协作行为分析和设计

邵栋

面向对象分析与设计

- 面向对象分析和设计中如何分析类与类之间的关系、设计它们之间的职责与合作以共同完成软件功能，是软件设计的核心问题。
- UML：面向对象分析和设计工具

类的协作

- 面向对象软件系统通过对象之间的互动来完成整个软件的功能。
- 每个对象都具有一些数据以及处理这些数据的逻辑。我们可以认为它们在某种程度上具有一定的决策能力，能够了解并处理一些事情，能够与其它对象协作完成共同的目的。

- 这样可以使变动的影响局部化，从而提高软件的可维护性。

结构化程序设计

- 结构化程序设计风格往往偏好由一个参与者进行大量逻辑的处理，而其它很多参与者仅仅提供数据，采用一种集中式控制的风格。
- 数据和处理数据的算法是分开的，代码放在完全不同的函数或者是过程中。理想状态下，输入数据通过函数和过程从而得到输出，数据放在与函数和过程不同的地方，并由这些函数和过程来操作、管理。
- 上课例子。

面向对象

- 使用对象来模拟现实世界。
- 但并不会照搬现实世界的内容到软件系统中来。
- 我们将真实世界中的对象按照系统的需要设计成易于管理的对象，还可能创建出真实世界中不存在的新对象（比如有时我们根据设计需要，会封装一个命令或行为成为一个对象）。
- 决定设计哪些对象以及对象之间的交互关系取决于软件满足现实的需要，而不是对现实世界的模拟程度。

理解

- 软件= 一组相互作用的对象
- 对象 = 一个或多个角色的实现
- 角色 = 一组相关的责任
- 责任 = 执行一项任务或掌握某种信息的义务
- 协作 = 对象或角色（或两者）之间的互动

对象

- 问题域中的事物（信号、建筑物、汽车、报表等）、和系统交互的实体（人、设备、其它软件等）、系统中人的角色（系统管理员、普通用户等）、与系统有关的组织（公司、团队、小组等）、地点（车间、办公室等）。
- 行为也可以成为对象。比如我们可以将电视遥控器说要执行的一个命令封装成一个对象。

角色、责任、协作

- 对象角色（Role）是对象责任（Responsibility）的体现，责任是指对象持有、维护特定数据并基于该数据进行操作的能力。
- 因此，要求对象有明确的角色就是要求对象在应用中维护一定的数据并对其进行操作，简单的说就是要拥有状态和行为。

- 状态是对象的特征描述，一般可以认为是类的成员变量。

- 对象的行为通常是针对状态的操作，一般表示为类的方法。

- 对象是对现实世界事物的抽象，在应用中履行特定的职责。对象具有标识、状态和行为。

- 协作是对象之间的相互请求，一般表现为对象之间的方法调用。
- 独立的对象能够完成的责任是有限的，面向对象系统需要通过对象之间的相互协作来完成复杂的任务。

变化局部化

- 当使用几个对象共同完成某个责任时，我们需要细致地安排每个对象的责任，并且设计它们相互的协作方式。
- 一个设计良好的软件应当保证变化的局部化，即在发生变化时，软件所需的变动最小。应对软件变更。
- 变动的影响局部化是好的软件设计的一个重要目标。数据与对该数据的操作往往是一同变化的，因此，将数据以及对数据的操作同时封装在一起是面向对象设计的核心原则。

软件设计方法与编程语言

- 初学者要注意，面向对象设计、结构化程序设计都是软件设计的方法，它们和具体的程序设计语言没有直接联系。
- 通常面向对象程序设计语言会有较好的对于面向对象设计机制的支持，比较方便编写面向对象风格的程序。
- 程序员可以用大部分面向对象程序设计语言来编写结构化程序，比如使用java来编写完全结构化程序。也可以使用结构化程序设计语言来编写面向对象风格的程序，比如使用C语言来完成封装、继承等功能。但这样往往会给程序设计带来很多不便。

用例文本描述

- What to do?
- 1992年，Jacobson在Objectory方法中提出使用用例来表述需求.
- 其它需求描述方法：“故事”（敏捷软件开发）

场景

- 场景（Scenario）是用来描述一个用户和系统之间交互的一系列步骤。
- 借书的场景：
  - “用户根据图书名查询图书，确认需要借阅的图书是否有库存，确定借阅。系统检查用户的借阅配额以及借阅权限，并立刻确认借阅。”
  - 这个场景是借阅时发生的一种成功的情况，但也许用户已经借阅的图书已经达到可借阅的上限，无法再借阅新的图书；或者需要借阅的图书已经没有库存了。这些都是新的场景。

用例

- 用例是具有共同用户目标的一系列场景的集合。
- UML [Rumbaugh，2004]将用例定义为“在系统（或者子系统或者类）和外部对象的交互当中所执行的行为序列的描述，包括各种不同的序列和错误的序列，它们能够联合提供一种有价值的服务”。
- 一个用例是所有和用户某一目标相关的成功和失败场景的集合，用例用来记录系统的功能需求。

用例模板

表 9-1 简单的用例描述模板

项目	内容描述
名称	对用例内容的精确描述，体现了用例所描述的任务
参与者	描述系统的参与者和每个参与者的目标
正常流程	在常见和符合预期的条件下，系统与外界的行为交互序列
扩展流程	用例中可能发生的其他场景
特殊需求	和用例相关的其他特殊需求，尤其是非功能性需求

名称	借阅图书
参与者	用户，目标是能够借阅到自己需要的图书
正常流程	1、 用户查询图书，并确定需借阅的图书。 2、 系统检查用户的借阅配额是否使用完。 3、 系统确认需借阅的图书是否有库存。 4、 系统检查用户借阅的图书是否是珍本，用户是否有借阅珍本图书的权限。 5、 系统授权借阅。 6、 用户借阅。 7、 系统向用户表明借阅成功。
扩展流程	2a：用户借阅图书已经达到最大限额。 2a1：系统提醒用户归还图书后借阅。 3a：没有库存 3a1：如果用户是教师，系统提醒可以要求借阅该书的用户在 7 天内归还该图书。 3a2：如果用户不是教师，系统提醒无法借阅。 4a：是珍本图书，并且用户没有借阅珍本图书权限。 4a1：系统拒绝借阅，并给出原因。
特殊需求	给出的提示信息应当足够清晰，告诉借阅者借阅成功，或不成功的原因。

图 9-3 图书借阅用例文本

用例使用

- 在描述用例时，可以有多种形式和内容，通常并没有严格的规定，但某个团队在某个项目中应该保持一致的约定。
  - 比如说，我们可以在用例前加入前置条件，表明本用例只有在前置条件成立时才可以执行；也可以加入主要使用者、最低保证、成功保证等内容。
- 我们可以在用例文本描述中加入任何有助于理解和交流用例的内容。一个开发团队在使用用例时可以考虑设定一个团队共同接受的用例模板。

用例大小

- 使用用例的另外一个重要问题是如何分解用例。比如例子中的第3步没有库存，有些人认为是该用例的一个场景，有些人会认为是一个新的用例。很多时候，这种决定并没有完全的正确与错误的区分，开发人员可以在保证理解和交流的前提下自行决定使用方式。
- 关于在用例文本描述中细节描述程度建议由用例交流和理解中的风险来决定，如果开发人员觉得这个用例的风险较大，就应该在用例中包含更多的细节。
- 在迭代式开发中，我们往往会在开始实现这个用例的迭代周期中细化一个用例。遵循“最后责任时刻”（Last Responsible Moment）的原则，团队等到开始实现该用例时才写下具体的细节。

用例图

- 使用用例图来可视化的描述用例，现在用例图是UML的一个组成部分。
- 用例是一种需求获取的重要方法，用例被用来组织需求、估算、计划、软件设计和测试。

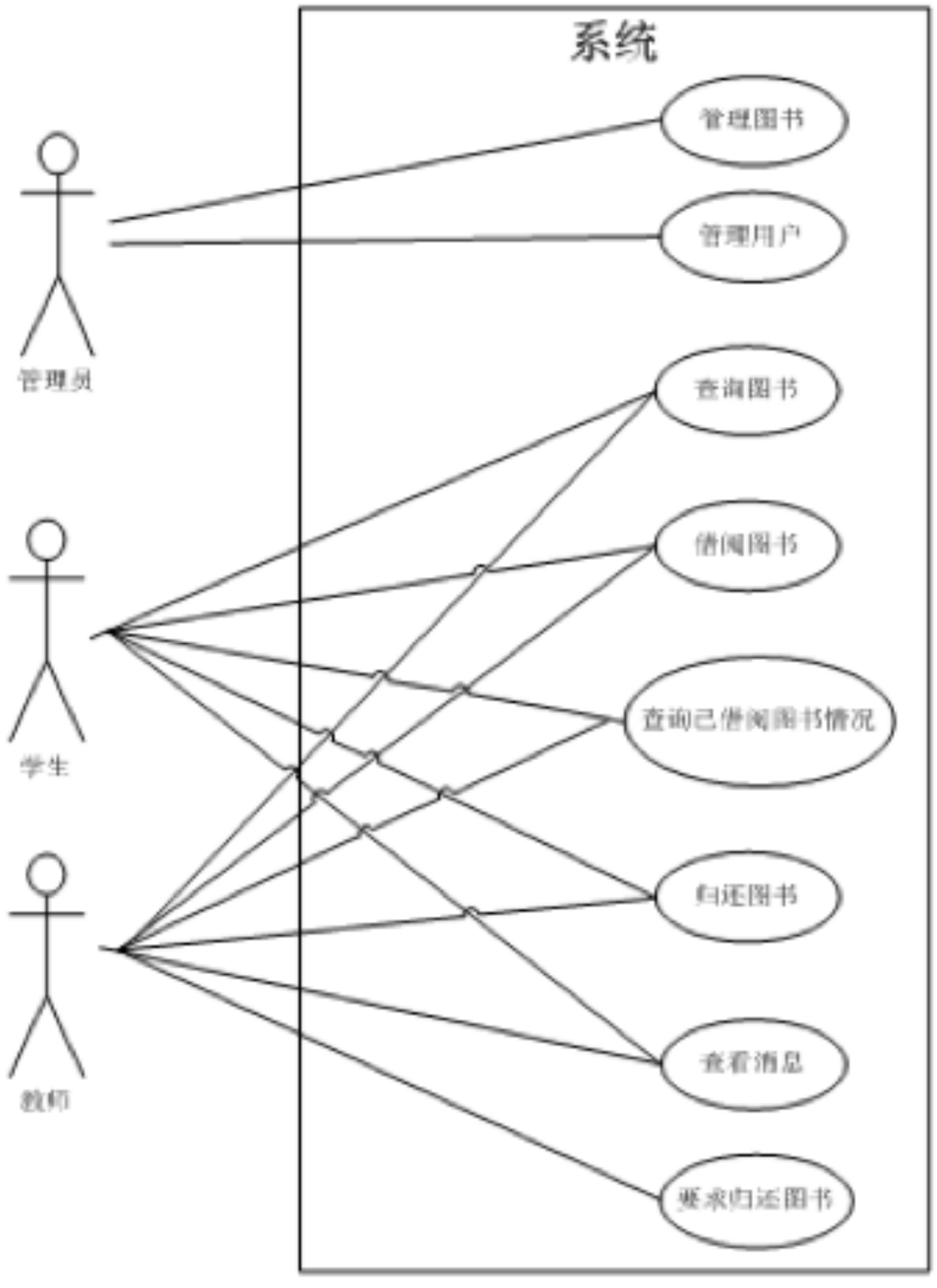


图9.4

- 在构造用例图时，有些人会画出一个用例所有相关的外部用户和系统，有些人选择仅仅显示用例的发起者，也有些人选择画出能够从用例中获得价值的角色。

- 角色在UML用例图中用类似人形图标表示，如图9-4左侧的管理员、学生和教师三个角色。用例在UML用例图中用一个椭圆表示，如图9-4右侧的“管理图书”、“管理用户”等。图9-4中右侧系统方框表明了系统边界，用来表明我们关注的系统部分。

- 当我们面对一个很大的复杂系统时，有时候难于分析清楚系统功能需求，也难以得到系统的用例，这时先分析系统中角色列表往往会容易一些。

- 当确认了系统角色（这时可能多数是具体的用户）后先从最重要的角色开始分析他们的用例，从而捕获到相对完整的系统用例。

- 例如：在图书借阅系统中，我们可以先分析出系统的用户有本科生、研究生、教师、管理员。其中，本科生和研究生使用系统是仅仅有借阅配额的差异，在考虑用例时可以归为一种角色，他们的用例包括：查询图书、借阅图书、查询已借阅图书情况、归还图书、查看消息。而教师涉及的用例除了以上列出的外，还有请求归还图书的用例。

设计类图

- 类图是使用最为广泛的一种UML图。
- 类图中的特性（Property）表示类的结构特征，可以类似理解为程序设计语言中类中的成员变量。
- 属性（Attribute）表示法和关联（Association）表示法。

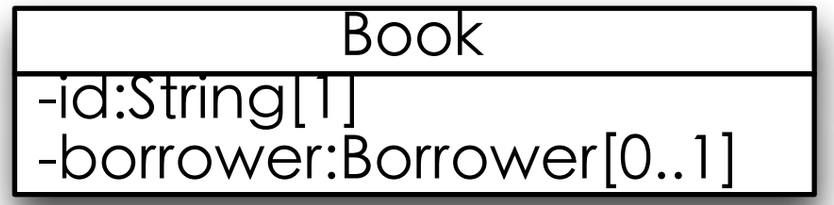


图9-5属性表示法Book类的属性

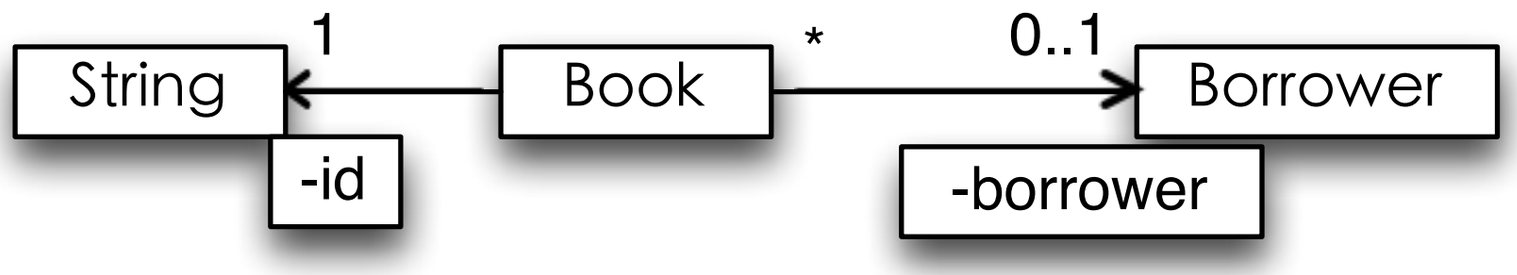


图9-6关联表示法Book类的属性

两种方法等价

- 属性表示法用类图矩形中的一行文字来表示类的一个属性，其语法如下:

- 可见性名称: 类型多重性 = 缺省值 {属性字符串}

- visibility name: type multiplicity= default {property-string}

- 举例如下 :

- -name: String [1] = “Untitled” {readOnly}

- 可见性（Visibility）可以使用public (+), private (-), protected (#);
- 名称通常为类定义中的属性名称
- 类型限制在属性中放入的数据类型，通常为类定义中的属性类型
- 多重性：多重性表示可能会有多少个对象存在
  - 1 （一次续借的发起者只能是一个用户）
  - 0..1 （某本图书当前可能有借阅者，也可能没有）
  - （某本书可能没有在馆图书，也可能有很多，没有上限）
- 缺省值表明的系统创建对象时，如果没有特别指明，我们就将此值指定给该属性
- {属性字符串}允许使用者指明一些额外属性，例子中指明用户不能修改该属性值

- 属性的另外一种表示方式是使用关联。关联用连接两个类图的实线来表示，箭头由来源指向目标类图。属性的名称会出现在关联的目标端，同时加上它的多重性。关联的目标端所连接的类别就是这个属性的类型。

- 在使用类图时，开发者可以使用属性表示法，也可以使用关联表示法。一般来说，建议对重要的类（比如Book, Catalog, BookInfo）使用关联表示法，而对不那么重要的类，比如一些值类型（日期、字符串）使用属性表示法。

操作

- 类图中的操作（Operation），指类可以完成的动作，通常相当于类定义当中的方法。通常开发者在类图中忽略getXXX(), setXXX()等方法。
- 在UML中，其表示语法为：
- 可见性名称（参数列表）：返回值类型{属性字符串}

- 可见性可以是public（+）、private（-）、protected（#）
- 名称是一个字符串
- 参数列表是方法的参数列表
- 返回值类型是方法返回值的类型
- 属性字符串代表可以使用的一些性质（比如可以使用{query}表示仅读取值，但不会修改）

- 参数列表中的参数表示方法和属性类似。其语法如下：
  - 方向性名称：类型 = 缺省值
  - Direction name: type = default value
  - 名称、类型和缺省值都和属性表示中一致
  - 方向性代表参数是用来输入（in）、输出（out）或即输入又输出（inout）。如果没有特别给出，缺省为in。例如：+borrowBook(bookinfo: BookInfo):Message。

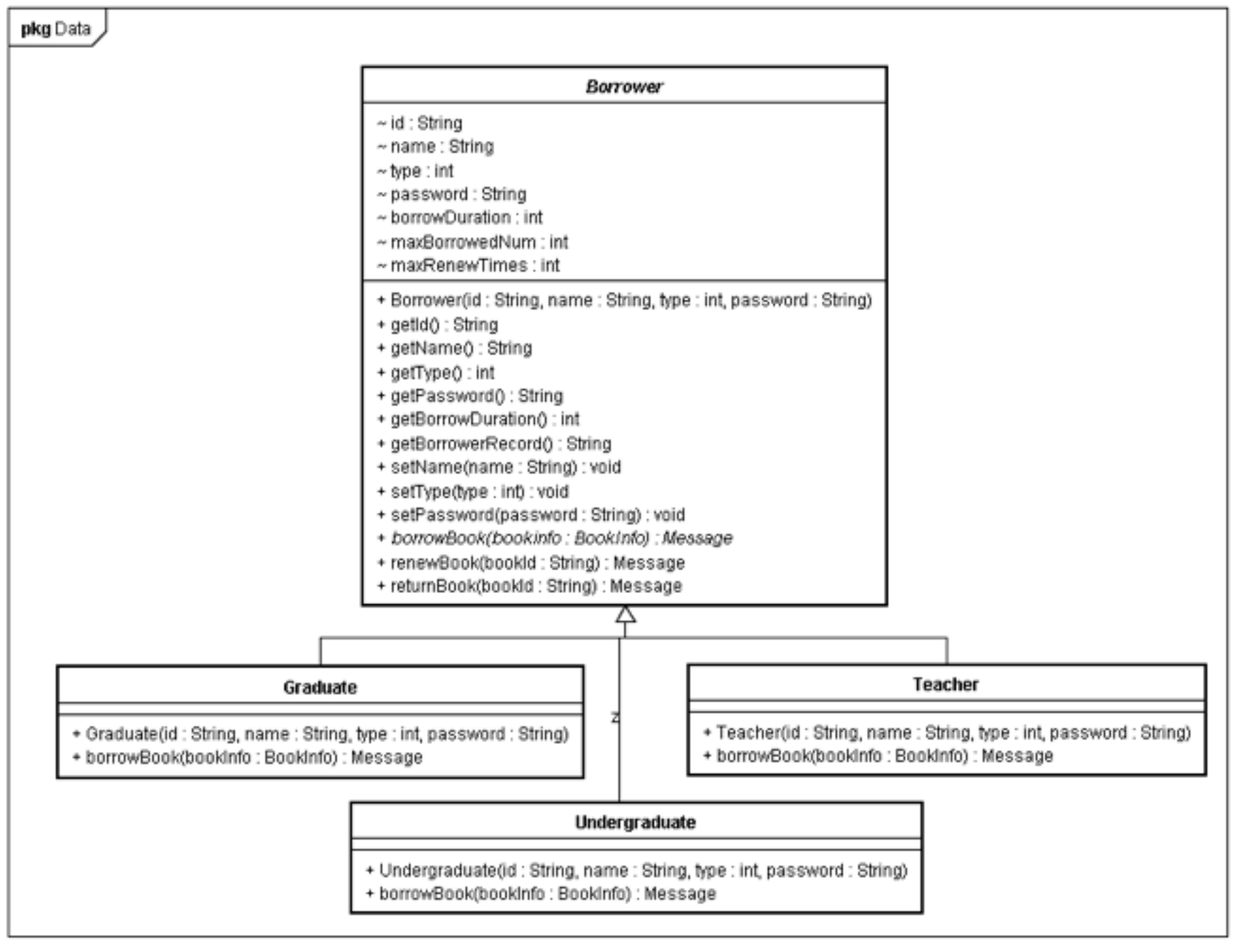
类间关系

- 关联（Association）、泛化（Generalization）、依赖（Dependency）。

泛化关系

- 某些类之间存在一般元素和特殊元素间的关系，特殊元素是一般元素的一个子类型，描述了一种“is-a-kind-of”的关系。
  - 比如说麻鸭是鸭子的一个子类型；而鸭子是动物的一个子类型。
- 泛化关系在编程中一般体现为继承关系，子类继承父类的所有特性。UML中使用带三角箭头的实线表示，箭头指向父类。

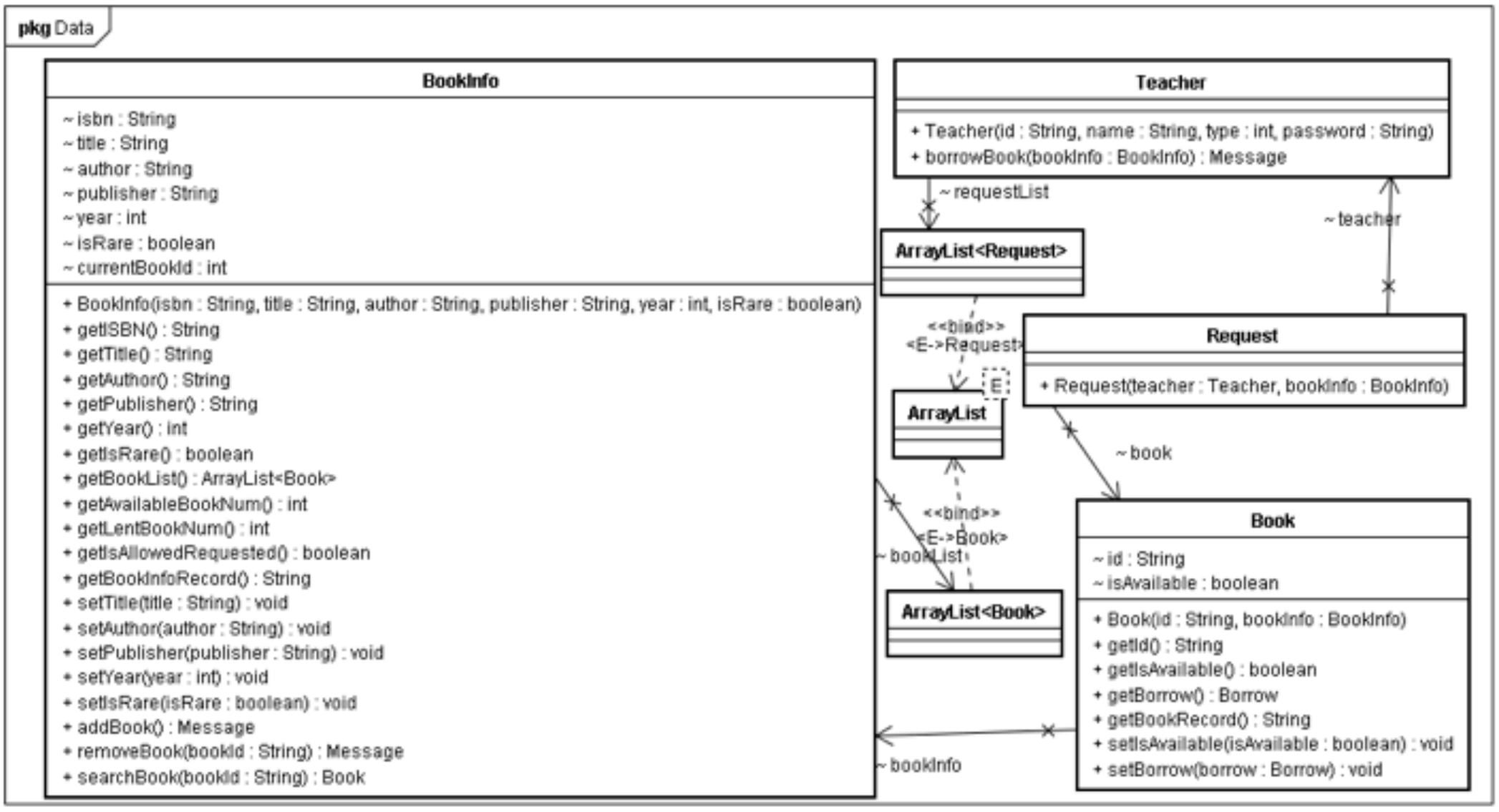

超类或父类
子类


pkg Data
Borrower
~ id : String
~ name : String
~ type : int
~ password : String
~ borrowDuration : int
~ maxBorrowedNum : int
~ maxRenewTimes : int
+ Borrower(id : String, name : String, type : int, password : String)
+ getId() : String
+ getName() : String
+ getType() : int
+ getPassword() : String
+ getBorrowDuration() : int
+ getBorrowerRecord() : String
+ setName(name : String) : void
+ setType(type : int) : void
+ setPassword(password : String) : void
+ borrowBook(bookInfo : BookInfo) : Message
+ renewBook(bookId : String) : Message
+ returnBook(bookId : String) : Message
Graduate
+ Graduate(id : String, name : String, type : int, password : String)
+ borrowBook(bookInfo : BookInfo) : Message
Teacher
+ Teacher(id : String, name : String, type : int, password : String)
+ borrowBook(bookInfo : BookInfo) : Message
Undergraduate
+ Undergraduate(id : String, name : String, type : int, password : String)
+ borrowBook(bookInfo : BookInfo) : Message

依赖关系

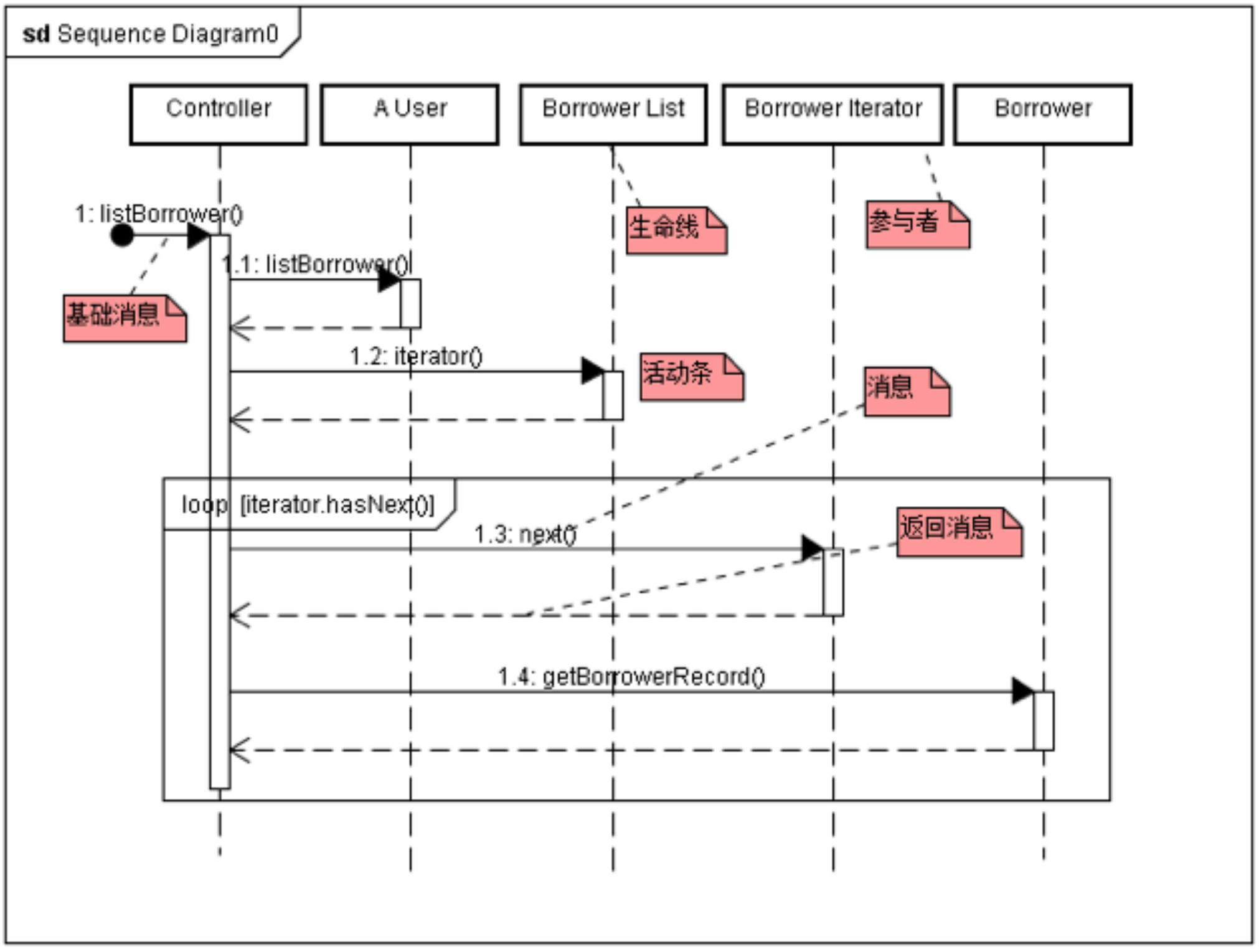
- 是一种使用关系，表现为一样事物的改变会影响到使用它的其它事物。
- 可以是：一个类把消息发送给另外的类；一个类以另外一个类作为其数据部分；一个类使用另外一个类作为操作参数。
- 在编程实践中，经常表现为局部变量、方法参数或者对静态方法的调用等。
- 在UML中使用带箭头的虚线指向被使用者。


提供者
（被使用者）
使用者
Book
Borrower


pkg Data
BookInfo
~ isbn : String
~ title : String
~ author : String
~ publisher : String
~ year : int
~ isRare : boolean
~ currentBookId : int
+ BookInfo(isbn : String, title : String, author : String, publisher : String, year : int, isRare : boolean)
+ getISBN() : String
+ getTitle() : String
+ getAuthor() : String
+ getPublisher() : String
+ getYear() : int
+ getIsRare() : boolean
+ getBookList() : ArrayList<Book>
+ getAvailableBookNum() : int
+ getLentBookNum() : int
+ getIsAllowedRequested() : boolean
+ getBookInfoRecord() : String
+ setTitle(title : String) : void
+ setAuthor(author : String) : void
+ setPublisher(publisher : String) : void
+ setYear(year : int) : void
+ setIsRare(isRare : boolean) : void
+ addBook() : Message
+ removeBook(bookId : String) : Message
+ searchBook(bookId : String) : Book
Teacher
+ Teacher(id : String, name : String, type : int, password : String)
+ borrowBook(bookInfo : BookInfo) : Message
~ requestList
~ teacher
ArrayList<Request>
<<bind>>
<E->Request>
E
ArrayList
Request
+ Request(teacher : Teacher, bookInfo : BookInfo)
~ book
<<bind>>
<E->Book>
~ bookList
ArrayList<Book>
Book
~ id : String
~ isAvailable : boolean
+ Book(id : String, bookInfo : BookInfo)
+ getId() : String
+ getIsAvailable() : boolean
+ getBorrow() : Borrow
+ getBookRecord() : String
+ setIsAvailable(isAvailable : boolean) : void
+ setBorrow(borrow : Borrow) : void
~ bookInfo

顺序图

- 交互图（interaction diagrams）的一种。还包括通信图和时序图。
- 顺序图可以清晰的指明参与者之间的调用关系，但是顺序图并不善于表示循环、条件分支等算法细节，这些内容更加适合使用活动图或代码本身表示。


sd Sequence Diagram0
Controller
A User
Borrower List
Borrower Iterator
Borrower
1: listBorrower()
生命线
参与者
1.1: listBorrower()
基础消息
1.2: iterator()
活动条
消息
loop [iterator.hasNext()]
1.3: next()
返回消息
1.4: getBorrowerRecord()