执行之后将返回 Rowx 结构，这种结构拥有 StructScan 方法，该方法可以将列自动地映射到相应的字段里面。另一方面，对于 Create 方法，我们还是跟之前一样，使用 QueryRow 方法进行查询。

除了这里提到的特性之外，Sqlx 还拥有其他一些有趣的特性，感兴趣的读者可以通过访问 Sqlx 的 GitHub 页面来了解：https://github.com/jmoiron/sqlx。

Sqlx 是一个有趣并且有用的 database/sql 扩展，但它支持的特性并不多。与此相反，我们接下来要学习的 Gorm 库不仅把 database/sql 包隐藏了起来，它还提供了一个完整且强大的 ORM 机制来代替 database/sql 包。

6.5.2 Gorm

Gorm 的开发者声称 Gorm 是最棒的 Go 语言 ORM，他们的确所言非虚。Gorm 是“Go-ORM”一词的缩写，这个项目是一个使用 Go 实现的 ORM，它遵循的是与 Ruby 的 ActiveRecord 以及 Java 的 Hibernate 一样的道路。更确切地说，Gorm 遵循的是数据映射器模式（Data-Mapper pattern），该模式通过提供映射器来将数据库中的数据映射为结构。（在 6.3 节介绍关系数据库时，使用的就是 ActiveRecord 模式。）

Gorm 的能力非常强大，它允许程序员定义关系、实施数据迁移、串联多个查询以及执行其他很多高级的操作。除此之外，Gorm 还能够设置回调函数，这些函数可以在特定的数据事件发生时执行。因为详尽地描述 Gorm 的各个特性可能会花掉整整一章的篇幅，所以我们在这里只会讨论它的基本特性。代码清单 6-18 展示了使用 Gorm 重新实现论坛程序的方法，跟之前一样，这次的代码也是存储在 store.go 文件里面。

代码清单 6-18 使用 Gorm 实现论坛程序

```
package main

import (
    "fmt"
    "github.com/jinzhu/gorm"
    _ "github.com/lib/pq"
    "time"
)

type Post struct {
    Id       int
    Content   string
    Author    string `sql:"not null"`
    Comments  []Comment
    CreatedAt time.Time
}

type Comment struct {
    Id       int
```

```
    Content   string
    Author    string `sql:"not null"`
    PostId    int    `sql:"index"`
    CreatedAt time.Time
}

var Db gorm.DB

func init() {
    var err error
    Db, err = gorm.Open("postgres", "user=gwp dbname=gwp password=gwp
    ➥sslmode=disable")
    if err != nil {
        panic(err)
    }
    Db.AutoMigrate(&Post{}, &Comment{})
}

func main() {
    post := Post{Content: "Hello World!", Author: "Sau Sheong"}
    fmt.Println(post)

    Db.Create(&post)
    fmt.Println(post)

    comment := Comment{Content: "Good post!", Author: "Joe"}
    Db.Model(&post).Association("Comments").Append(comment)

    var readPost Post
    Db.Where("author = $1", "Sau Sheong").First(&readPost)
      var comments []Comment
      Db.Model(&readPost).Related(&comments)
fmt.Println(comments[0])
}
```

{0 Hello World! Sau Sheong []
0001-01-01 00:00:00 +0000 UTC}

创建一篇帖子

{1 Hello World! Sau Sheong []
2015-04-12 11:38:50.91815604 +0800 SGT}

添加一条评论

通过帖子获取评论

{1 Good post! Joe 1 2015-04-13
11:38:50.920377 +0800 SGT}

这个新程序创建数据库句柄的方法跟我们之前创建数据库句柄的方法基本相同。另外需要注意的一点是，因为 Gorm 可以通过自动数据迁移特性来创建所需的数据库表，并在用户修改相应的结构时自动对数据库表进行更新，所以这个程序无需使用 `setup.sql` 文件来设置数据库表：当我们运行这个程序时，程序所需的数据库表就会自动生成。为了正确地运行这个程序，并让程序能够正常地创建数据库表，我们在执行这个程序之前必须先将之前创建的数据库表全部删除：

```
func init() {
    var err error
    Db, err = gorm.Open("postgres", "user=gwp dbname=gwp password=gwp sslmode=disable")
    if err != nil {
        panic(err)
    }
    Db.AutoMigrate(&Post{}, &Comment{})
}
```

负责执行数据迁移操作的 `AutoMigrate` 方法是一个变长参数方法，这种类型的方法和函

数能够接受一个或多个参数作为输入。在上面展示的代码中，AutoMigrate 方法接受的是 Post 结构和 Comment 结构。得益于自动数据迁移特性的存在，当用户向结构里面添加新字段的时候，Gorm 就会自动在数据库表里面添加相应的新列。

上面的 Gorm 程序使用了下面所示的 Comment 结构：

```
type Comment  struct {
    Id        int
    Content   string
    Author    string `sql:"not null"`
    PostId    int
    CreatedAt time.Time
}
```

Comment 结构里面出现了一个类型为 time.Time 的 CreatedAt 字段，包含这样一个字段意味着 Gorm 每次在数据库里创建一条新记录的时候，都会自动对这个字段进行设置。

此外，Comment 结构的其中一些字段还用到了结构标签，以此来指示 Gorm 应该如何创建和映射相应的字段。比如，Comment 结构的 Author 字段就使用了结构标签`sql: "not null"`，以此来告知 Gorm，该字段对应列的值不能为 null。

跟前面展示过的程序的另一个不同之处在于，这个程序没有在 Comment 结构里设置 Post 字段，而是设置了一个 PostId 字段。Gorm 会自动把这种格式的字段看作是外键，并创建所需的关系。

在了解了 Post 结构和 Comment 结构的新定义之后，现在，让我们来看看程序是如何创建并获取帖子及其评论的。首先，程序会使用以下语句来创建新的帖子：

```
post := Post{Content: "Hello World!", Author: "Sau Sheong"}
Db.Create(&post)
```

这段代码没有什么难懂的地方，它跟之前展示过的代码的最主要区别在于——程序这次遵循了数据映射器模式：它在创建帖子时会使用数据库句柄 gorm.DB 作为构造器，而不是像之前遵循 ActiveRecord 模式时那样，通过直接调用 Post 结构自有的 Create 方法来创建帖子。

如果直接查看数据库内部，应该会看到 created_at 这个时间戳列在帖子创建出来的同时已经自动被设置好了。

在创建出帖子之后，程序使用了以下语句来为帖子添加评论：

```
comment := Comment{Content: "Good post!", Author: "Joe"}
Db.Model(&post).Association("Comments").Append(comment)
```

这段代码会先创建出一条评论，然后通过串联 Model 方法、Association 方法和 Append 方法来将评论添加到帖子里面。注意，在创建评论的过程中，我们无需手动对 Comment 结构的 PostId 字段执行任何操作。

最后，程序使用了以下代码来获取帖子及其评论：

```
var readPost Post
Db.Where("author = $1", "Sau Sheong").First(&readPost)
var comments []Comment
Db.Model(&readPost).Related(&comments)
```

这段代码跟之前展示过的代码有些类似，它使用了 gorm.DB 的 Where 方法来查找第一条作者名为"Sau Sheong"的记录，并将这条记录存储在了 readPost 变量里面，而这条记录就是我们刚刚创建的帖子。之后，程序首先调用 Model 方法获取帖子的模型，接着调用 Related 方法获取帖子的评论，并在最后将这些评论存储到 comments 变量里面。

正如之前所说，本节展示的特性只是 Gorm 这个 ORM 库众多特性的一小部分，如果你对这个库感兴趣，可以通过 https://github.com/jinzhu/gorm 了解更多相关信息。

Gorm 并不是 Go 语言唯一的 ORM 库。除 Gorm 之外，Go 还拥有不少同样具备众多特性的 ORM 库，比如，Beego 的 ORM 库以及 GORP（GORP 并不完全是一个 ORM，但它与 ORM 相去不远）。

在本章中，我们了解了构建 Web 应用所需的基本组件，而在接下来的一章中，我们将要开始讨论如何构建 Web 服务。

6.6 小结

- 通过使用结构将数据存储在内存里面，以此来构建数据缓存机制并提高响应速度。
- 通过使用 CSV 或者 gob 二进制格式将数据存储在文件里面，可以对用户提交的文件进行处理，或者为缓存数据提供备份。
- 通过使用 database/sql 包，可以对关系数据库执行 CRUD 操作，并在不同的数据之间建立起相应的关系。
- 通过 Sqlx 和 Gorm 这样的第三方数据访问库，可以使用威力更强大的工具去操纵数据库中的数据。

第三部分

实战演练

在上一个部分，我们学习了如何编写基本的服务器端 Web 应用，但这些知识只不过是 Web 应用开发中的沧海一粟。绝大多数现代化的 Web 应用早已超越了简单的请求-响应模型，并以多种不同的形式在不断地演进当中。比如，单页应用（Single Page Application，SPA）和移动应用（无论是原生的还是混合的）就能够在获取 Web 服务中的数据的同时，快速地与用户进行交互。

在本书的最后一部分，我们将会学习如何使用 Go 语言编写能够为单页应用、移动应用以及其他 Web 应用提供服务的 Web 服务。除此之外，我们还会深入了解 Go 语言强大的并发特性，并学习如何通过并发提高 Web 应用的性能。之后，我们会了解 Go 提供的几个测试工具，并使用这些工具对 Web 应用进行测试。

在本书的最后，我们将会学习如何以多种不同的方式部署 Web 应用，其中包括只需要将可执行二进制文件复制到目标服务器的简单部署方法，以及需要执行一系列步骤才能将 Web 应用推送到云端的高级部署方法。

第 7 章　Go Web 服务

本章主要内容

- 使用 REST 风格的 Web 服务
- 使用 Go 创建和分析 XML
- 使用 Go 创建和分析 JSON
- 编写 Go Web 服务

正如本书第 1 章所言，Web 服务就是一个向其他软件程序提供服务的程序。本章将扩展这一定义，并展示如何使用 Go 语言来编写或使用 Web 服务。因为 XML 和 JSON 是 Web 服务最常使用的数据格式，所以我们首先会学习如何创建以及分析这两种数据格式，接着我们将会讨论 SOAP 风格的服务以及 REST 风格的服务，并在之后学习如何创建一个使用 JSON 传输数据的简单的 Web 服务。

7.1 Web 服务简介

通过 Go 语言编写的 Web 服务向其他 Web 服务或应用提供服务和数据，是 Go 语言的一种常见的用法。所谓的 Web 服务，一言以蔽之，就是一种与其他软件程序进行交互的软件程序。这也就是说，Web 服务的终端用户（end user）不是人类，而是软件程序。正如“Web 服务”这一名字所暗示的那样，这种软件程序是通过 HTTP 进行通信的，如图 7-1 所示。

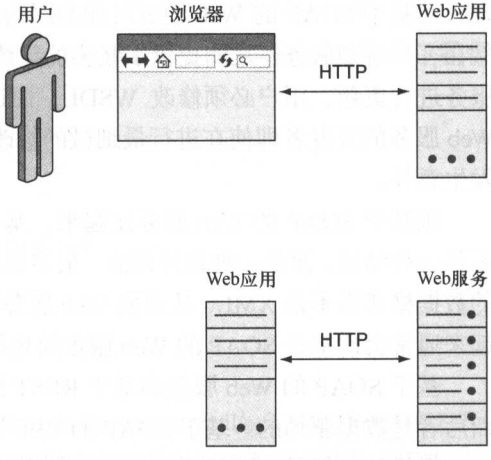

图 7-1　Web 应用与 Web 服务的不同之处

有趣的是，虽然 Web 应用并没有一个确切的定义，但 Web 服务的定义却可以在 W3C 工作组发布的《Web 服务架构》（Web Service Architecture）文

档中找到：

> Web 服务是一个软件系统，它的目的是为网络上进行的可互操作机器间交互（interoperable machine-to-machine interaction）提供支持。每个 Web 服务都拥有一套自己的接口，这些接口由一种名为 Web 服务描述语言（web service description language，WSDL）的机器可处理格式描述。其他系统需要根据 Web 服务的描述，使用 SOAP 消息与 Web 服务交互。为了与其他 Web 相关标准实现协作，SOAP 消息通常会被序列化为 XML 并通过 HTTP 传输。
>
> ——《Web 服务架构》，2004 年 2 月 11 日

从这一定义来看，似乎所有 Web 服务都应该基于 SOAP 来实现，但实际中却存在着多种不同类型的 Web 服务，其中包括基于 SOAP 的、基于 REST 的以及基于 XML-RPC 的，而基于 REST 的和基于 SOAP 的 Web 服务又是其中最为流行的。企业级系统大多数都是基于 SOAP 的 Web 服务实现的，而公开可访问的 Web 服务则更青睐基于 REST 的 Web 服务，本章稍后将会对此进行讨论。

基于 SOAP 的 Web 服务和基于 REST 的 Web 服务都能够完成相同的功能，但它们各自也拥有不同的长处。基于 SOAP 的 Web 服务出现的时间较早，W3C 工作组已经对其进行了标准化，与之相关的文档和资料也非常丰富。除此之外，很多企业都对基于 SOAP 的 Web 服务提供了强有力的支持，并且基于 SOAP 的 Web 服务还拥有数量颇丰的扩展可用（因为这些扩展的名字绝大多数都是像 WS-Security 和 WS-Addressing 这样以 WS 为前缀的，所以这些扩展被统称为 WS-*）。基于 SOAP 的服务不仅健壮、能够使用 WSDL 进行明确的描述、拥有内置的错误处理机制，而且还可以通过 UUDI（Universal Description, Discovery, and Integration，统一描述、发现和集成）（一种目录服务）规范发布。

在拥有以上众多优点的同时，SOAP 的缺点也是非常明显的：它不仅笨重，而且过于复杂。SOAP 的 XML 报文可能会变得非常冗长，导致难以调试，使用户只能通过其他工具对其进行管理，而基于 SOAP 的 Web 服务可能会因为额外的资源损耗而无法高效地运行。此外，WSDL 虽然在客户端和服务器之间提供了坚实的契约，但这种契约有时候也会变成一种累赘：为了对 Web 服务进行更新，用户必须修改 WSDL，而这种修改又会引起 SOAP 客户端发生变化，最终导致 Web 服务的开发者即使在进行最细微的修改时，也不得不使用版本锁定（version lock-in）以防止发生意外。

跟基于 SOAP 的 Web 服务比起来，基于 REST 的 Web 服务就显得灵活多了。REST 本身并不是一种结构，而是一种设计理念。很多基于 REST 的 Web 服务都会使用像 JSON 这样较为简单的数据格式而不是 XML，从而使 Web 服务可以更高效地运行，并且基于 REST 的 Web 服务实现起来通常会比基于 SOAP 的 Web 服务简单得多。

基于 SOAP 的 Web 服务和基于 REST 的 Web 服务的另一个区别在于，前者是功能驱动的，而后者是数据驱动的。基于 SOAP 的 Web 服务往往是 RPC（Remote Procedure Call，远程过程调用）风格的；但是，正如之前所说，基于 REST 的 Web 服务关注的是资源，而 HTTP 方法则是

对这些资源执行操作的动词。

ProgrammableWeb 是一个流行的 API 检测网站，它会对互联网上公开可用的 API 进行检测。在编写本书的时候，ProgrammableWeb 的数据库搜集了 12 987 个公开可用的 API，其中 2 061 个（占比 16%）为基于 SOAP 的 API，而 6 967 个（占比 54%）为基于 REST 的 API[①]。可惜的是，因为企业很少会对外发布与内部 Web 服务有关的信息，所以想要调查清楚各种 Web 服务在企业中的使用情况是非常困难的。

为了满足不同的需求，很多开发者和公司最终还是会同时使用基于 SOAP 的 Web 服务和基于 REST 的 Web 服务。在这种情况下，SOAP 将用于实现内部应用的企业集成（enterprise integration），而 REST 则用于服务外部以及第三方的开发者。这一策略的优势在于，它最大限度地利用了 REST（速度快并且构建简单）以及 SOAP（安全并且健壮）这两种技术的优点。

7.2 基于 SOAP 的 Web 服务简介

SOAP 是一种协议，用于交换定义在 XML 里面的结构化数据，它能够跨越不同的网络协议并在不同的编程模型中使用。SOAP 原本是 Simple Object Access Protocol（简单对象访问协议）的首字母缩写，但这实际上是一个名不符实的名字，因为这种协议处理的并不是对象，并且时至今日它也已经不再是一种简单的协议了。在最新版的 SOAP 1.2 规范中，这种协议的官方名称仍然为 SOAP，但它已经不再代表 Simple Object Access Protocol 了。

因为 SOAP 不仅高度结构化，而且还需要严格地进行定义，所以用于传输数据的 XML 可能会变得非常复杂。WSDL 是客户端与服务器之间的契约，它定义了服务提供的功能以及提供这些功能的方式，服务的每个操作以及输入/输出都需要由 WSDL 明确地定义。

虽然本章主要关注的是基于 REST 的 Web 服务，但出于对比需要，我们也会了解一下基于 SOAP 的 Web 服务的运作机制。

SOAP 会将它的报文内容放入到信封（envelope）里面，信封相当于一个运输容器，并且它还能够独立于实际的数据传输方式存在。因为本书只会对 SOAP Web 服务进行考察，所以我们将通过 HTTP 协议来说明被传输的 SOAP 报文。

下面是一个经过简化的 SOAP 请求报文示例：

```
POST /GetComment HTTP/1.1
Host: www.chitchat.com
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

① SOAP API 的搜集结果可以通过访问 www.programmableweb.com/category/all/apis?data_format=21176 查看，而 REST API 的搜集结果可以通过访问 www.programmableweb.com/category/all/apis?data_format= 21190 查看。