这里指构件这些要素），同时还需要制定这些元素的联系方式（这里是指元素的组合模式，及模式中的约束），适当的时候你还需要对具有某些特殊性质的元素进行分类（这里根据元素的作用，将元素分为了两大类：构件 & 连接件）

接下来，你可以从利用常规的集合理论理解该思考框架了。

理解的角度2：（实际意义）

我们人类解决复杂问题的方式，基本都是采用分解的方式，即分而治之。所以给定一个系统的时候，我们尝试将其分解成不同的部分，按照一定的划分标准（比如，功能作用），我们可将其分出实际进行处理的模块（构件、连接件），以及负责连接（负责通信）的要素（分布，约束）等。

举个简单的例子，以 “人体”这个系统为例，根据人体各部分的功能，我们可将人体分为不同的构件：头、手、中间部位身体、腿。但是只具备这些要素还是不够的，他们必须连起来，相互配合才能形成真正的人体。



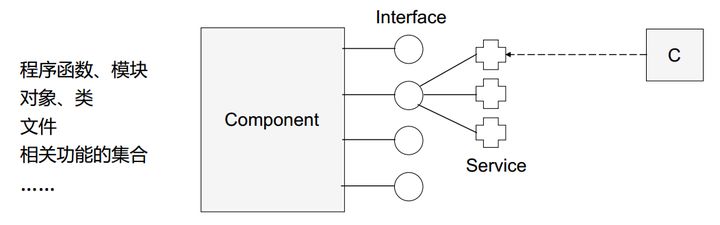
简而言之，**软件体系结构 = 构件 + 连接件 + 约束**。

接下来，进一步说明其他概念。

**构件**

构件是具有某种功能的可复用的软件结构单元，表示系统中主要的**计算元素**和 **数据存储**。

构件是一个抽象的概念，在程序中可以指程序函数、模块、对象、类等。

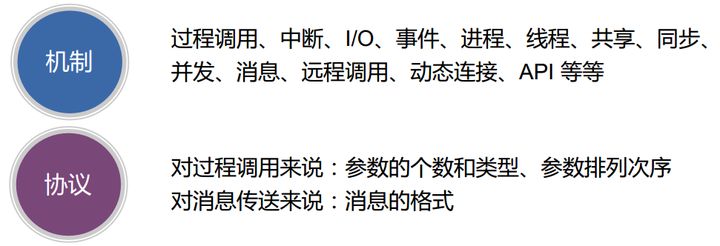


**连接件**

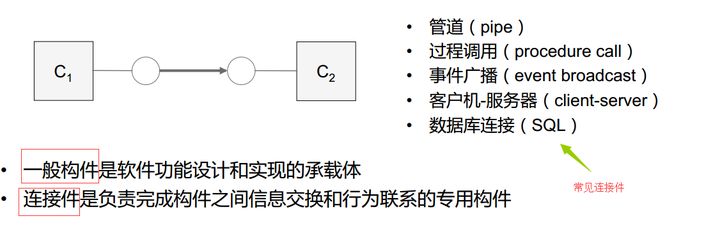
连接是构件间建立和维护行为关联与信息传递的途径。连接包含下面两种要素：

其中，机制指的实际中的**消息传递方式**。

而协议则决定了 **消息的语义理解**。



连接件表示构件之间的交互并实现构件之间的连接。



**软件体系结构目标**

为了更好理解后面的软件体系结构涉及的原则和体系结构风格，**请牢记这些目标**，时不时的对照后面的内容回顾这些目标。

**所有的设计原则等理论基本上都可以映射到下面一个或几个目标上**。



**体系结构的发展**

现在软件的复杂性及多变性，导致了软件粒度越来越粗，越来越开放。

**软件设计的原则**

设计原则是系统分解和模块设计的基本标准，应用这些原则可以使代码更加灵活、 易于维护和扩展。这里提到的设计原则包括以下几点：



需要指出的是，这些原则有一些交叉的部分，并非完全独立没有交集的，理解这一点，你将能更清楚细微的区别。比如：

* 通过封装来实现模块化
* 通过将抽象程度相同的模块放在同一层次，形成层次化的结构。
* 另外，层次化可以理解为模块化的特例

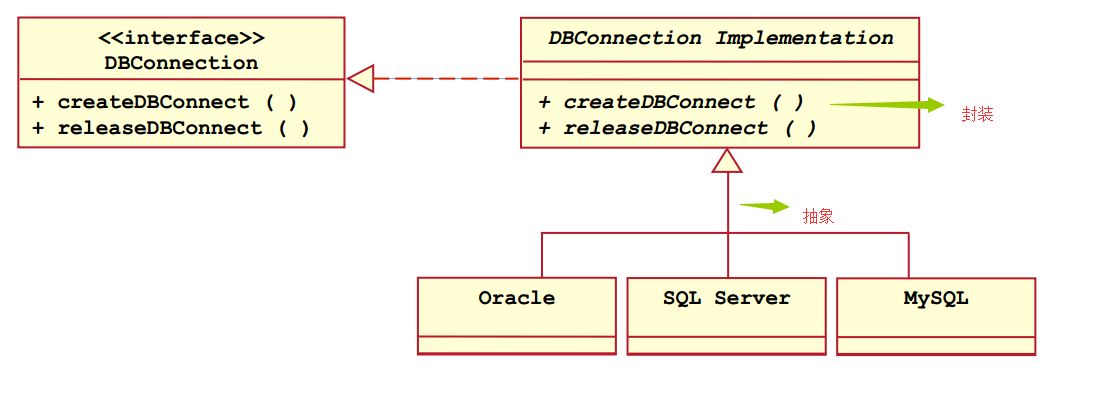
接下来分别介绍上述原则：

**抽象**

抽象是关注事物中与问题相关部分而忽略其他无关部分的一种思考方法。

待抽像的例子：

抽象后：

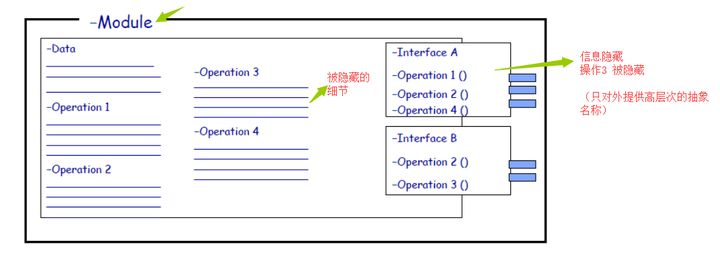


**封装**

封装和信息隐藏是指每个软件单元对其他所有单元都隐藏自己的设计决策，各个单元的特性通过其外部可见的接口来描述。

【要求】：**应将单元接口设计得尽可能简单，并将单元对于环境的假设和要求降至最低**。

举例：



**模块化**

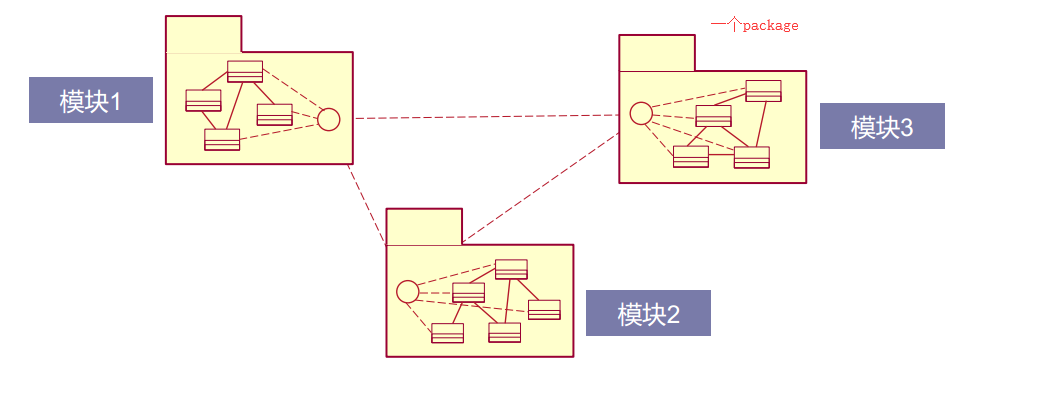
模块化是在**逻辑**和**物理**上将整个系统分解成多个更小的部分，其实质是“分而治之” ，即将一个复杂问题分解成若干个简单问题，然后逐个解决。

（一般是先进行逻辑模块化，随后进行物理分配）

举例：

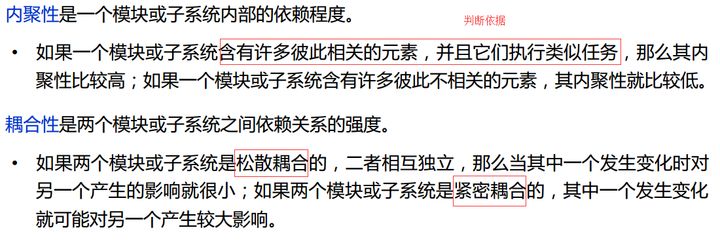
下面的模块化，主要是指逻辑上的模块化，并为涉及到如何将这些模块如何部署到物理机上的过程。

（在物理模块化的时候，可能将模块1和模块3放在同一台机器上，而模块2放在另一台机器上）



**系统分解**

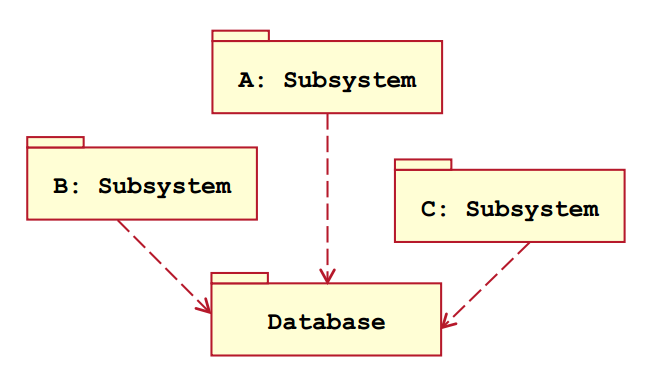
模块化，不可避免的一个问题是，对于系统如何进行分解。在大量实践中，人们总结出了两个衡量的原则：高内聚、 低耦合。



举个简单的例子：

在一个软件系统中，有三个子系统A、 B、 C都要访问一个关系数据库。

设计方案1：（高耦合方案）



设计方案2：（增加隔离层）

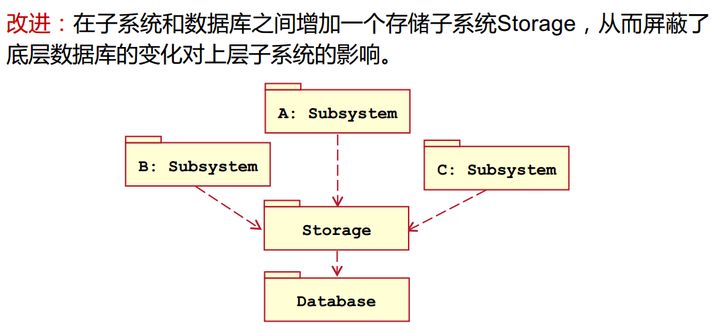
说说这种方案的好处。

设计1中，如果数据库结构发生变化，那么你需要分别在A，B，C中分别进行更改，这样分散的修改，特别容易造成错误。而在设计2中，只需要修改Storage层就可以了。

另外，在A，B，C系统中可能有一些共同的读数据库操作，比如三个系统中都有ReadAnimal（）这个操作，设计1中需要分别在A,B，C中进行修改，而设计2中将该共用方法提到Storage模块中，从而修改时，只需要进行一次修改。

**PS**:

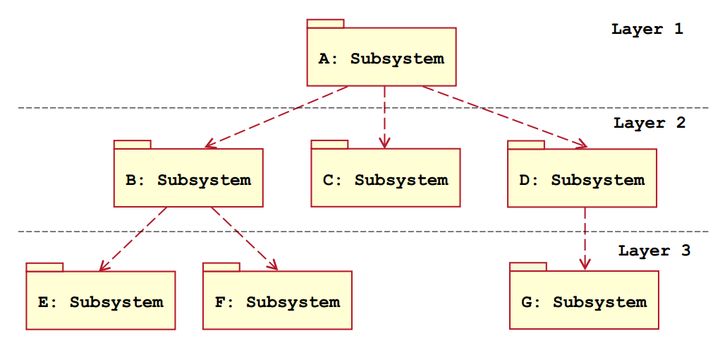
这里加入Storage，是因为假设3个系统中有一些共同的部分。但是如果子系统没有任何共享的数据库数据（每个表被一个系统所独占），那么去掉Storage层也没有关系，因为数据库某个表的变化只会影响到一个子系统。



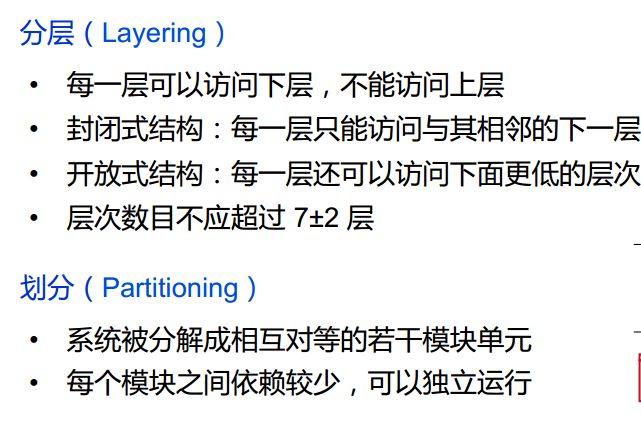
**层次化**

先给出层次化的表现形式：

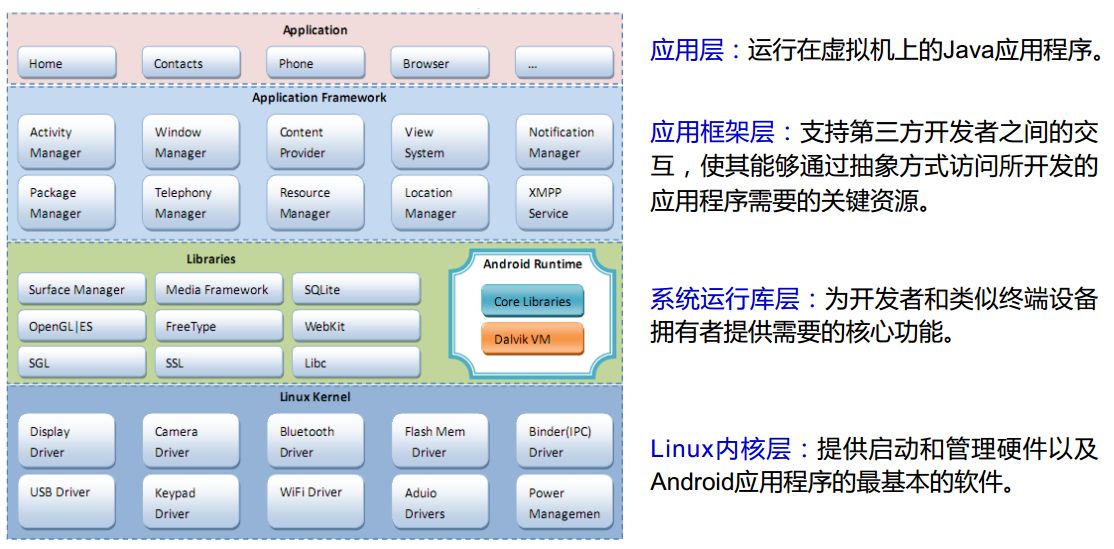
个人感觉，**层次化是模块化的一种特例**。在模块化的基础上，加入了一些限制。在层次化结构中，layer可以作为一种特殊的模块（包含特定的子系统集合），在这类模块（layer）的联系中，**规定了layer模块只能被设计成类似于双向链表的顺序结构**，而不能出现树状结构或者图状结构。



层次化的一些说明：



举例：Android操作系统



**复用**

复用（Reuse）是利用某些已开发的、 对建立新系统有用的软件元素来生成新的软件系统，其好处在于提高生产效率，提高软件质量。

* 源代码复用：对构件库中的源代码构件进行复用

• 软件体系结构复用：对已有的软件体系结构进行复用

• 框架复用：对特定领域中存在的一个公共体系结构及其构件进行复用

• 设计模式：通过为对象协作提供思想和范例来强调方法的复用

**李坚：**

https://www.zhihu.com/question/39261991

http://www.pearsonhighered.com/educator/academic/product/0,3110,0321127420,00.html

https://www.jianshu.com/p/857f49ef027f

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。

处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组合连接起来。

这一定义注重区分处理构件、数据构件和连接构件，这一方法在其他的定义和方法中基本上得到保持。

体系结构问题包括总体组织和全局控制，通讯协议，同步，数据存取。

软件体系结构处理算法与数据结构之上关于整体系统结构设计和描述方面的一些问题。

基于构件的软件开发优势是降低开发成本，缩短开发时间。劣势是兼容性问题，独创性缺失从而竞争力下降。对构件提供者的依赖性。

**王惠中：**

软件体系结构风格（Architectural Styles）是描述特定系统组织方式的惯用范例，强调了软件系统中通用的组织结构。

一个软件体系结构风格定义了构件和连接件类型的符号集，及规定了它们怎样组合起来的约束集合。

不同的架构风格分别适用于特定领域：

**通信：**SOA，消息总线，管道和过滤器

**部署：**客户端/服务器，三层架构，N层架构

**领域：**领域模型，网关

**交互：**分离表现层

**结构：**基于组件的架构，面向对象，分层架构

介绍几种常见的架构风格：

**管道和过滤器风格：**

在管道/过滤器风格的软件体系结构中，每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了。因此，这里的构件被称为过滤器，这种风格的连接件就像是数据流传输的管道，将一个过滤器的输出传到另一过滤器的输入。此风格特别重要的 过滤器必须是独立的实体，它不能与其它的过滤器共享数据，而且一个过滤器不知道它上游和下游的标识。一个管道/过滤器网络输出的正确性并不依赖于过滤器进 行增量计算过程的顺序。

**数据抽象与面向对象风格：**

抽象数据类型概念对软件系统有着重要作用，目前软件界已普遍转向使用面向对象系统。这种风格建立在数据抽象和面向对象的基础上，数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中。这种风格的构件是对象，或者说是抽象数据类型的实例。对象是一种被称作管理者的构件，因为它负责保持资源的完整性。对象是通过函数和过程的调用来交互的。

**基于事件的隐式调用风格**

基于事件的隐式调用风格的思想是构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其它构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。

从体系结构上说，这种风格的构件是一些模块，这些模块既可以是一些过程，又可以是一些事件的集合。过程可以用通用的方式调用，也可以在系统事件中注册一些过程，当发生这些事件时，过程被调用。

基于事件的隐式调用风格的主要特点是事件的触发者并不知道哪些构件会被这些事件影响。这样不能假定构件的处理顺序，甚至不知道哪些过程会被调用，因此，许多隐式调用的系统也包含显式调用作为构件交互的补充形式。

**分层的体系结构风格：**

层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见。这样的系统中构件在一些层实现了虚拟机(在另一些层次系统中层是部分不透明的)。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。

这种风格支持基于可增加抽象层的设计。这样，允许将一个复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件重用提供了强大的支持。

**仓库风格**

在仓库风格中，有两种不同的构件：中央数据结构说明当前状态，独立构件在中央数据存贮上执行，仓库与外构件间的相互作用在系统中会有大的变化。

控制原则的选取产生两个主要的子类。若输入流中某类时间触发进程执行的选择，则仓库是一传统型数据库；另一方面，若中央数据结构的当前状态触发进程执行的选择，则仓库是一黑板系统。

**C2风格**

C2体系结构风格可以概括为：通过连接件绑定在一起的按照一组规则运作的并行构件网络。C2风格中的系统组织规则如下：

(1)系统中的构件和连接件都有一个顶部和一个底部；

(2)构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部，而构件与构件之间的直接连接是不允许的；

(3)一个连接件可以和任意数目的其它构件和连接件连接；

(4)当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

1. **保证项目质量属性的设计：**

- 首先指定质量保证计划，分析哪些部分容易出现质量问题。

- 代码问题：开发的时候代码编写者可能存在一些考虑不周的情况，从而编写的代码出现质量问题

- 设计问题：设计项目框架的时候可能存在考虑不周的情况。

- 开发前选定制定开发标准，开发规范。

- 指定相应的代码风格和命名风格，便于代码维护，从而保证了代码质量。

- 开发过程统一一些内容术语，减少设计文档和开发文档的二义性，便于明确设计。

- 选择分析设计方法和工具

- 使用单元测试以及性能分析工具检查一些常见的程序问题和漏洞。

- 严格执行阶段评审，代码检查等

- 对代码和设计提交进行review，减少由于一些疏忽大意带来的问题

- 每一个开发阶段都需要进行一定的检查和分析，确保当前质量。

- 进行阶段测试，每个开发阶段都设计对应的测试样例。对每个模块和功能进行测试。

- 使用单元测试工具测试每个模块的功能和性能。

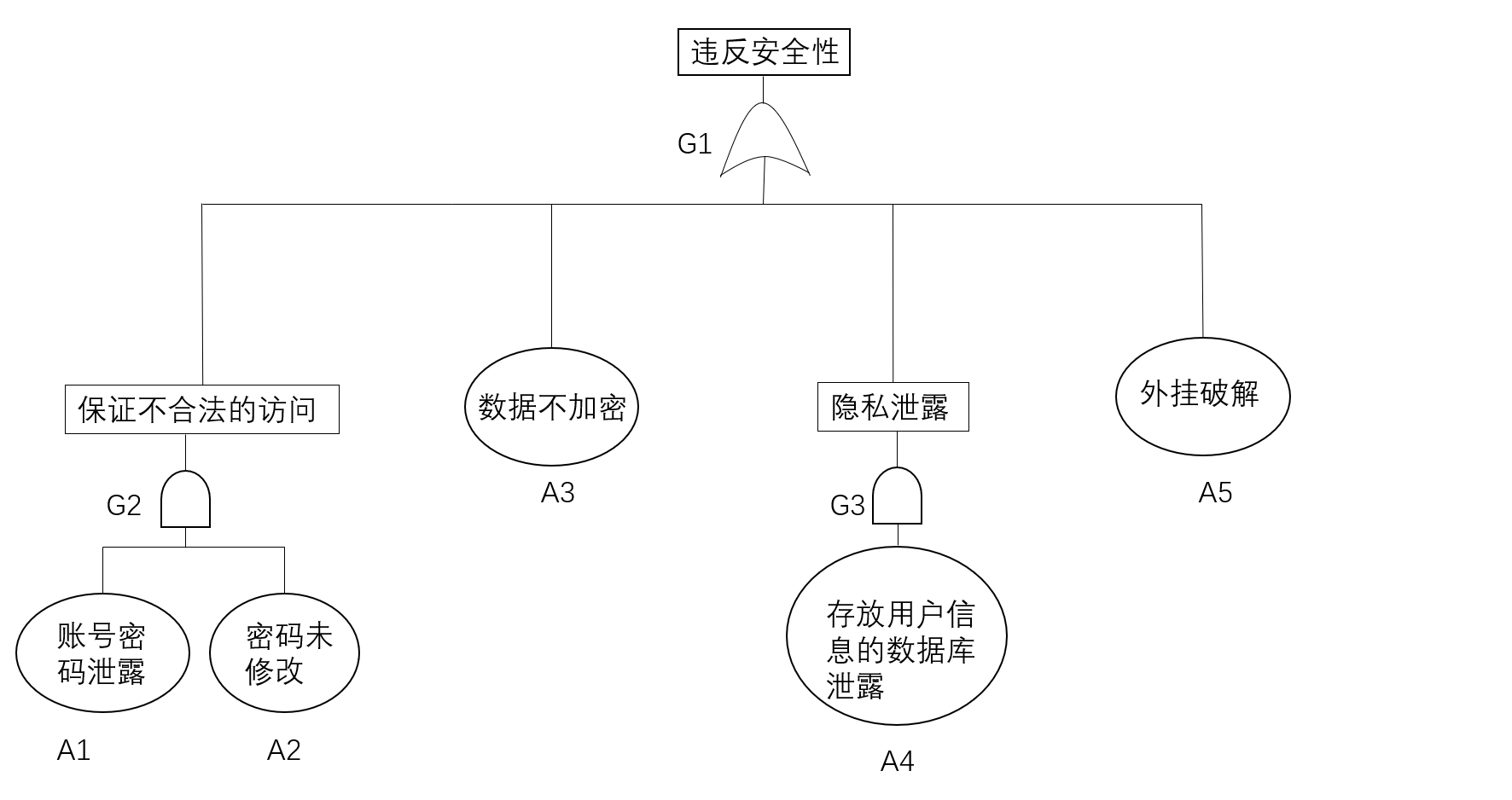
- 使用性能分析工具检查内存泄漏等问题。

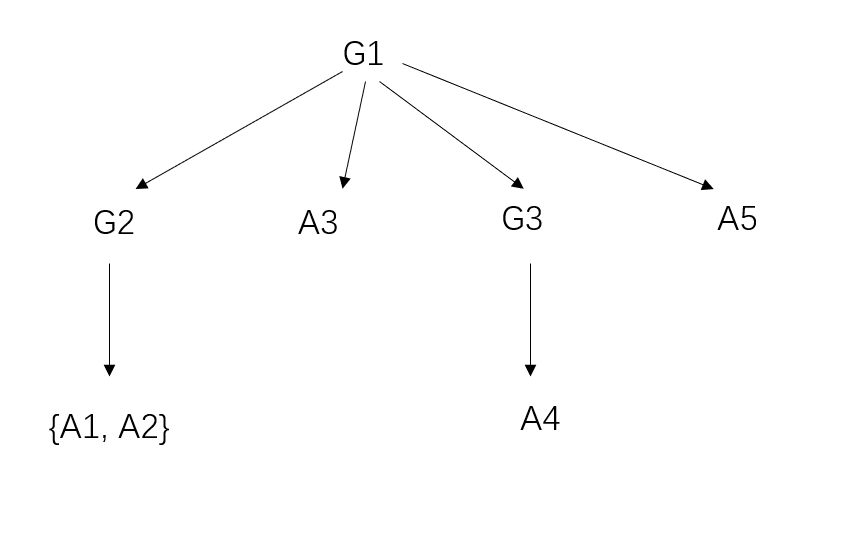
1. **针对自己的项目，描绘故障树，分解为割集树**

可能出现的故障为违反安全性、运算错误、性能过差

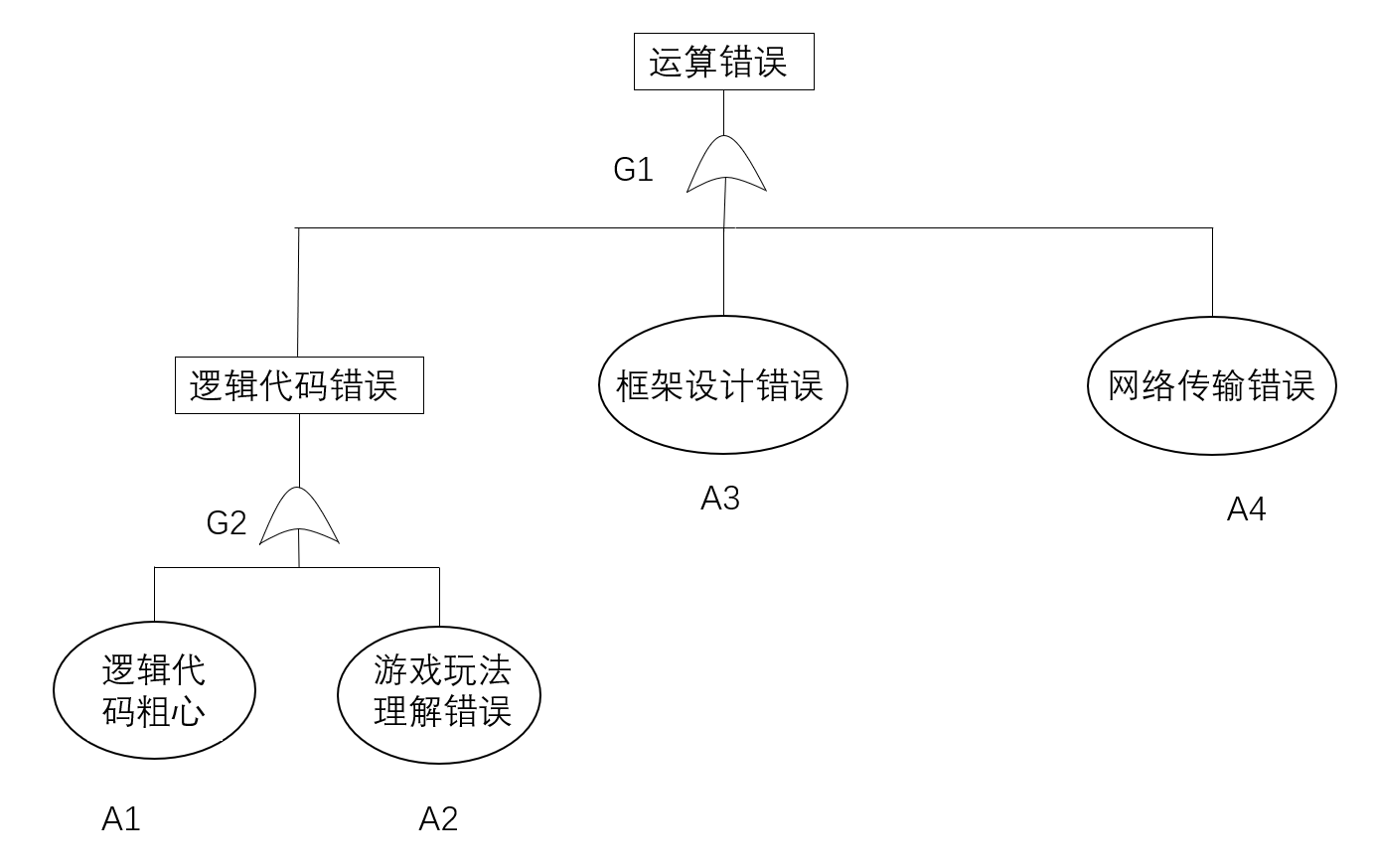
各自的故障树和割集树如下图所示：

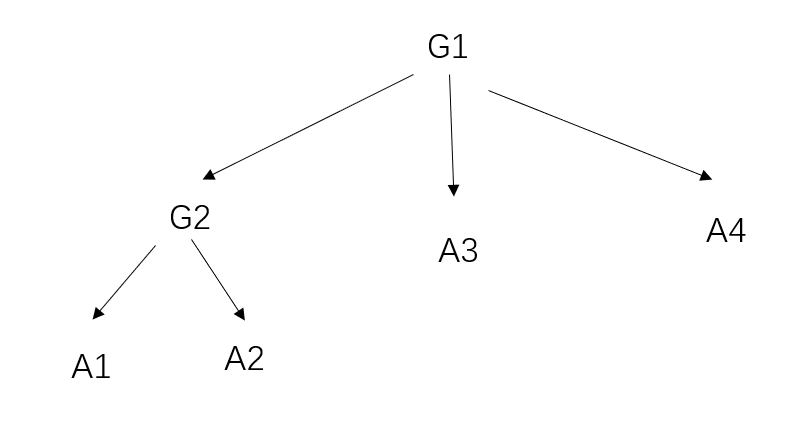
**违反安全性：**





**运算错误：**





**性能过差：**

