结构化分析与设计：案例解析

何啸

Version 0.2

1 系统概述

假设我们要开发一个教室在线预定系统，它能够满足管理员管理教室信息、审核教室预定请求，普通用户申请预定教室、查看教室预定情况的功能。具体来说，普通用户能够通过系统进行教室预定、查看教室预定情况、取消自己所预定的教室。我们假设用户不能修改自己的预定，因为修改预定操作用可以看作是取消预定和重新预定这两个操作的结合。管理员能够审核教室预定请求、强制取消教室已经批准的教室预定请求、修改教室信息和状态。

为了方便表述，我们将上述核心需求进行编号，如下：

Table 1: 需求列表

编号	用户类型	说明
U1	普通用户	教室预定
U2	普通用户	查看教室预定情况
U2	普通用户	取消自己所预定的教室
G1	管理员	审核教室预定请求
G2	管理员	强制取消教室预定
G2	管理员	修改教室信息和状态

2 结构化系统分析

2.1 分析的目标和方法

需求分析的目的是将用户的需求转化成系统需求。为什么要进行这样的转换/分析呢？这主要是因为用户需求一般是采用自然语言描述的，便于人的理解，但通常比较抽象，其中可能存在一些矛盾和不清晰的地方。而系统的需求则是明确的、具体的、无歧义的，更加结构化，主要方便开发人员理解。用户的需求往往带有很强的领域特征，如果要完全理解用户的需求可能需要一定相关知识，而这不是每个开发人员都能做到的（也不应该要求每个开发人员都做到）。因此，为了让开发能够顺利进行下去，需要

利用计算机语言将用户需求进行重新描述，并将其中隐含的、抽象的需求明确，将矛盾的需求识别出来。

需求分析涉及到两个基本问题：（1）分析的方法和过程；（2）用于描述需求的计算机语言。

对于结构化分析方法而言，它的基本视角是将待开发的系统看作是一个**数据加工器**——一个能够接受输入，并根据输入产生相应输出的装置。这就好像是一个工厂，接受一些原材料（输入），然后将它们加工成产品（输出）。当然，加工的过程可能包含好几步，前一个加工的结果传递给下一个加工。结构化分析的核心就是从用户的需求出发，建立一个能够体现用户需求的数据加工系统，其中可能由若干个子加工相互组合而成。在结构化方法中，数据流图被用来描述这个数据加工系统。

结构化方法以数据为核心，对系统的需求进行梳理。那么如何定义数据呢？这就需要进一步进行数据建模。简单来说，数据建模是提炼用户需求中的所有数据管理需求，并建立相应数据结构的过程。在经典的结构化方法中，E-R模型通常被用来进行数据建模。

下面将结合实例，详细展示结构化分析的过程。

2.2 数据流图的建立

数据流图是一种软件建模工具，它提供一组概念帮助开发人员对系统进行抽象和描述。数据流图主要提供以下几个建模概念：

- 数据源/数据潭：表示系统的输入和输出，它们是对系统用户和外系统的抽象，表示系统的边界。数据源/数据潭并不属于系统内部构造，它们只表示与系统交互的接口。

- 数据加工：表示系统中的一个数据处理单元，整个系统也可以看作是一个复杂的数据加工。数据加工的作用是将输入数据变换成输出数据。在变换过程中，数据所包含的信息不应增多——因为系统不可能猜测未知的信息。

- 数据存储：表示系统的数据存储单元，是对数据库、文件等存储功能的抽象。严格地讲，数据存储不具备数据处理的功能，它只能读取和写入一定数据。如果要进行数据查询和检索，则应当将查询和检索的功能移到数据加工中。不过出于简化的目的，有时候也会让数据存储支持数据检索能力。

- 数据流：表示系统中的数据传递和转移，用来连接数据源/数据潭、数据加工和数据存储。这里需要注意的是，数据源/数据潭之间、数据存储之间、数据源/数据潭和数据存储之间不能直接使用数据流相连，它们中间必须适当插入数据加工。

数据流图的建立过程是一个**自顶向下，逐步求精**的过程。一开始，我们将整个系统看作是一个数据加工，然后将用户抽象成数据源/潭，接着识别用户和系统之间存在的各种数据流，从而建立一个**顶层数据流图**。之后，我们进入到一个迭代的过程——通过分解上一个步骤中定义的数据流图中的数据加工，逐步对系统模型进行细化，从而建立完整的系统模型。

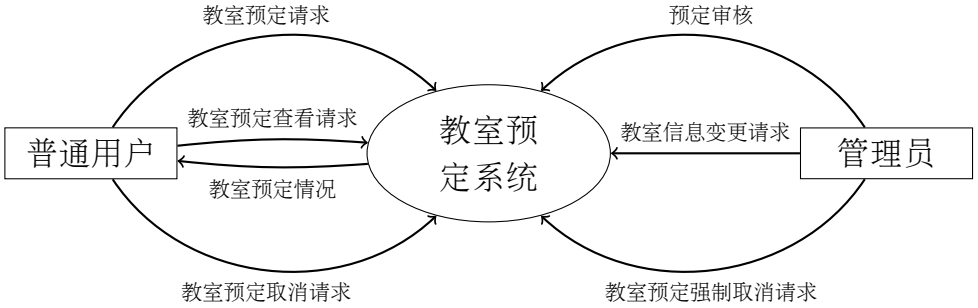

Figure 1: 简化的顶层数据流图

图1展示了该系统的顶层DFD。我们将整个系统抽象成一个数据加工“教室预定系统”。然后将“普通用户”和“管理员”抽象成数据源/潭，因为他们会与向系统发送请求并接受处理结果。这里需要注意两点：（1）除了用户（人）可以被抽象成数据源/潭以外，外系统、外部设备也可以被抽象成数据源/潭；（2）数据源/潭属于系统边界——也就是介于系统内外进行交互的接口处，因此不能将系统内部的构造成分识别成数据源/潭。

再来关注图中的数据流，不难发现我们将需求中明确提出的交互抽象成数据源/潭和数据加工之间的数据流（也就是数据交互），比如，普通用户能够预定教室，这就意味着普通用户必须能够向系统发送一个“教室预定请求”，因此在“普通用户”和“教室预定系统”之间存在一个传递“教室预定请求”的数据流。按照这样的思想，我们可以将需求中描述的所有交互表示成数据流。在这一过程中，需要特别注意以下几个问题：（1）并非所有的需求都能抽象成数据流；（2）需求中直接体现出来的数据流可能是不完整的，比如用户向系统发送一个请求，那么系统是否需要返回请求处理结果？这一点可能没有在需求陈述中说明，需要需求分析人员进行进一步分

析和辨别，从而识别和定义出准确的系统需求；（3）用户通常喜欢从业务流程的角度描述需求，而业务流程中体现出来的交互是分步骤的，在建立数据流图时（特别是顶层数据流图），可以将这些步骤合并起来，将其中分阶段传递的数据合成一次数据交互，从而简化模型。

自己动手 请根据你的理解，完善图1中的数据流。

接下来，我们需要采用“自顶向下，逐步求精”的分析策略，对图1进行细化，将其中的数据加工进行分解。如何进行分解呢？这里没有统一的方法，需要根据分析人员对于该数据加工的理解采取不同的分解方法。一般而言，可以从下面两个角度考虑：（1）如果一个数据加工体现了不同的、相互并列的数据处理过程，那么可以考虑将其分解，使得每一个子加工只实现一个数据处理功能；（2）如果一个数据加工体现了一个相对复杂的数据处理流程，那么可以考虑将其分解，使得每个子加工只实现一个步骤。在进行分解的时候，必须注意分解前后的**一致性**。简单来说，一致性约束包含以下几点：（1）如果讲一个数据加工分解为多个数据加工，那么分解前后的数据加工必须具有完全相同的功能——常见的错误是分解后的数据加工所能完成的功能比分解前的数据加工多！（2）分解前的DFD所出现的数据流必须出现在分解后的DFD中，并且在分解后DFD中所有射入和射出系统的数据流也必须出现在分解前的DFD中，除非数据流也被分解了——常见的错误是分解后的数据流比分解前的数据流传递的信息多，或者分解前的数据流在分解后丢失了！

通过分析图1，不难发现顶层数据加工综合了“普通用户”和“管理员”的数据处理功能，因此我们可以将这个数据加工一分为二——这就得到了第一层的数据流图，如图2所示。你一定会问，这个图为什么是不连通的？当你发现你的数据流图是不连通的，那可能说明在你的分析过程中忽略了一些必要元素[1]。事实上图2缺少一些数据存储，它们能够将这两个独立的数据加工连接起来。

缺少哪些数据存储呢？这就需要进一步思考“普通用户子系统”和“管理员子系统”之间可能共享（注意不是交换）哪些信息。我们知道用户预订教室后，需要管理员对这个预定请求进行审核。因此，“普通用户子系

[1]还有一种可能是你的系统确实是由多个独立的子系统构成，尽管这在某种意义下不能称之为“一个”系统。在课程范围内，这种情况是不会出现的。

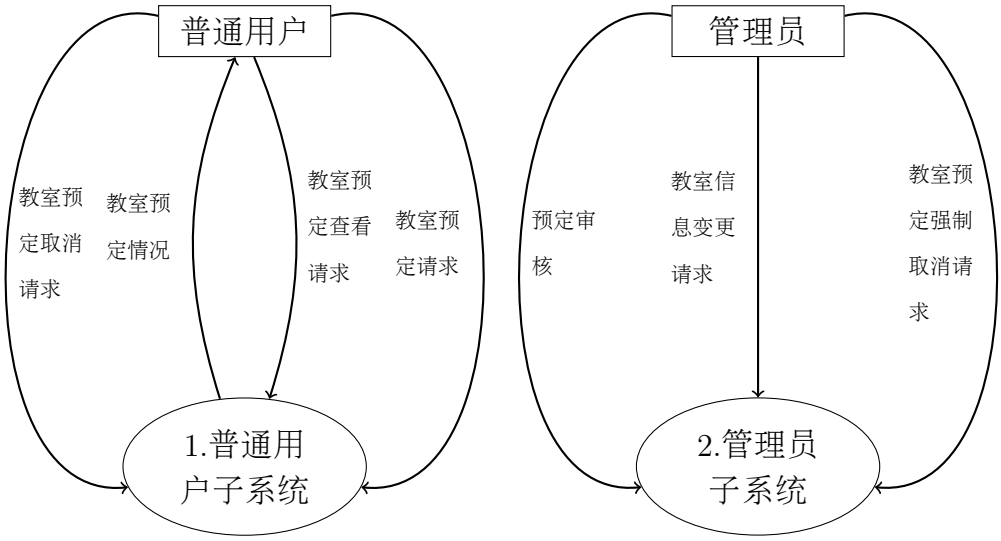

Figure 2: 第一层分解的数据流图（缺少数据存储）

统”和“管理员子系统”必须共享一些“预定单”（也可以叫做其他名字），同时也说明“预定单”是系统内部需要长期保存的数据项。我们将“预定单”识别成一个数据存储，“普通用户子系统”显然会读写这个数据存储，而“管理员子系统”也需要能够修改这个存储。“普通用户”发送的预定请求经过系统的处理后会被保存在“预定单”中，然后“管理员”能够审核、取消用户的预定——也就是修改“预订单”中的数据。

除了“预定单”，系统还需要存储另一个信息，也就是“教室信息”。这个信息会被“普通用户子系统”读取，而可以被“管理员子系统”修改。“普通用户子系统”为什么需要读取“教室信息”呢？这里可以从两个角度理解：（1）在创建”预定单“时，需要读取“教室信息”以便检查用户的预定请求是否合理；（2）用户在使用该系统创建预定请求时，系统可能需要列出哪些教室可用，并且用户进行选择。特别需要注意的是，用户的需求并没有明确说明普通用户是如何利用教室信息的，这是需求分析人员对于用户需求的理解和分析。因此可能存在多种理解，最终需要得到用户的确认。

将这两个数据存储添加到模型中，我们就得到了一个（相对）完整的第一层数据流图，如图3所示。不过请注意，图3所示的数据流模型依然可能是不完整的，比如“管理员子系统”实际上可能需要读取“教室信息”。但从一定的抽象层次看，图3基本上是自洽的，并且能够覆盖用户的必要需求。这里有一个概念非常重要：软件建模（即分析与设计）是对软件系统的一

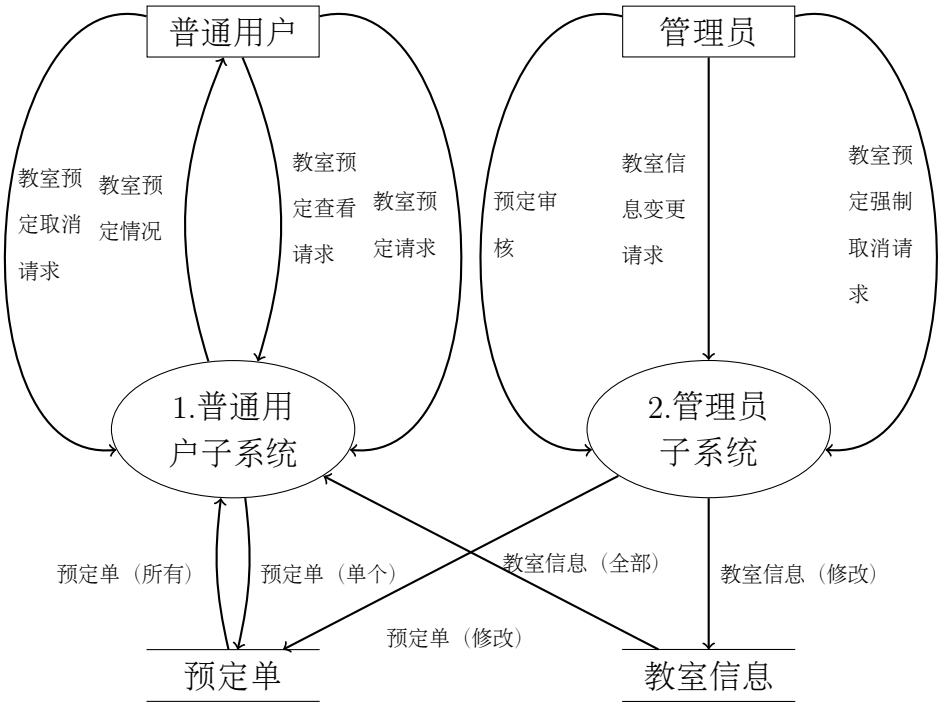

Figure 3: 第一层分解的数据流图（完整）

种**抽象**过程。在每一个抽象层次上所展现的细节，与这个抽象层次的关注点有关。比如，在分析阶段，我们更多地侧重于将用户的核心需求表达出来，并且发现其中的隐含、不明确的需求，解决需求的冲突，因此我们可以忽略掉一些与这个目标无关的细节。就本例而言，“管理员子系统”是否能够读取“教室信息”并不属于用户的核心需求，因此我们在分析阶段可以将其忽略掉（即抽象），这一部分内容可以推迟到软件设计甚至是编码阶段再进行决定。当然，在分析阶段考虑这个问题也不能算是错误。但如果过多的考虑这些可以被抽象/忽略的细节问题，可能会导致分析过程变得十分复杂，不利于控制软件开发的复杂性。

按照上述过程，我们可以继续对图3进行分解。作为例子，我们只考虑对“普通用户子系统”的分解，而将“管理员子系统”作为练习留给读者。

“普通用户子系统”主要响应用户的三种交互请求，即教室预定请求、教室预定查看请求、教室预定取消请求。这三种请求可以看做三个相互独立的功能，因此我们可以将“普通用户子系统”分解成三个基本部分。然后我们考虑每一个基本功能:

1.1 教室预定。用户的需求中没有明确说明这个功能是如何实现的（即没有给出具体的业务流程），所以我们可以假设一个合理的业务流程，如下：（a）用户填写教室预定申请，其中包括所要预定的教室、预定的时间段、预定人等信息。（b）这个信息被传入系统后，需要验证这个申请是否合理，比如教室是否存在、所申请的时间段是否空闲、预定人是否合法等。（c）如果申请合法，系统会产生一个预定单并存入相应的数据存储中。

1.2 预定查询。预定查询的功能比较简单，可以看作是（a）给定一个用户，查询该用户所预定的所有教室信息，以及（b）查看所有教室的预定情况（需要同时读取教室信息和预定单。

1.3 取消预定。取消预订就是取消一个预定申请，用户需要输入预定申请的ID，然后系统将这个ID所对应的预定单进行取消。

通过上述分析，我们可以建立一个第二层的数据流图，如图4所示。

这里需要重点体会数据加工“1.1a 教室预定”、“1.1b 请求验证”和“1.1c 创建预定单”的关系。加工1.1a接受用户的输入，并且从中提取出所借教室、时间段和用户ID，并将这些数据传递给加工1.1b进行校验，并将整个用户输入转发给加工1.1c。加工1.1b接收所需要验证的信息，并通过读取数据存储中的内容，校验预定请求是否合法（比如，通过读取所有预定单，检查所借教室在所预定的时间段是否可空闲），并将检验结果发送给加工1.1c。最后，加工1.1c根据校验结果以及用户的预定请求，产生一个预定单，并将其存入相应的数据存储中。请注意，我们建立的是数据流图，它反映了系统中数据的流动和变换，所以要保证后一个数据加工所需要的数据能够从前一个数据加工所产生的结果和数据存储中获得。

最常见的错误是将数据流图看作是控制流程图，将数据加工理解为控制流中的步骤和操作。这样做的后果是某个数据加工所需要的信息可能不能从前面的数据加工中获得。比如，一个业务流程可能是“用户首先登陆（输入用户名和密码），然后执行文档编辑操作”，一个错误的抽象是将“登陆”和“文档编辑”识别成两个数据加工，并将它们利用一个数据流连接起来（看起来符合业务流程的描述）。如图5所示，用户向”登陆“发送用户名密码，然后经过“登陆”加工后，数据直接传递给了“文档编辑”，那么请问从用户输入的“用户名”和“密码”这两个数据项，如何分析、转换得到“文档编辑”所需要的输入呢？这显然是不可能的。

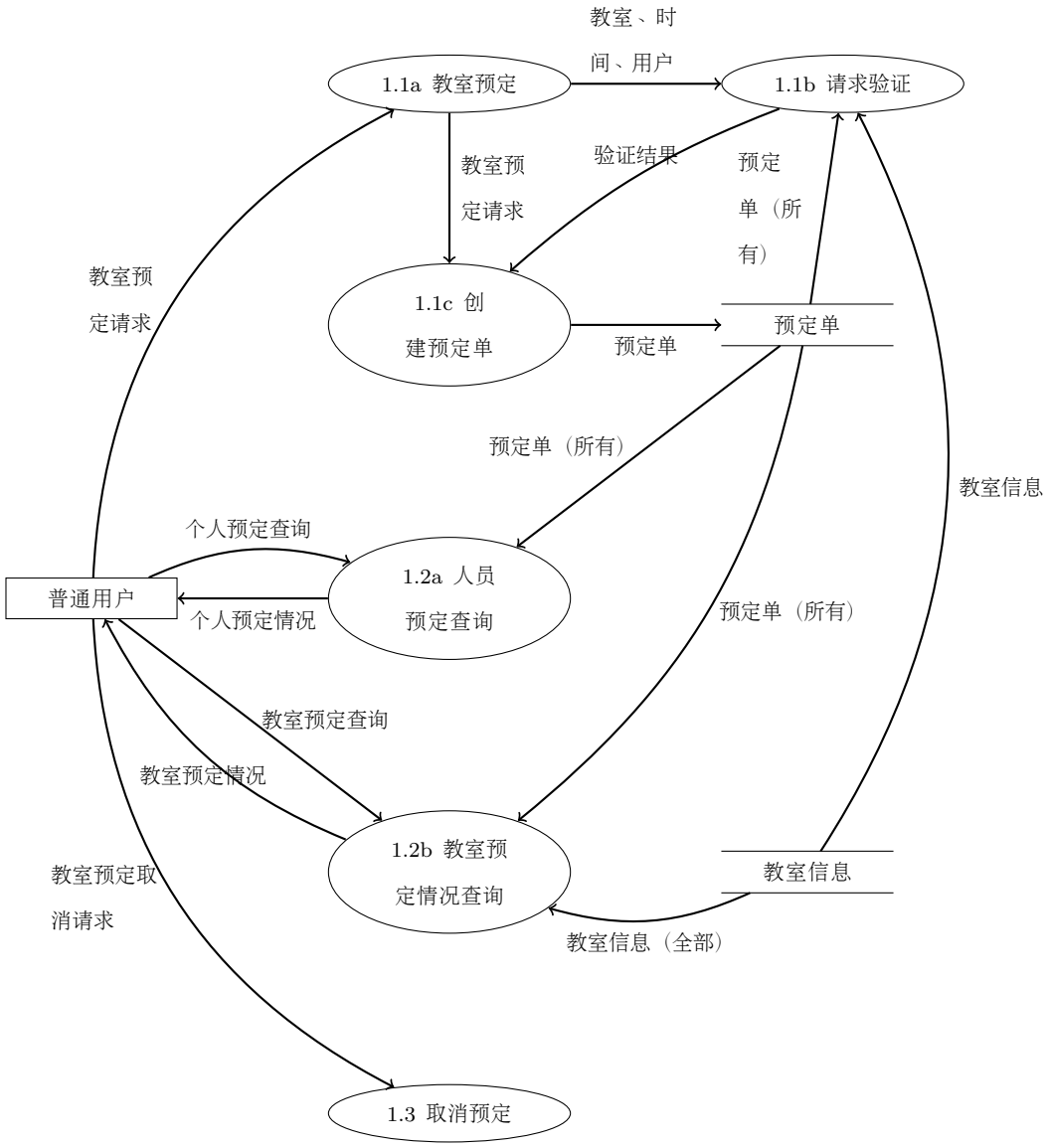

Figure 4: 第二层分解的数据流图（普通用户子系统部分）

Figure 5: 错误的数据流图

2.3 数据字典的建立

数据字典是对数据流图的辅助说明，它对数据项、数据结构、数据流、

数据存储、数据加工等进行定义和描述。其中，数据流、数据存储、数据加工是说明的重点。

- **数据项**是对数据流图中数据块的数据结构中的数据项说明，是不可再分的数据单位。数据项的一般定义格式如下：数据项描述={数据项名，数据项含义说明，别名，数据类型，长度，取值范围，取值含义，与其他数据项的逻辑关系}。

- **数据结构**是对数据流图中数据块的数据结构说明，一个数据结构可以由若干个数据项组成，也可以由若干个数据结构组成，或由若干个数据项和数据结构混合组成。数据结构的一般定义格式如下：数据结构描述={数据结构名，含义说明，组成:{数据项或数据结构}}。

- **数据流**是对数据流图中数据流的说明。数据流的定义格式如下：数据流描述={数据流名，说明，数据流来源，数据流去向，组成:{数据结构}，平均流量，高峰期流量}。

- **数据存储**是对数据流图中数据存储特性说明，它的定义格式如下：数据存储描述={数据存储名，说明，编号，流入的数据流，流出的数据流，组成:{数据结构}，数据量，存取方式}。

- **数据加工**是对数据流图中数据加工的说明，它的定义格式如下：处理过程描述={数据加工名，说明，输入:{数据流}，输出:{数据流}，处理:{简要说明}}。

我们仅以图3为例建立数据字典，并将其它数据流图的数据字典留作练习。

数据项，实际操作时可根据情况省略这一部分。

1. 用户ID, 用户的标识符, UID, String, 32字节
2. 教室ID, 教室的标识符, RID, String, 32字节
3. 申请ID, 申请单的标识符, QID, String, 32字节
4. 日期, 系统中的日期, DATE, 日期类型, 32字节
5. 时间段, 预定的时间段, TIME, Integer, 1字节, {1,2,3,4,5}, 1表示8:00-9:30, 2表示10:00-11:30, 3表示13:30-15:00, 4表示15:30-17:00, 5表示19:00-20:30。

6. 申请状态, 申请单的状态, STATE, Integer, 1字节, {0,1,2,3}, 0表示待审核, 1表示批准, 2表示未批准, 3表示预定取消。

7. 验证结果, 预定申请的验证结果, RESULT, Boolean, 1字节, {true, false}。

数据结构，实际操作时可根据情况省略这一部分。

1. 预定请求, 普通用户的教室预定请求, {UID, RID, DATE, TIME}。

2. 预定单, 系统存储的教室预定记录, {QID, UID, RID, DATE, TIME, STATE}。

3. 教室预定查看请求, 用户发送的教室预定查看请求，包括个人预定查看请求和教室预定状态查看请求, {个人预定查看请求|教室预定状态查看请求}。

4. 个人预定查看请求, 用户发送的个人预定查看请求，{UID}。

5. 教室预定状态查看请求, 用户发送的教室预定状态查看请求，{DATA}。

6. 教室预定结果, 系统返回给用户的一组教室预定状态, {预定单*}。

7. 教室预定取消请求, 用户发送的的教室预定取消请求, {QID, UID}。

8. 教室信息, 系统存储的教室信息, {RID}, 这里简化了教室信息的定义。

9. 教室信息更新, 用户发送的教室信息更新请求, 包括要更新的教室ID和新的教室信息（简化成一个新的RID）, {RID, RID}。

10. 审核, 用户发送的教室预定审核, 包括要申请ID和审核结果息, {QID, STATE}

数据流，这一部分必须包括数据流图中的所有数据流。

1. 教室预定请求, 普通用户的教室预定请求, 普通用户, 普通用户子系统, 预定请求, 平均每天10次, 最高每天20次。

2. 教室预定查看请求, 普通用户的教室预定请求, 普通用户, 普通用户子系统, 教室预定查看请求, 平均每天20次, 最高每天50次。

3. 教室预定情况, 系统返回的教室预定情况, 普通用户子系统, 普通用户, 教室预定结果, 平均每天40次, 最高每天100次。

4. 教室预定取消请求, 用户发送的的教室预定取消请求, 普通用户, 普通用户子系统, 教室预定取消请求, 平均每天2次, 最高每天10次。

5. 教室预定强制取消请求, 管理员发送的的教室预定强制取消请求, 管理员, 管理员子系统, 教室预定取消请求, 平均每天1次, 最高每天20次。

6. 教室预定审核, 管理员发送的的审核过的教室预定单, 管理员, 管理员子系统, 审核, 平均每天10次, 最高每天20次。

7. 教室信息变更请求, 管理员发送的的教室信息更新请求, 管理员, 管理员子系统, 教室信息更新, 平均每天0次, 最高每天20次。

8. 预定单（单个）, 普通用户子系统, 预定单（数据存储）, 预定单, 平均每天12次, 最高每40次。

9. 预定单（所有）, 预定单（数据存储）, 普通用户子系统, 预定单*, 平均每天40次, 最高每100次。

10. 预定单（修改）, 管理员子系统, 预定单（数据存储）, 预定单, 平均每天1次, 最高每20次。

11. 教室信息（修改）, 管理员子系统, 教室信息（数据存储）, 教室信息更新, 平均每天0次, 最高每天20次。

12. 教室信息（全部）, 教室信息（数据存储）, 普通用户子系统, 教室信息, 平均每天30次, 最高每天70次。

数据存储，这一部分必须包括数据流图中的所有数据存储。

1. 预定单, 系统保存的所有预定单, （编号省略）, 流入的数据流包括“预定单（单个）”、“预定单（修改）”, 流出的数据流包括“预定单（所有）”, 预定单*, 数据量每年增长不超过2000条, 检索/更新。

2. 教室信息, 系统保存的所有教室信息, （编号省略）, 流入的数据流包括“教室信息（修改）”, 流出的数据流包括“教室信息（全部）”, 教室信息*, 数据量每年增长和变化不超过100条, 检索/更新。

数据加工，这一部分必须包括数据流图中的所有数据加工。图3中的数据加工粒度较粗，描述起来比较复杂，而在实际建模过程中，应当选择更细力度的数据加工进行定义（比如如4中的数据加工）。

1. 普通用户子系统, 响应普通用户请求的加工，输入包括：教室预定请求、教室预定查看请求、教室预定取消请求、预定单（所有）、教室信息（全部），输出包括：预定单（单个）、教室预定情况, 根据不同的请求进行相应的处理（我们在之前的介绍中已经对其进行详细描述了，这里不再赘述）。

2. 管理员子系统, 响应管理员请求的加工，输入包括：教室预定强制取消请求、教室信息变更请求、教室预定审核，输出包括：教室信息（修改）、预定单（修改）, 根据不同的请求进行相应的处理（我们在之前的介绍中已经对其进行详细描述了，这里不再赘述）。

关于数据字典的讨论 数据字典是一种比较简单的、传统的辅助说明工具，它实际上规定的对数据流图进行说明时必须解释的内容。如果教条地规定必须使用数据字典对数据流图进行说明，那么会导致数据流建模的工作量大幅增加，因此在实际操作时可以将数据字典看作是一种“建议”，按照其中的规定，根据实际的需要灵活地进行裁剪。但必须指出的是，无论如何裁剪，也不应该去掉必要的说明，没有文字说明的模型是不可接受的。软件模型只是对软件系统高度抽象化的表示，它的作用是辅助开发人员的理解。但在抽象过程中，模型必然忽略、简化、抽象系统中的一些信息，这必须靠文字说明进行补充。

2.4 E-R模型的建立

数据流图的核心是数据和数据加工。对于数据的定义和建模，我们推荐采用更加结构化的工具——E-R模型。E-R模型是一种常用的数据库设计模型，我们同样可以把它用在需求分析阶段进行数据建模。

E-R模型中主要包括实体（Entity）、关系（Relationship）和属性（Attribute）这三类概念。实体是一种数据对象，反映了系统中需要存储的数据。关系是实体之间的联系，可以看作是一种N元祖。最常见的关系是二元关系，二元关系又可以分为一对一关系、一对多关系和多对多关系。属性是对实体（和关系）的细节描述。

用E-R模型进行数据建模时，最重要的是理解究竟应当对系统中的哪些成分进行建模。简单来说，我们需要对系统需求中提及的重要数据对象及它们之间的关系进行建模。那么什么是**重要数据对象**呢？一个简单的判定原则是：如果系统需要存储这个对象，那么它就是重要的[2]。

数据建模的基本过程是先识别实体，在识别实体之间的关系，最后细化实体和关系的属性。当然，这同样是一个迭代的过程，必要的时候可以重复进行上述过程，直到模型不再变化。

首先，我们来识别系统中的实体。根据之前的分析，不难看出系统中存储的数据主要是教室预定单和教室信息，因此可以识别出“预定单”和“教室”两个实体。

接着，我们考虑这两个实体之间的关系。我们知道预定单需要关联一个教室，因此预定单和教室之间存在一个关系，可以命名为“预定”。

最后，我们我们考虑这两个实体有哪些属性。对于这一点我们已经在数据字典中给出了相应的定义。

综上所述，我们可以建立如图6所示的E-R模型。

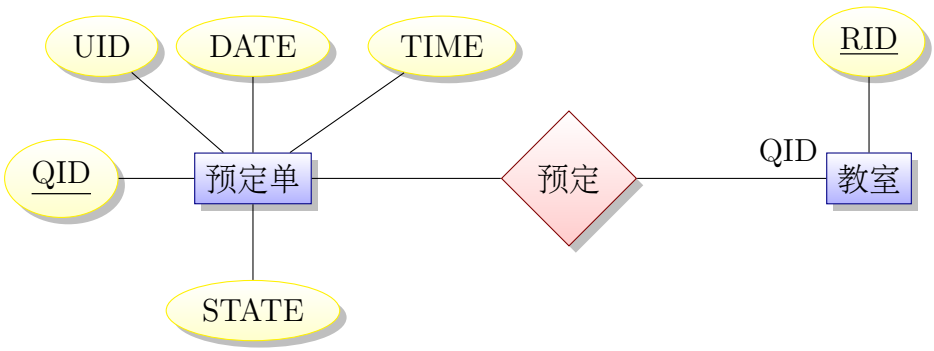

Figure 6: E-R模型

在建立E-R模型时，一个常犯的错误就是对于关系的理解和运用。什么是关系？关系反映了实体之间的联系，**它必须为系统提供了一组有意义的信息**。实体之间的联系可能很多，但并不是所有的联系都需要定义成关系。举个例子，在一个教务系统中，教务员能够“修改”和“统计”学生成绩，那么是否意味着教务和学生之间存在“修改”和“统计”这两个关系呢？答案是否定的。这是因为系统不需要记录是哪个教务员修改了哪个学生的信息，也不需要记录教务员执行了一次统计操作，因此“修改”和“统计”虽

[2]这只是一般而言，特殊情况下一些不需要存储的数据也可能需要进行建模。

然是教务员和学生之间的某种联系，但却不能识别成实体之间的关系。发生这种错误的原因是把用户能够执行的某种操作看作是这个用户和操作对象之间的关系——这并不符合数据建模的主要关注点。从面向对象的视角看，这种联系是一种“依赖”，但不是关系（关联）。

进一步思考 系统中是否还存在其他的实体和关系呢？从需求陈述和之前的分析中，我们还可以发现其它的数据对象，比如用户、预定申请等等。那么为什么不将它们识别为实体并在E-R模型中定义呢？这里一个最主要的原因就是根据目前的需求分析，这些信息并不需要存储在系统中。当然，如果需求发生变化，比如系统用户需要注册，并且系统需要维护用户的注册信息，那么就必须将用户识别为实体。

3 结构化系统设计

3.1 设计的目标和方法

设计的目标是将系统需求（分析的结果）转化成系统的结构定义，是将“做什么”变成“怎么做”的过程，也是体现“计算思维”的过程。计算机技术的神奇之处就是将一个问题变成一系列可计算的步骤（算法和操作）。从系统层次看，这样的变换依然存在，只是我们会在一个更高的层面上讨论可计算的步骤（模块和构件）。

系统设计又可以分成两个阶段：（1）软件体系结构设计（也称概要设计），和（2）详细设计。软件体系结构设计的目标主要是建立软件体系结构（Software Architecture[3]）。不失一般性，软件体系结构是由一组构件（Component）和连接子（Connector）组成的软件模型。而详细设计的主要目标则是定义软件构件的行为和约束。

3.2 软件体系结构设计

我们将使用模块结构图（Module Structure Diagram）来描述软件体系结构。模块结构图主要包含模块和模块间调用这两个概念，分别对应软件体系结构中的构件和连接子。

[3]关于SA的更多讨论，可以参阅https://en.wikipedia.org/wiki/Software_architecture，以及国际会议ICSA。

如果已经定义了一个DFD图，那么可以利用课上及教材中介绍的从DFD到MSD的映射方法，将DFD图机械地转换成MSD。本文不再赘述这一过程，将其作为练习留给读者。本文将从一个更加宏观的视角出发，建立和定义软件体系结构。

我们首先考虑普通用户是如何使用系统的。普通用户能够预定教室（填写预定请求）、查看预定单、取消预订。为了和系统进行上述交互，我们需要为普通用户设计一个接口来实现人机交互，比如叫做“普通用户接口”。这个接口能够允许用户填写预定请求、填写预定查询请求、填写预定取消请求（输入），并且能够显示预定查询的结果（输出）。接收到上述请求后，接口将这些请求分别转发给相应的模块处理——预定请求交给教室“预定处理”模块处理，查询请求交给“预定查询”模块处理，预定取消请求则交给“预定取消”模块处理。“预定处理”接受预定请求后需要继续调用“预定请求验证”模块对用户的预定请求进行验证，如果验证结果为通过，那么将构造一个预定单，并将其发送给“预定单存储”模块。“预定请求验证”模块在检验预定请求时也需要将进一步调用“预定查询”模块，从而对预定请求进行校验（“预定请求验证”还需要读取教室信息，这里并没有识别出相应的模块）。“预定取消”模块需要调用“预定单存储”模块完成对预定单的取消（实际上是一种修改）。在这里我们将对于订单的增

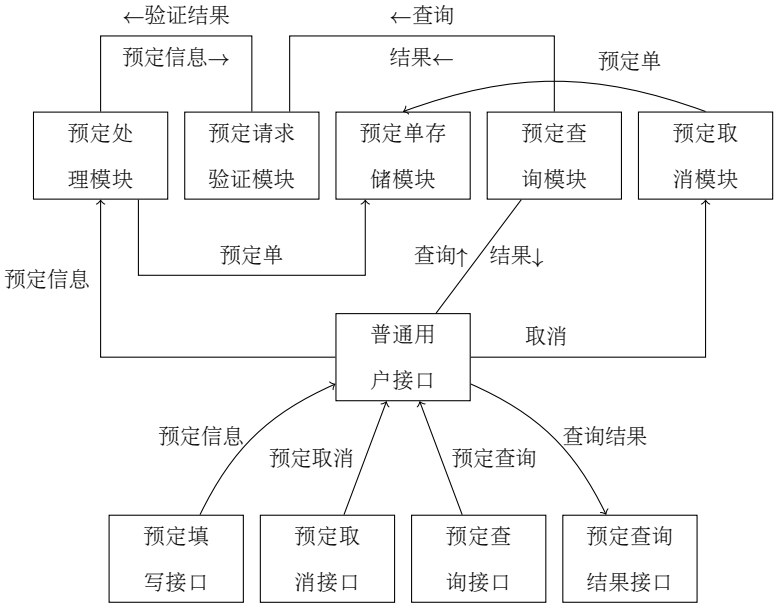

Figure 7: 处理普通用户功能的初始MSD

加、修改操作都封装在“预定单存储”模块中。最终，我们建立了如图7所示的MSD图。

进一步优化 初步建立的MSD可能存在若干需要优化的地方，这主要体现在以下两点：（1）一些模块可能过细，可以将功能上相近或相关的模块合并；（2）一些模块可能粒度过粗，还可以继续分解。我们对MSD进行优化的总体原则是“高内聚、低耦合”原则。比如，在图7中，我们实际上建立了5个模块用来表示用户接口（即“普通用户接口”——作为总控模块、“预定填写接口”、“预定查询接口”、“预定查询结果接口”、“预定取消接口”）。不难想象这几个模块主要是界面、UI，具有一定的内聚性。因此在体系结构设计过程中[4]，我们可以将它们合并为一个“普通用户接口”。

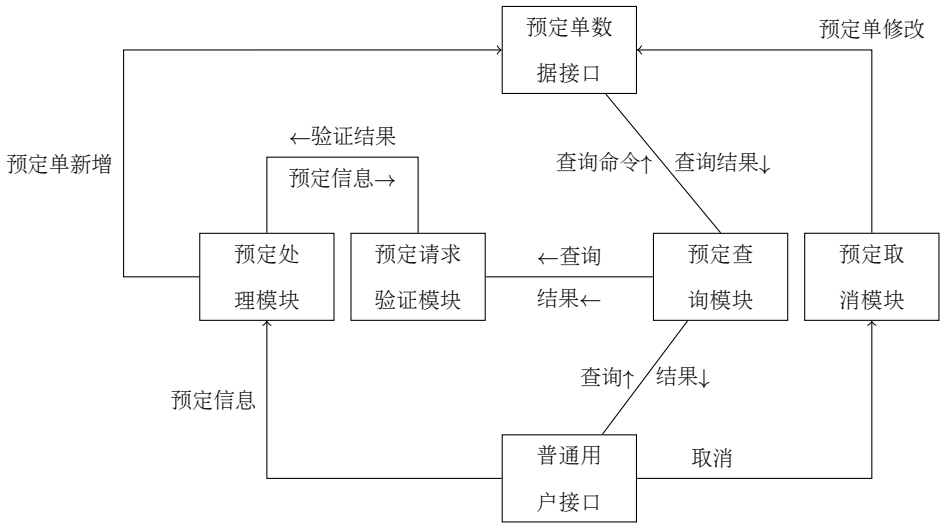

Figure 8: 处理普通用户功能的MSD，符合MVC模式

软件体系结构模式（Software Architecture Pattern） 在体系结构设计过程中，我们可能还需要考虑软件体系结构的整体风格和模式。假设我们采用基于MVC的软件体系结构模式来设计系统，我们需要仔细考虑各个模块的功能划分。在我们的例子中，“普通用户接口”一定是属于视图层的；“预定处理”、“预定验证”、“预定取消”实现了相应的业务逻辑，因此可以看作是控制层；“预定查询”和“预定单存储”模块负责读取、检索预定

[4]体系结构设计依然是比较高抽象层次的设计，因此可以适当忽略一些细节。

单，因此可以看作是与模型层交互的模块（数据接口部分）。按照这样的划分，现有的设计不严格符合MVC模式，这是因为“用户接口”会直接调用“预定查询”（相当于V直接调用了M）。为了解决这个问题，我们可以重新定位“预定查询”模块，将它看作是控制层中的模块，负责处理用户的查询请求。而为了实现这个模块，我们需要将所有的数据访问（存取）进行封装，设计一个新的数据接口“预定单数据接口”，用它处理数据的底层存取操作。经过重新设计，我们可以得到图8所示的MSD。

自己动手 完成管理员部分的模块设计。管理员的情况与普通用户类似，请按照上面的例子完成管理员部分的模块设计，并且将管理员部分及用户部分进行合并，形成完整的软件体系结构模型。

3.3 详细设计

当建立软件体系结构后，就可以进行详细设计了。详细设计可以分成三个主要工作：（1）接口设计，包括用户界面的设计，系统接口设计等；（2）数据设计，可以利用E-R模型对系统需要的数据库（数据存储）进行详细设计，具体请参考数据库相关书籍和课程；（3）构件设计，对软件体系结构模型中出现的所有构件（模块）进行设计，描述它们的接口和实现算法，以及一切与该构件有关内容。

构件设计的关键就是说明构件的实现方法，那么如何说明构件是怎么实现的呢？可以采用以下几种方法：（1）自然语言描述构件的实现逻辑；（2）伪代码描述构件的实现逻辑；（3）NS图；（4）PAD图；（5）流程图；（6）描述构件（中每个操作）的前置/后置条件。我们推荐（1）、（2）和（6）三种方法，方法（3）和（4）操作起来比较复杂，方法（5）也是比较常用的方法（特别是在写论文和技术文档时），但不如（1）、（2）、（6）方便。

我们以“预定处理”模块为例，介绍如何进行构件设计。如果采用第（1）种方法描述构件的实现逻辑，我们可以采用如下描述：在接收到用户的预定请求（包括用户ID、房间ID、日期和时间段后），首先调用预定验证模块对用户的预订请求进行检验，如果没有冲突或错误，创建一个预定单并发送给预定单数据接口进行存储。

如果采用第（2）种方法，我们可以使用下面的伪代码描述该模块：

```
BookingRequestHandler.handle(request)
BEGIN:
  check := BookingRequestVerfication.verify(request);
  IF check = TRUE THEN
    booking := new Booking(Booking.nextID(),
      request.uid, request.rid, request.date, request.time, 0);
    BookingInterface.save(booking);
  END
END
```

如果采用第（3）种方法，我们首先需要分析模块的前置/后置条件是什么。前置条件是指这个模块能够运行所必须满足的条件。从模块的文字描述看，这个模块没有前置条件——即任何情况下该模块都可以执行。后置条件是指这个模块运行后系统必须按满足的条件，也就是模块运行后对这个系统产生的影响或返回的结果。从模块的文字描述看，这个模块执行后会向系统中插入一个预定单，当预定请求是合法的。我们采用OCL[5]的语法形式来描述该模块的前置后置条件，如下：

```
BookingRequestHandler.handle(request)
pre: -- no precondition
post: BookingRequestVerfication.verify(request) = true implies
  Booking::allInstance()->exists(b| b.oclIsNew()
    and b.uid=request.uid and b.rid=request.rid
    and b.date=request.date and b.time=request.time
    and b.state=0)
```

自己动手 作为练习，请完成系统其他模块的设计。

4 新需求

作为进一步的练习，我们对教室预定系统的需求进行一定修改，请利用结构化分析和设计的思想，对新的系统需求进行建模。

[5]请参阅OCL的规范以便了解更多关于OCL的知识。

新的需求主要包括一下几点：

- （1）学生用户能够通过系统进行教室和会议室预定、查看教室和会议室预定情况、取消自己所预定的教室和会议室。教室预定分为临时预定和长期预定，临时预定采用抢占式，无需审核；长期预定需要管理员审核。学生用户只能对会议室进行临时预定，且需要得到管理员审核。

- （2）教师用户能够执行学生用户的全部功能，且教师用户可以对会议室进行临时预定而无需管理员审核。教师用户可以对会议室进行长期预定，但需要管理员审核。

- （3）管理员能够审批教室和会议室预定请求，特别的，如果管理员批准一个预定请求，他/她所批准的教室预定计划可以和预定请求不同（比如，预定请求是“在未来1年中，每周一上午借用1号会议室”，批准后的结果可以是“在未来1个月中，每周一上午借用2号会议室”。管理员用户能够强制取消教室已经批准的教室预定请求、修改教室信息和状态。