# 模型

* 构建一个大家合作编辑的模型是开端；
* 合作的形式以场景为分工、以场景中共同使用的实体、岗位、组件等为合作；
* 一张图有一张图的价值，一张图不能呈现所有关注；
* 一件复杂的事儿是需要多个步骤，产生多张图才可描述清楚的；

# 业务建模

## 业务建模的职责和手段

**业务建模的职责是确定待建软件系统的职责（做什么）**，这种职责是通过业务流程分析得到的，可以通过在描述业务用例实现时引入待建软件系统来建模其职责，以及交互的岗位和其他系统（业务工作者）。

## 业务流程的识别

为了做业务流程分析，首先需要识别业务流程。业务流程的识别线索来自于业务流程的服务对象（业务参与者）。

## 业务实体的识别

业务建模过程中需总结业务内相关的业务概念，以及它们之间的关系，这些可以输入到软件分析。一般来说，一个业务流程会对应一个核心的业务实体，描述出该业务实体的状态图是重要的。另外，通过实体关系及其核心业务实体的属性变化应该可以解释业务流程执行的效果。在识别业务实体时，在单一业务步骤中或两个业务步骤间交互的信息应该在后续的分析模型中进行建模。

# 系统用例

系统用例是衔接业务建模和后续分析设计的桥梁，系统用例由于是业务建模的产出物，于是将其分配给业务建模人员来创建更合理。

## 用例描述

每一个系统用例应描述出其核心操作步骤及其软件要实现的重要规则。

## 词汇表

当没有业务建模或在业务建模过程中没有识别业务实体的情况下，可以通过创建【词汇表】的形式来描述其软件需求中涉及的核心业务概念，这些信息会在接下来的分析中使用到。

# 分析模型

分析模型是以重要的系统用例作为分析对象，通过描述时序图的手段来做用例分析构建起来的。分析模型需对系统中1/4的有代表性的系统用例进行分析才可以得出一个能够满足多个用例场景中的分析模型。

## 分析包的划分

分析包的划分除了要考虑分析类的耦合内聚关系外，分离易变和稳定的分析类也是重要的划分思路。分析包的划分是组件模型的概念性输入，也是其最核心的输入，它从业务层面描述出了系统的概念性组成。

## 通过创建抽象的分析类来容纳变化

需求本身提出的可变性要求，以及需求中不明确的地方都是需要抽象思考，通过创建抽象的分析类来容纳变化。这时候这些抽象类的稳定性变得更加重要，需要着重思考和交流。

# 设计模型

设计模型要影响代码，通过工具导出成最早的代码框架来影响之。设计模型的稳定性比分析模型弱，其组件关系、接口，以及组件内部重要的设计类值得更新，其他细节的设计类不必保持与代码的一致性。

## 设计机制

在构建设计模型时须确定核心的架构风格（如EDA）、组件内结构范型、错误处理的模式等等，这些工作需要在设计机制章节描述清楚。

## 设计模型构建应遵守的原则、模式和经验

* 数据库表应私有化，并视其为不稳定的设计元素；
* 组件间通过接口访问，强调其接口的稳定性；
* 通过参数的接口化可增强接口的稳定性；
* 对于操作频繁的Entity，应通过冗余来减少创建实体时表连接的开销；
* Entity间的关系采用manytoone和onetomany来构建更合适；
* 对于打算在其他环境下还要使用的组件，其接口、移植性、扩展性、错误处理等方面要强化其质量；

# 部署模型

云计算的风靡使得单一计算模型（单应用、单数据库）已经过时，而分布式计算、微服务等计算方式越来越流行。构建一个由单应用向分布式计算过度的方案成为一件大事，挑战不小。最大的挑战不是数据一致性的保证，而是演变过程中设计和编程习惯的改变。

# 实现模型

实现模型就是代码框架，包括了开发期的工程结构、依赖的第三方框架版本、运行的具体方式等等。对于java路线，推荐使用SpringBoot+Maven来构建。对于在设计上强调的组件间的单项关系可以通过在maven环境下创建多个module来约束。