第5章

领域驱动的服务设计

对于服务请求者而言，无论其是查询信息的个人还是调用服务的网络应用，需要的都是数据、图像、算法结果或其他功能，这些就是所谓的资源。如果对某一资源的请求存在共性，那么可以把这个功能设计为服务以实现共用。

领域可以被理解为软件需求分析中业务场景对应的业务域，每一个领域又可被分为问题域和解决方案域。

领域本身可以进一步划分为子领域，限界上下文定义了子领域的边界，限界上下文的目的就是厘清子领域，然后区分这些子领域中哪些是核心域、支撑子领域和通用子领域等。

软件系统来自于不同的现实应用领域，其所操作的资源也因具体的业务而各不相同，正如开发软件需要理解业务一样，开发服务也需要深入实际业务去理解资源。这种“理解”包括哪些实体可以作为资源？资源如何表述？如何操作这些资源？这是在设计服务系统时需要首先考虑清楚的问题。

5.1 领域模型与领域驱动设计

1. 领域模型

之所以需要开发软件系统，通常是因为实际业务中遇到了问题，希望通过软件系统解决问题。通过对问题的分析，可以知道需要一个什么样的系统，进一步可以思考如何设计与实现这个系统。

软件系统的需求通常对应某个特定的领域，如银行业务系统对应的是银行业务领域。这种领域本质上可以被理解为包含若干问题的一个问题域，其中的核心问题往往对应着该领域的核心业务，对一个领域而言，这些都是确定的。

领域也不是无限大的，它有边界，设计软件系统时有必要把问题限定在这种边界里。只要能够确定系统所属的领域，那这个系统的核心业务（即要解决的关键问题、问题的范围边界）就基本被确定了。一般在领域专家的指导下，设计人员、开发人员和用户能够在不断交流的过程中发现和挖掘该领域的主要概念，然后以某种各方都能理解的“通用语言”作为交流的工具，将这些概念设计成一个领域模型。这种为描述领域中的核心问题而建立模型的过程就是领域建模。

2. 领域驱动设计

领域驱动设计（domain-driven design，DDD）是一种建模方法，其针对一个领

域内各个业务需求进行建模，本身就是要完成从问题域到解决方案域的映射和抽象，它同时提供了战略（宏观）和战术（细化）层面的建模方法及工具。

建模的宏观层面首先利用领域限界上下文表示业务模型，并通过上下文之间的映射集成多个限界上下文，多个限界上下文结合在一起可以表示领域内一个完整业务的实体、行为、接口等方面。

划分上下文边界是解决方案域的一个关键内容，在领域驱动设计中首先要完成的就是划分上下文边界，一个领域模型的核心域、子域也可以被表达为若干子领域模型，这样一层层嵌套下去，符合传统软件分析设计方法中的子系统或组件划分思路。

领域的战术设计层面着重描绘限界上下文内的细节。限界上下文中包括很多领域对象，为实现某种功能而体现出“高内聚”的特性；而限界上下文之间的边界则体现了“低耦合”的特性。

可以看到领域驱动设计的主要任务有两个：一是发现系统中的聚合（aggregate）；二是划分限界上下文（bounded context）。这两个元素是领域驱动设计的核心概念，分别对应了单个业务功能模块内核心的领域对象建模，以及划分业务功能的边界。这种建模方法可以方便开发者和领域专家更好更快速地配合进行开发。

聚合是一组相关的领域对象（或者称为实体），是由业务和逻辑上紧密关联的事物和值对象二者组合而成的。所谓的值对象 Value Object 类似一座大厦的地址，也可以作为一个有价值的信息用于查询。

聚合用来将若干事物和值组织在一起，但并不是简单地将对象组合在一起，而是要确保业务规则在边界内的稳定性。每个聚合都有一个根实体，被叫作聚合根，聚合根具有全局标识，所有对聚合根内对象的修改都只能通过聚合根实现。聚合内有一套不变的业务规则，各实体和值对象按照统一的业务规则运行，实现对象数据的一致性，边界之外的任何东西都与该聚合无关，这就是聚合能实现业务高内聚的原因。

领域建模的过程包括理解用户行为、找出领域对象和聚合根、对实体和值对象进行聚类组成聚合、划分限界上下文以及建立领域模型。

建模时，第一步需要根据业务行为梳理出发生这些行为的所有实体和值对象，按照功能/模块/业务对项目进行划分，将符合同一个功能/模块/业务的实体、值对象找出来，聚合到同一个领域内。

第二步，从众多实体中选出适合作为对象管理者的根实体，也就是聚合根。判断一个实体是否是聚合根需要结合场景分析，具体包括：实体是否有独立的生命周期？是否有全局唯一的 ID？是否可以创建或修改其他对象？是否有专门的模块来管理这个实体？

第三步，根据业务单一职责和高内聚原则，找出与聚合根关联的所有紧密依赖的实体和值对象，构建出一个包含唯一聚合根与多个实体和值对象的对象集合，这个集合就是聚合。

第四步，梳理聚合内实体之间的复杂关系，根据聚合根、实体和值对象的依赖关系画出实体及值对象间的引用和依赖模型。

最后根据业务语义和上下文将多个聚合一起划分到同一个限界上下文内。

领域建模可以借助可视化建模工具实现，如有领域的上下文（图 5.1），则上下文中的实体与关系如图 5.2 所示。

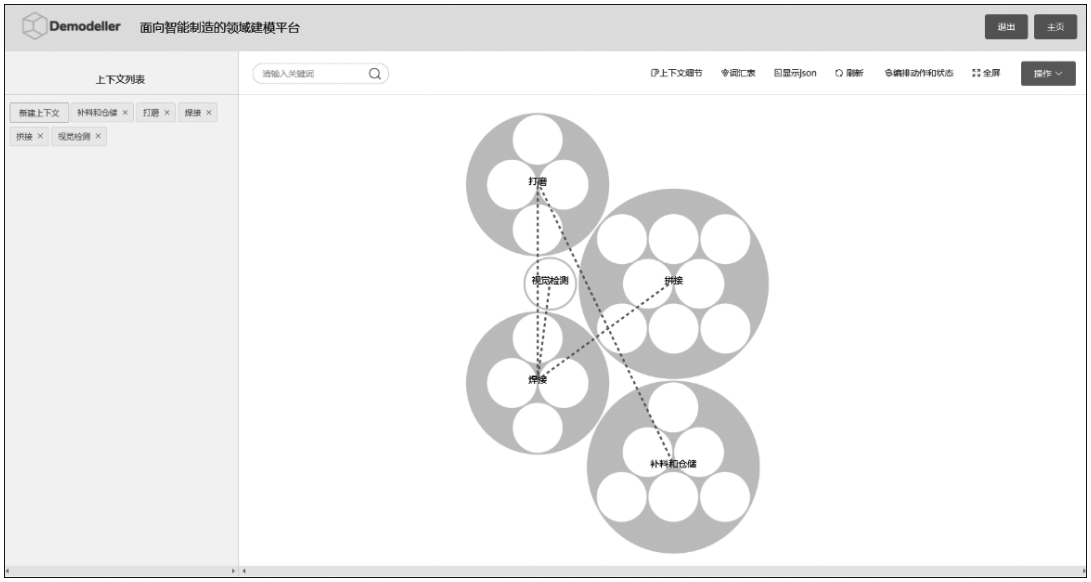

图 5.1　领域的上下文

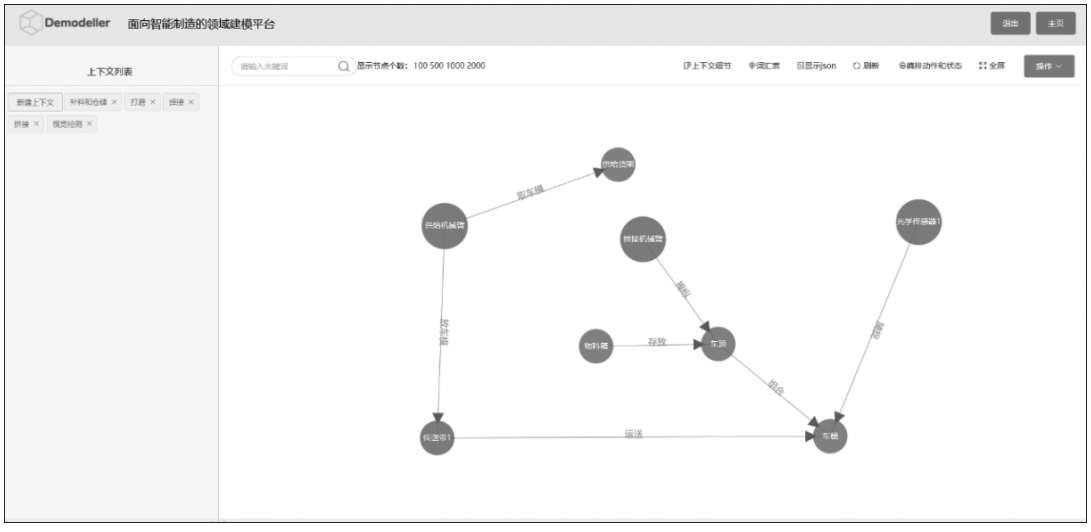

图 5.2　上下文中的实体与关系

5.2　理解领域、识别资源、划分服务

面向资源进行 RESTful 服务设计，识别和定义资源时可以参考领域设计的思路，首先定义领域对象，将领域对象建模为对应的资源，然后再考虑这个资源应该暴露哪些功能接口。

1. 导入场景

在进行 Web 服务设计时首先要考虑用户的功能需求，还要理解实现这些功能的逻辑，这就需要将功能放入服务的场景去分析。下面以一个电商网站中经常出现的“顾客订单处理”场景为例。

（1）电商网站创建商品 SPU（standard product unit，即标准化产品单元，是商品信息聚合的最小单位）。

（2）电商网站创建商品 SKU（stock keeping unit，商品的最小库存单位，商品的进货、销售、售价、库存等最终都是以 SKU 为准）。

（3）电商网站按照 SKU 增加商品库存。

（4）顾客创建订单，电商网站锁定库存。

（5）顾客支付订单，电商网站扣减库存。

（6）顾客取消订单，电商网站恢复库存。

（7）电商仓储发货。

（8）顾客评价、投诉等。

2. 理解事件

根据领域中业务行为中的事件可以梳理系统的数据和行为，从而进行合适的建模。

电商平台系统的事件如图 5.3 所示，在这里可以尝试理解“订单处理”这个应用所属的领域，以及可能发生的事件和涉及的数据和行为。

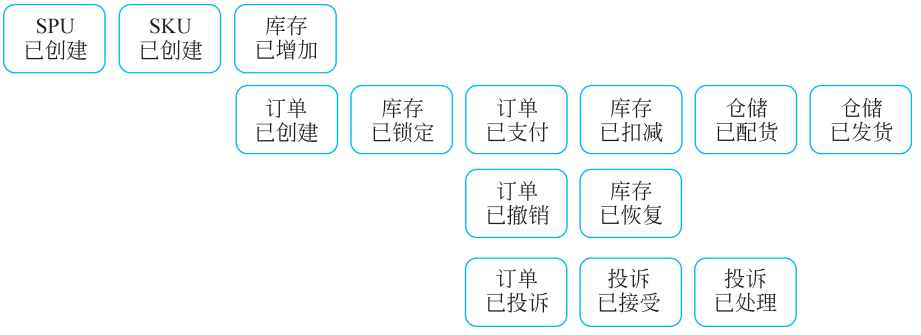

图 5.3　电商平台系统的事件

进一步带入角色，理解不同角色在系统中的操作（图 5.4）。

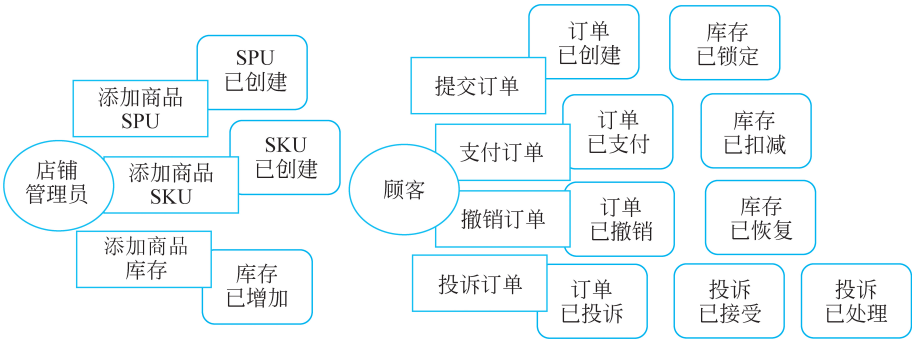

图 5.4　理解电商平台不同角色用户在系统中的操作

3. 聚合对象

聚合是一组相关的领域对象，其目的是确保业务规则在边界内的稳定性。聚合根具有全局标识，所有对聚合根内对象的修改都只能通过聚合根进行。在识别聚合时，可以通过对命令和事件的划分找到聚合边界。电商平台的聚合对象如图 5.5 所示。

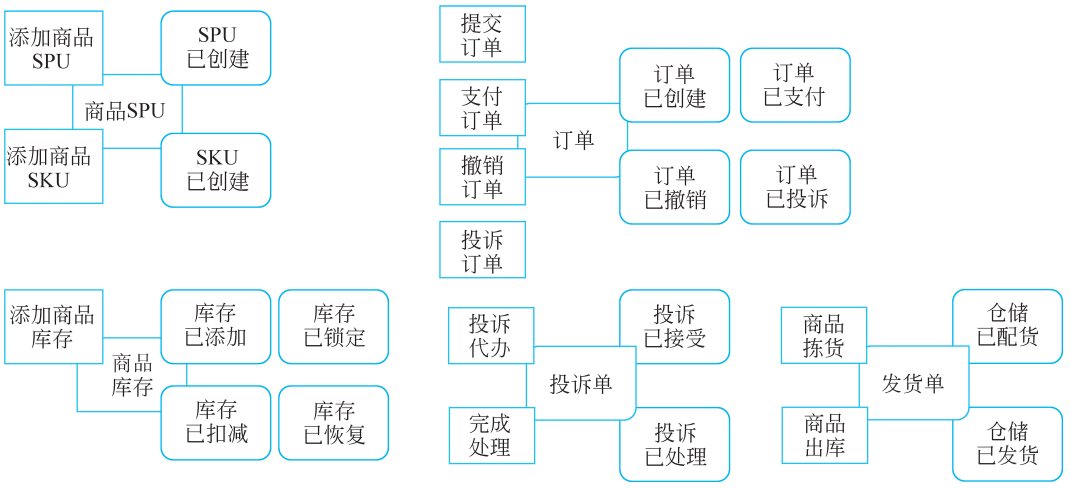

图 5.5 聚合对象

4. 划分边界

在一定程度上服务边界对应的就是“限界上下文”，它有一个非常形象的定义：细胞之所以会存在，是因为细胞膜定义了什么在细胞内，什么在细胞外，并且确定了什么物质可以通过细胞膜。聚合可能是最小粒度的限界上下文，同时，人们经常需要合并业务相关性很高的聚合。电商平台的限界上下文如图 5.6 所示。

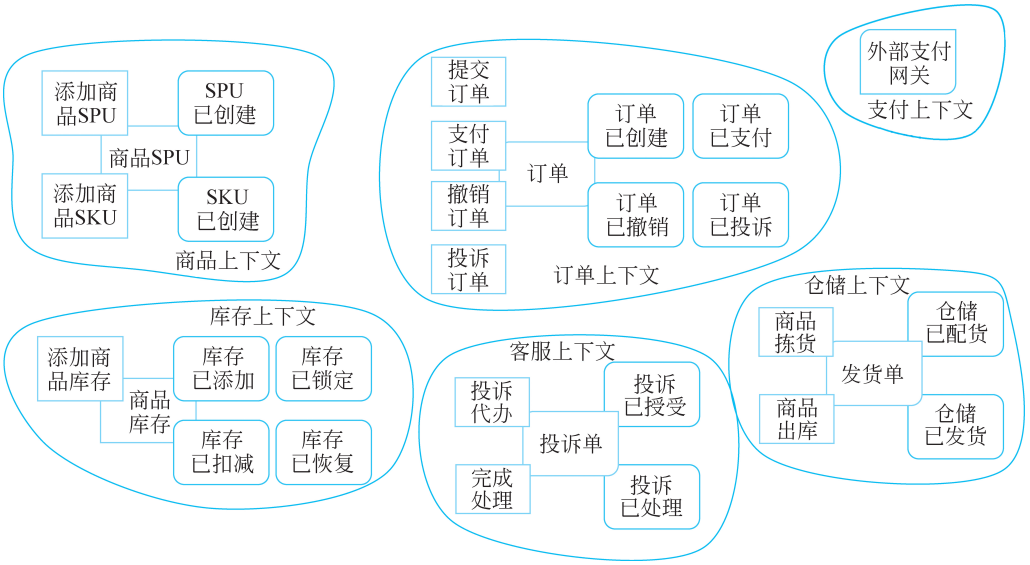

图 5.6 限界上下文

在领域驱动设计中，如果聚合设计得过大，则其会因为包含过多的实体而导致实体之间的管理过于复杂，高频操作时会出现并发冲突或者数据库锁，最终导致系统可用性变差。而小聚合设计则可以避免由于业务过大导致的聚合重构，让领域模型能更适应业务的变化。

5. 识别资源

在设计面向服务的系统时，设计的主要是服务，但仍然要从资源入手，即设计服务应提供的资源？

资源即实体，实体即对象，这些对象代表的是业务对象，有明确的业务含义，类似供应商、采购订单、产品、合同等。同时这些对象本身存在关联和递进的层次结构，如供应商有对应的联系人，有对应的银行账号，产品可能有对应的维修记录等。这些业务对象正是在领域驱动设计时候经常会识别的领域对象。

为实现服务内聚合之间的解耦以及未来以聚合为单位的服务组合和拆分，应避免调用跨聚合的领域服务和关联跨聚合的数据库表。

现实世界中每个具体事物都一定会有唯一的标识，例如，一张火车票，如果具体到日期、车次、到站地点和座位号，那就是一个独立的实体，座位号是其唯一的标识。但如果设计的车票服务系统主要是提供车次以及余票查询服务，那么由于只需要关心剩余座位数，则并不需要以座位号为唯一标识，日期、车次才是最需要被关注的。这里的关键点是实际的业务场景和需求是否需要管理到唯一标识，所以实体划分跟业务需求紧密相关。此外，是否将值对象设计为资源也要看具体的场景。

6. 划分服务

理想情况下，限界上下文与微服务可以一一对应，但在实际项目中，又需要根据业务做一些灵活的调整，包括将多个限界上下文合并，对应的就是将相对简单的服务合并在一起。但一般而言，聚合是服务的最小单元（一个限界上下文可以包括多个聚合），打破聚合，就很有可能破坏事务一致性和业务约束。

如果粗粒度的、体现业务价值的接口服务全部都变成了数据库访问类细粒度接口服务，那么接口就失去了其本身的意义，同时又会导致其本身应该完全内聚在服务内部的业务逻辑全部被暴露到外层。如果一个资源完全不需要和外部模块或外部应用打交道，那么其完全不用开放任何接口，这一方面能提升性能，另一方面也能减少各类难以应对的分布式事务问题。

7. 在边界上定义接口

下面考虑这个资源应该暴露哪些能力接口？对于上面的电商订单场景，可将其拆分为顾客生成新的商品订单、顾客对已有的商品订单进行修改、顾客查询商品订单集合、顾客查看某个特定商品订单的明细数据等业务场景。基于商品订单资源可以设计如下接口需求。

（1）创建新的顾客订单：POST/Orders。

（2）修改一张 ID 为 1111 的已有订单：PATCH /Orders/1111。

（3）删除 ID 为 1111 的已有订单：DELETE/Orders/1111。

（4）查询所有顾客订单：GET/Orders。

（5）查询 ID 为 1111 的顾客订单：GET/Orders/1111。

如果没有按照领域对象的方式定义资源，那么最容易犯的错误就是将所有的数据库表对象都全部定义为一个个独立的资源，并将这些资源的增、删、查、改操作全部暴露为 GET、PUT、POST 和 DELETE 接口方法，那么这样暴露出来的 HTTP REST 接口方法将全都是细粒度的接口。

根据领域驱动开发的通用设计原则，实际开发中还需要考虑项目的具体情况，综合便利性、高性能、事务管理等影响因素，以解决实际问题为出发点灵活运用。

5.3 理解行为、设计表述

在美剧《生活大爆炸》中，主角谢尔顿和他的朋友们下班回到租住的公寓后，经常玩一款叫作《龙与地下城》(*Dungeons & Dragons*，D&D)的游戏。D&D 属于桌面角色扮演游戏(tabletop role-playing game，TRPG)，这种游戏的基本玩法是玩家扮演不同角色，在一个丰富的幻想世界中冒险。游戏中玩家可以自选角色，在城主(dungeon master，DM)给出的故事情节、场景地图、怪物等剧情元素中根据官方制定的规则与 DM 一起游戏，通过掷骰子的方式来进行诸如战斗等动作，完成升级、打怪等任务。D&D 追求完善和复杂，有着精细的设计和繁复的场景，以至于其游戏规则相当复杂，官方规则书动辄数百页。

下面将以 D&D 为参照设计一个极简版的 TPRG，游戏借用 D&D 的基本设定，但适度简化其规则，简称 NDnD。

(1) 游戏中只有一个单独的关卡，即一个场景，包括一张平面地图，地图被平分为 25 个单元格，左上角是关卡的入口，右下角是通关的出口，玩家每次只能移动一个单元格。

(2) 简化游戏为单人游戏，定义玩家是一位骑士，保留游戏中各种怪物(如地精、恶龙、象人、巨噬鲨、巨蜈蚣等)，以及悬崖、河流、火山、陷阱等障碍。骑士有初始的能力值，包括战斗技能值、跳跃技能值，分别用于打怪和越过障碍。怪物都有自身的战斗技能值，各种障碍有不同的难度值。

(3) 每一个单元格中会有一种怪物或障碍，也可能是安全的平地，骑士遇到怪物可以选择战斗，战斗规则是简单的比拼能力值，例如，骑士的战斗技能是 10，现在面对一个战斗技能是 20 的怪物，这时玩家需要扔一个 20 面骰子，得到的随机数值加上 10(骑士的战斗技能)，如果结果大于 20 则骑士获胜，反之则会失败。骑士获胜以后会获得怪物战斗技能的 10%(增加到其战斗技能中)，并可以选择行进方向继续前进；如果失败则骑士自身战斗技能会损失 10%，并回退到上一关；越过障碍的游戏规则类似。

(4) 游戏中的随机事件由掷骰子来决定，通过随机值增加游戏的不确定性，并推动故事的发展。

下面通过 NDnD 这样一个简单的案例理解问题领域中的行为和设计表述。

首先是如何表达地图场景。如图 5.7 所示，游戏地图可以被抽象成一个由单元格构成的网格，每个单元格是一个最小单元，用 A～Y 的字母标记。平面网格在几何上有所谓“四连通”(指对应单元位置的前、后、左、右共 4 个方向连通)、八连通(指对应位置的前、后、左、右、左前、右前、左后、右后共 8 个方向连通)。考虑表述得简洁，这里把地图设计为四连通，即游戏玩家只能从当前单元格向前、后、左、右四个方向移动。

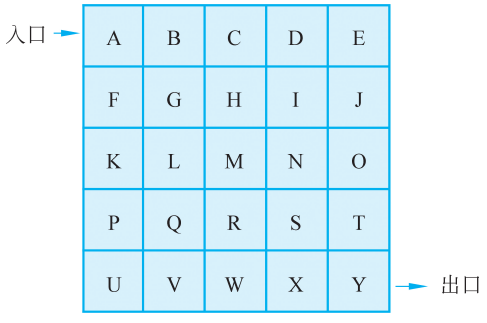

图 5.7 游戏地图的一种抽象

整个地图场景对玩家是不透明的，这样会使冒险过程充满不确定性，增加挑战性与玩游戏的乐趣。在任何时刻，身处游戏内部的骑士都看不到场景的全貌，而仅能知道当前单元格中的场景(客户端可以用简笔画描绘这个场

景，如图 5.8 所示）以及当前单元格与相邻单元之间的连通关系。

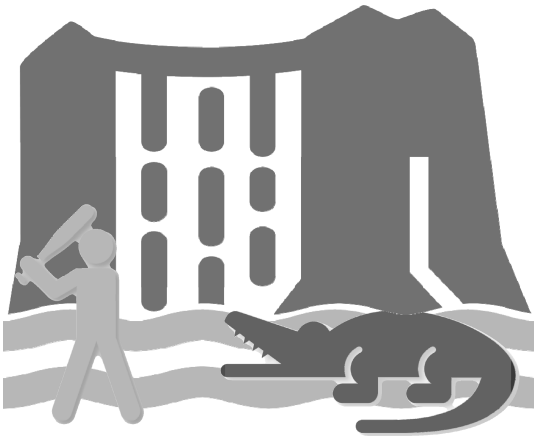

图 5.8　NDnD 中的一个游戏场景

据此，对游戏的理解已经形成了一些共识。

（1）游戏需要且只需要为玩家呈现当前单元格的场景，包括该单元格中有什么怪物或者障碍、可以连通的其他单元格是哪些。

（2）玩家根据当前单元格的场景确定下一步的行动，可以打败怪物/越过障碍然后继续前进，也可以退回上一个单元格。

（3）玩家知道自己当前的战斗技能值和跳跃技能值，但不知道怪物或障碍的技能值，掷骰子的结果也是随机的，这个战斗过程由后台服务计算完成。

（4）战斗结果由后台服务反馈给玩家，玩家技能值的增加或者减少也要反馈给玩家，再由玩家根据结果判断局势并确定下一步的行动。

下面可以考虑表述的设计了。这里借用一个已有的设计——Maze＋XML。Maze＋XML 是一个 XML 格式的数据格式，用于描述简单的迷宫（Maze）类游戏数据，其 MIME 类型为 application/vnd.amundsen.maze＋xml。在 Maze＋XML 的基础上扩展，将每个地图单元格都设计成一个拥有独立 URL 的 HTTP 资源，每当游戏玩家进入新的单元就向服务端发送一个 GET 请求，服务端就会给客户端反馈一个表示当前单元格的表述，如下所示。

```
<NDnD version="1.0">
  <cell href="/cells/M" rel="current">
  <title>峡谷地</title>
     <monster>
        <name>史前巨鳄</name>
        <link rel="fight" href="/fightWithMonster/crocodile"/>
     </monster>
     <link rel="east" href="/cells/N"/>
     <link rel="west" href="/cells/L"/>
     <link rel="south" href="/cells/R"/>
     <link rel="north" href="/cells/H"/>
  </cell>
</NDnD >
```

这条表述包含了当前单元格地址"/cells/M"，一个显示给玩家的单元格名字：“峡谷地”，这个名字其实没有具体含义，只是便于玩家记忆以及增加一些游戏中的临场感。这里

用 link 标记将单元格与它附近的其他单元格连接起来，注意，在 Maze+XML 格式中，方向是用东、西、南、北来表示的，就像人们使用地图的习惯一样，即“左西右东，上北下南”，从单元格 M 开始，玩家可以选择向西走进入单元格 L，也可以选择向东进入单元格 N；monster 标记是笔者自行扩展的，表示当前玩家遇到的怪物，同样，这里也增加了一个与怪物格斗的链接。

玩家可以从反馈的表述中选择自己的行动，首先，他可以选择战斗，即向/fightWithMonster/crocodile 链接再发送一个 GET 请求，后台服务器会将之理解为玩家要与史前巨鳄决斗一番，并获取 fightWithMonster 的结果，这里服务器只需简单反馈两个数值就可以了，如下所示。

```
<NDnD version="1.0">
  <fightWithMonster >
      <Dice>8</Dice>
      <Score>-2</Score>
  </fightWithMonster >
</NDnD >
```

其中 Dice 标记将调用后台一个随机算法，计算出一个玩家掷骰子的数字，Score 标记是战斗结果，负数表示玩家失败了，并损失 2 点战斗技能。

这样设计的表述实际上是把当前的状态都反馈给了客户端，再由客户端去设计游戏的交互逻辑，如下所示。

首先 Dice 标记的值可以先反馈给用户，可以用可视化的形式，例如，显示一个旋转的骰子最后将值停止在 8，也可以直接显示一个数值；然后也可以可视化地展现战斗的结果，模拟一个打斗的过程，或者也只是显示一个文字；玩家获得/失去的技能值也应显示给玩家。

然后，客户端要根据战斗结果决定显示给玩家的其他链接选项，如果胜利了，玩家会看到连通相邻单元格的链接，进入新的冒险历程；如果失败了，而其技能值还是正数，玩家可以选择再战，因此 fight 链接还在，或者选择退回上一个单元格；因为允许玩家回退，如果回退到已经取得胜利的单元格，玩家进去后看到的是平地，没有任何怪物/障碍，但是连通周围单元格的链接关系还在，这里需要客户端做好标记。

假设玩家获胜了，客户端会显示可选的前进方向，玩家自己选一个（如向东走），则客户端对单元格 N 发出一个 GET 请求，并收到新的表述，内容如下。

```
<NDnD version="1.0">
  <cell href="/cells/N" rel="current">
  <title>大瀑布</title>
    <barrier>
        <name>激流</name>
        <link rel="conquer" href="/conquerTheBarrier/torrent"/>
    </barrier>
    <link rel="east" href="/cells/O"/>
    <link rel="west" href="/cells/M"/>
    <link rel="south" href="/cells/S"/>
    <link rel="north" href="/cells/I"/>
  </cell>
</NDnD >
```

客户端的应用状态会因玩家的操作发生变化。借用 HTML 标准中的术语，客户端刚才在“访问”单元格 M，现在它正在“访问”单元格 N。

barrier 标记也是笔者扩展的元素，表示在当前单元格玩家遇到了障碍；同样，这里也增加了一个征服障碍的链接：/conquerTheBarrier/torrent，客户端把这个链接表述给玩家后，玩家可以选择征服，即向该链接再发送一个 GET 请求，后台服务器会将之理解为玩家要利用自己的跳跃技能征服障碍，并获取 conquerTheBarrier 计算的结果。这实现起来也很简单，服务器也只需反馈两个数值就可以了，代码如下。

```
<NDnD version="1.0">
  <conquerTheBarrier>
      <Dice>17</Dice>
      <Score>1</Score>
  </conquerTheBarrier >
</NDnD >
```

同样，这里的 Dice 标记将调用后台一个随机算法，计算出一个用户掷骰子的数字，Score 标记是征服结果，若玩家成功地跨越了激流将获得 1 点跳跃技能。

之后，客户端会为玩家显示可选的前进方向，玩家继续前进，经过一系列的探险，最终来到 Y 单元格。Y 单元格名字叫“龙巢”，游戏设计的惯例会在最后一关放一个大 BOSS，但这里也包含了游戏通关的出口，这个出口是通过一个链接关系为 exit 的 link 标记来表明的，如下所示。

```
<maze version="1.0">
  <cell href="/cells/Y">
    <title>龙巢</title>
    <monster>
        <name>恶龙</name>
        <link rel="fight" href="/fightWithMonster/dragon"/>
    </monster>
    <link rel="west" href="/cells/X"/>
    <link rel="north" href="/cells/T"/>
    <link rel="exit" href="/success.txt"/>
  </cell>
</maze>
```

当然，玩家需要先战胜恶龙，在此之前，客户端是不会把出口显示出来的。

5.4　客户端与服务端的设计

NDnD 的表述设计还有一些其他内容，如整个游戏的入口。进入不同关卡的链接等。但基本的行为已经有了，从一个单元格选择与怪兽战斗或者征服障碍，成功后访问链接进入下一个单元格，最终发现一个通关标记为 exit 的链接。这些信息已经足够实现客户端了。

这个服务最重要的用途就是开发供人类玩乐的游戏。下面先介绍一个只需要用文字呈现 NDnD 布局的极简客户端，这有点类似 MUD 游戏（multiple user domain，多用户虚拟空间游戏）。

这个游戏将从获取一个游戏场景集合开始，让玩家从中选择一个场景。一旦玩家进入