

```

getLogger(ApiApp.class);

static {
    //仅供本地主机测试
    LOG.warn("现在将禁用 SSL中的主机名检查，只在开发期间使用");
    HttpsURLConnection.setDefaultHostnameVerifier((hostname,
    sslSession) -> true);
}

@Value("${app.rabbitmq.host:localhost}")
String rabbitMqHost;

@Bean
public ConnectionFactory connectionFactory() {
    LOG.info("Create RabbitMqCF for host: {}", rabbitMqHost);
    CachingConnectionFactory connectionFactory = new CachingCo
nnectionFactory(rabbitMqHost);
    return connectionFactory;
}

public static void main(String[] args) {
    LOG.info("Register MDCHystrixConcurrencyStrategy");
    HystrixPlugins.getInstance().
registerConcurrencyStrategy(new MDCHystrixConcurrencyStrategy());
    SpringApplication.run(ApiApp.class, args);
}
}

```

3. 在 application.yml 中更新 api-service 配置，如下面的代码所示：

- security.oauth2.resource.userInfoUri：它表示安全服务用户URI。

```

application.yml
info:
  component: API Service

spring:
  application:
    name: api-service

```

```
aop:  
    proxyTargetClass: true  
  
server:  
    port: 7771  
  
security:  
    oauth2:  
        resource:  
            userInfoUri: https://localhost:9001/auth/user  
  
management:  
    security:  
        enabled: false  
## 其他属性，比如Eureka、Logging等
```

现在我们有了安全服务器，我们会使用新的微服务 api-service 公开 API，这将用于与外部应用程序和 UI 进行通信。

现在让我们测试和探索它如何为不同的 OAuth 2.0 许可类型工作。

[ 我们会使用 Chrome 浏览器的 postman 扩展来测试不同的流程。]

授权码许可

我们将在浏览器中输入以下 URL，对授权码的一个请求如下所示：

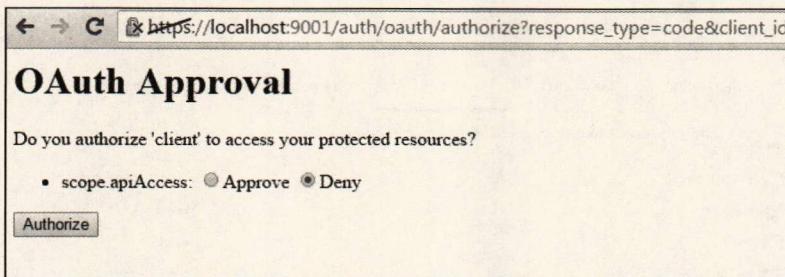
```
https://localhost:9001/auth/oauth/authorize?response_type=code&client_id=client&redirect_uri=http://localhost:7771/1&scope=apiAccess&state=1234
```

在这里，我们提供客户端 ID（硬编码 client 是在默认情况下，我们在安全服务中注册的），重定向 URI、范围（在安全服务中的硬编码值 apiAccess）和状态。你一定想知道 state 参数的含义，它包含我们在响应中重新验证的随机数字，以防止跨站点请求伪造。

如果资源所有者（用户）尚未通过身份验证，它会要求输入用户名和密码。提供 username

作为用户名，password 作为密码，我们已经在安全服务中硬编码了这些值。

一旦登录成功，它将要求你（资源所有者）提供审批。

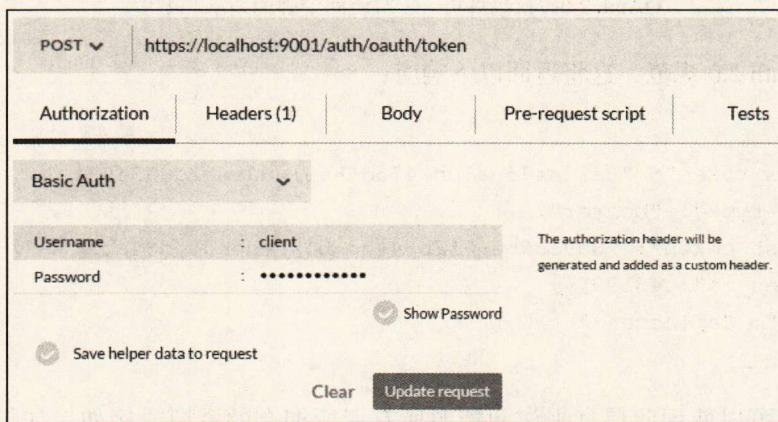


OAuth 2.0授权码许可——资源许可的审批

选中 **Approve**（审批）并在 **Authorize**（授权）上点击。此操作将把应用程序重定向到 `http://localhost:7771/1?code=o8t4fi&state=1234`。

正如你所看到的，它返回授权码和状态。

现在，我们将使用此代码来获取访问代码，将使用 Chrome 扩展 postman。首先，我们将使用用户名 client 和密码 clientsecret 添加授权标头，如下面的屏幕截图所示。



OAuth 2.0授权码许可 - 访问令牌请求 - 添加身份验证

这会将 Authorization 标头添加到请求中，值是 Basic Y2xpZW50OmNsawVudHN1Y3JldA==。

现在，我们会把其他几个参数添加到请求中，如下面的屏幕截图所示，然后提交该请求。

Key	Value
grant_type	authorization_code
client_id	client
code	o8t4fi
redirect_uri	http://localhost:7771/

```

1 <[{"access_token": "6a233475-a5db-476d-8e31-d0aeb2d003e9", "token_type": "bearer", "refresh_token": "8d91b9be-7f2b-44d5-b14b-dbbcd848b8", "expires_in": 43199, "scope": "apiAccess"}]>
    
```

OAuth 2.0授权码许可——对访问令牌的请求和响应

根据 OAuth 2.0 规范，这将返回以下响应：

```
{
  "access_token": "6a233475-a5db-476d-8e31-d0aeb2d003e9",
  "token_type": "bearer",
  "refresh_token": "8d91b9be-7f2b-44d5-b14b-dbbcd848b8",
  "expires_in": 43199,
  "scope": "apiAccess"
}
```

现在我们可以使用此信息来访问资源所有者所拥有的资源。例如，如果 <https://localhost:8765/api/restaurant/1> 表示 ID 为 1 的餐馆，那么它应该返回各餐馆的详细信息。

若没有访问令牌，如果我们输入这个 URL，它将返回错误 Unauthorized（未经授权）

权), 以及 Full authentication is required to access this resource (访问该资源需要完整的身份验证) 的消息。

现在, 让我们使用访问令牌访问这个 URL, 如下面的屏幕截图所示。

```

GET https://localhost:8765/api/restaurant/1
Authorization: Bearer 6a233475-a5db-476d-8e31-d0aeb2d003e9
Content-Type: application/json
Content-Length: 144
Date: Mon, 10 Dec 2018 14:45:10 GMT
Server: Apache/2.4.29 (Ubuntu)
Set-Cookie: JSESSIONID=6a233475-a5db-476d-8e31-d0aeb2d003e9; Path=/; HttpOnly
Status: 200 OK
Time: 200ms

{
  "tables": null,
  "id": "1",
  "isModified": false,
  "name": "Big-O Restaurant"
}
  
```

OAuth 2.0授权码许可——使用访问令牌进行API访问

正如你所看到的, 我们在 **Authorization** 标头中加入了访问令牌。

现在, 我们将研究隐式许可的实现。

隐式许可

除了代码许可这一步外, 隐式许可与授权码许可非常类似。如果你从授权码许可中删除第一步——代码许可这一步 (在这里, 客户端应用程序从授权服务器接收授权令牌), 剩下的步骤是一样的。我们来了解一下。

在浏览器中输入以下 URL 和参数, 然后按 Enter 键。此外, 请确保添加基本身份验证, 如果被要求提供, 则以 `client` 作为用户名, `password` 作为密码:

```

https://localhost:9001/auth/oauth/authorize?response_type=token&redirect_uri=https://localhost:8765&scope=apiAccess&state=553344&client_id=client
  
```

在这里, 我们调用授权端点时使用以下的请求参数: 响应类型、客户端 ID、重定向 URI、范围和状态。

当请求成功时，浏览器将被重定向到下面的 URL，并包含新的请求参数和值：

```
https://localhost:8765/#access_token=6a233475-a5db-476d-8e31-d0aeb2d003e9&token_type=bearer&state=553344&expires_in=19592
```

在这里，我们收到令牌的 `access_token`、`token_type`、状态和到期期限。现在，我们可以使用这个访问令牌来访问 API，如同在授权码许可部分使用的那样。

资源所有者密码凭据许可

在这个许可中，我们在请求访问令牌时提供 `username` 和 `password` 作为参数，以及 `grant_type`、`client_id` 和 `scope` 参数。我们还需要使用客户端 ID 和密码来对请求进行身份验证。这些许可流程使用客户端应用程序代替浏览器，并且通常用于移动和桌面应用程序。

在下面的 postman 工具屏幕截图中，已使用基本身份验证和 `client_id` 及 `password` 添加了授权标头。

Key	Value
grant_type	password
scope	apiAccess
client_id	client
username	user
password	password

```

1  [
2   "access_token": "6a233475-a5db-476d-8e31-d0aeb2d003e9",
3   "token_type": "bearer",
4   "refresh_token": "8d91b9be-7f2b-44d5-b14b-dbbcd848b8",
5   "expires_in": 17377,
6   "scope": "apiAccess"
7 ]

```

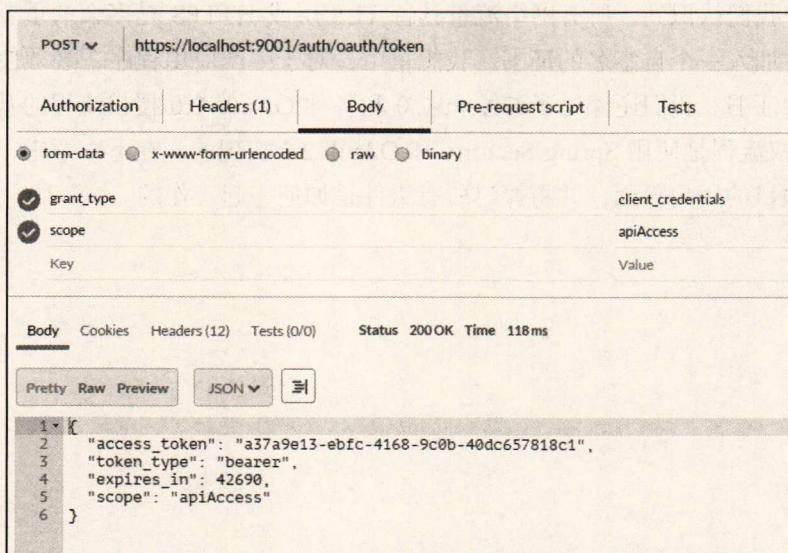
OAuth 2.0 资源所有者密码凭据许可——访问令牌的请求和响应

一旦客户端收到访问令牌，它就可以用类似于它在授权码许可中的用法来使用。

客户端凭据许可

在这种流程中，客户端提供自己的凭据并获取访问令牌。它不使用资源所有者的凭据和权限。

可以在下面的屏幕截图中看到，我们直接输入令牌端点，只包含两个参数：grant_type 和 scope。使用 client_id 和 client_secret 添加了授权标头。



OAuth 2.0客户端凭据许可——访问令牌的请求和响应

可以像授权码许可部分中解释的那样来使用访问令牌。

参考资料

更多的信息，请参阅以下链接：

- RESTful Java Web Services Security, Packt Publishing, René Enríquez, Andrés Salazar C 著: <https://www.packtpub.com/application-development/restful-java-web-services-security>

- *Spring Security [Video]*, Packt Publishing: <https://www.packtpub.com/application-development/spring-security-video>
- OAuth 2.0 授权框架: <https://tools.ietf.org/html/rfc6749>
- Spring Security: <http://projects.spring.io/spring-security>
- Spring OAuth2: <http://projects.spring.io/spring-security-oauth/>

小结

在本章中我们认识到，所有网络流量具备 TLS 层或 HTTPS 是多么的重要。我们在示例应用程序中加入一个自签名的证书。我想重申，对于生产应用程序，必须使用证书签发机构所提供的证书。我们还探讨了 OAuth 2.0 和各种 OAuth 2.0 授权流程的基础，不同的 OAuth 2.0 授权流程是使用 Spring Security 和 OAuth 2.0 实现的。在下一章中，我们将实现示例 OTRS 项目的用户界面，并将探讨所有组件是如何一起工作的。

7

利用微服务 Web 应用程序来使用服务

现在，开发了微服务之后，我们想看看如何可以通过 web 或移动应用程序使用在线餐馆订位系统(**OTRS**)所提供的服务。我们会使用 AngularJS/bootstrap 建立 web 应用程序(UI)来构建 web 应用程序的原型。示例应用程序将显示此示例项目——一个实用项目的数据和流程。此 web 应用程序也将是一个示例项目，并将独立运行。早期，web 应用程序被开发为单个 web 归档文件(.war 扩展名的文件)，它同时包含用户界面和服务器端代码。这样做的原因是开发过程相当简单，因为 UI 也使用 Java，包括 JSP、servlet、JSF 来开发。如今，UI 正在使用 JavaScript 独立开发。因此，这些 UI 应用程序也将部署为单独的微服务。在这一章，我们将探索如何开发这些独立的 UI 应用程序。我们将开发和实现没有登录和授权流程的 OTRS 示例应用程序。我们会部署一个功能非常有限的实现并涵盖高层次的 AngularJS 概念。要了解 AngularJS 的详细信息，你可以参考 Packt 出版的由 Chandermani 编写的《AngularJS by Example》一书。

在这一章，我们将介绍以下主题：

- AngularJS 框架概述
- 开发 OTRS 的功能
- 建立一个 web 应用程序(UI)

AngularJS 框架概述

现在既然我们准备好了 HTML5 web 应用程序的设置，可以浏览一下 AngularJS 的基本知识，这将帮助我们理解 AngularJS 代码。本节描述了对 AngularJS 高层次的理解，可以利

用它来了解示例应用程序，并利用 AngularJS 文档或通过参考其他 Packt 出版物来做进一步研究。

AngularJS 是一个客户端 JavaScript 框架。它有足够的灵活性来用作 **MVC**(**Model View Controller** 模型视图控制器) 或 **MVVM** (**Model-View-ViewModel** 模型-视图-视图模型)，它还使用依赖项注入模式提供内置的服务，如\$http 或\$log。

MVC

MVC 是知名的设计模式。Struts 和 Spring MVC 是流行的例子，让我们看看它们是如何适合 JavaScript 环境的：

- **模型**：模型是 JavaScript 对象，其中包含应用程序数据。它们也表示应用程序的状态。
- **视图**：视图是表示层，其中包含 HTML 文件。在这里，可以显示来自模型的数据，并向用户提供交互式界面。
- **控制器**：可以在 JavaScript 中定义控制器，它包含应用程序逻辑。

MVVM

MVVM 是一种专门针对 UI 开发的架构设计模式。MVVM 被设计为使得双向数据绑定更容易。双向数据绑定提供了模型和视图之间的同步，当模型（数据）更改时，它会立即反映在视图上。同样，当用户更改视图上的数据时，它也反映在模型上。

- **模型**：这非常类似于 MVC，并包含业务逻辑和数据。
- **视图**：像 MVC 一样，它包含表示逻辑或用户界面。
- **视图模型**：视图模型包含在视图和模型之间绑定的数据。因此，它是视图和模型之间的一个接口。

模块

对于任何 AngularJS 应用程序，模块都是我们定义的第一个事物。模块是一个容器，它包含控制器、服务、过滤器等应用程序的不同部分。AngularJS 应用程序可以在单个模块或多个模块中编写，AngularJS 模块也可以包含其他的模块。

许多其他 JavaScript 框架都使用 main 方法来实例化和联系应用程序的不同部分。AngularJS 并没有 main 方法。由于下列原因，它使用模块作为入口点：

- **模块化：**可以按照功能或可重用的组件来划分和创建应用程序。
- **简单：**你可能遇到过复杂和规模庞大的应用程序代码，它们使维护和扩展非常困难。更多的好处是，AngularJS 使代码简单、可读性好并容易理解。
- **测试：**它使单元测试和端到端测试更加容易，因为你可以重写配置并只装载所需的模块。

每个 AngularJS 的应用程序都需要有单个模块来引导我们的应用程序。AngularJS 应用程序需要以下三个部分：

- **应用程序模块：**包含 AngularJS 模块的一个 JavaScript 文件（app.js），如下所示。

```
var otrsApp = AngularJS.module('otrsApp', [ ])
//[] 包含对其他模块的引用
```

- **加载 Angular 库和应用模块：**包含对 JavaScript 文件与其他 AngularJS 库的引用的 index.html 文件。

```
<script type="text/javascript" src="AngularJS/AngularJS.js"/>
<script type="text/javascript" src="scripts/app.js"/></script>
```

- **应用程序 DOM 配置：**这告诉 AngularJS 应该发生启动的 DOM 元素的位置。它可以用两种方式完成：

- Index.html 文件，它还包含 HTML 元素（通常是<html>）与 app.js 中给定的值的 ng- app（AngularJS 指令）属性。AngularJS 指令包含 ng 前缀（AngularJS）：`<html lang="en" ng-app="otrsApp" class="no-js">`。
- 如果你异步加载 JavaScript 文件，使用此命令：AngularJS.bootstrap（document.documentElement, ['otrsApp']);。

AngularJS 模块除了其他组件，如控制器、服务、过滤器，等等，还有两个重要组成部分，config() 和 run()。

- `config()` 用于注册和配置模块，并且它只用于提供者和使用`$injector` 的常量。`$injector` 是一个 AngularJS 服务，我们将在下一节介绍提供程序和`$injector`。在这里不能使用实例，它在完全配置之前，会阻止使用服务。
- `run()` 是用于在使用前面的配置方法创建`$injector` 后执行代码。这只供实例和常量使用。此处的提供程序不能用于避免在运行时配置。

提供程序和服务

让我们看看下面的代码：

```
.controller('otrsAppCtrl', function ($injector) {  
    var log = $injector.get('$log');
```

`$log` 是内置的 AngularJS 服务，它提供了日志记录 API。在这里，我们使用另一个内置的服务，`$injector`，它允许我们使用`$log` 服务。`$injector` 是控制器中的参数。AngularJS 用函数定义和正则表达式向调用方，也称控制器提供`$injector` 服务。这些都是 AngularJS 如何有效地使用依赖注入模式的例子。

AngularJS 使用了大量的依赖注入模式。AngularJS 使用注入器服务（`$injector`）来实例化和布线我们在 AngularJS 应用程序中使用的大部分对象。这个注入器创建两种类型的对象——服务和专门的对象。

为了简化起见，可以说我们（开发人员）定义服务。与此相反，专门的对象是控制器、过滤器、指令之类的 AngularJS 工件。

AngularJS 提供五种方法类型来告诉注入器如何创建服务对象——提供程序、值、工厂、服务和常量。

- 提供程序是核心和最复杂的方法类型，其他的方法都是在它上面添加的合成糖衣。我们通常避免使用提供程序，除非我们需要创建需要全局配置的可重用代码。
- 值和常量的方法类型的工作方式恰如其名，两者都不能有依赖关系。此外，两者的区别在于它们的用法，在配置阶段，不能使用值服务对象。
- 工厂和服务是使用最多的服务类型，它们是相似的类型。当我们想要生成 JavaScript 基元和函数时，我们使用工厂食谱。另一方面，当我们想要生成自定义的类型时，将使用服务。

现在我们对服务有了一些了解，可以说，服务有两种常见的用法——组织代码和在应用程序之间共享代码。服务是单例对象，它由 AngularJS 服务工厂懒惰地实例化。到目前为止，我们已经看到一些内建的 AngularJS 服务，如 \$injector、\$log，等等。AngularJS 服务均以 \$ 符号作为前缀。

作用域

AngularJS 应用程序中有两种广泛使用的作用域类型：\$rootScope 和\$scope：

- \$rootScope 是作用域层次结构中顶层的对象，它与全局的作用域关联。这意味着，附加到它上面的任何变量都能在任何地方使用，因此使用 \$rootScope 应当是经过深思熟虑的决定。
- 控制器把 \$scope 作为回调函数中的一个参数，它用于将数据从控制器绑定到视图，其作用域仅限于与其关联的控制器使用。

控制器

当控制器有一个 \$scope 作为参数，它由 JavaScript constructor 函数定义。控制器的主要目的是把数据结合到视图。控制器函数也用于编写业务逻辑——设置 \$scope 对象的初始状态并将行为添加到 \$scope。控制器的签名如下所示：

```
RestModule.controller('RestaurantsCtrl', function ($scope,
  restaurantService) {
```

在这里，控制器是 RestModule 的一部分。控制器的名称是 RestaurantsCtrl。\$scope 和 restaurantService 都作为参数传递。

过滤器

过滤器的用途是格式化一个给定的表达式的值。下面的代码中我们定义了 datetime1 过滤器，它将日期作为参数并把值更改为 dd MMM yyyy HH:mm 的格式，如 04 Apr 2016 04:13 PM。

```
.filter('datetime1', function ($filter) {
  return function (argDateTime) {
    if (argDateTime) {
```

```

        return $filter('date')(new Date(argDateTime), 'dd MMM yyyy HH:mm a');
    }
    return "";
};

});

```

指令

我们已经在模块一节看到，AngularJS 指令是带有 ng 前缀的 HTML 属性。下面是一些流行的指令：

- **ng-app**: 此指令定义 AngularJS 应用程序
- **ng-model**: 此指令将 HTML 表单输入绑定到数据
- **ng-bind**: 此指令将数据绑定到 HTML 视图
- **ng-submit**: 此指令提交 HTML 表单
- **ng-repeat**: 此指令遍历集合

```

<div ng-app="">
  <p>Search: <input type="text" ng-model="searchValue"></p>
  <p ng-bind="searchedTerm"></p>
</div>

```

UI-Router

在单页面应用程序(**single page applications, SPAs**)中，页面只加载一次，用户通过不同的链接导航，而不刷新页面。这都是因为路由才有可能。路由是能使 SPA 导航感觉像普通网站的一种方法。因此，路由对于 SPA 是非常重要的。

AngularUI 团队构建了 UI-Router，它是一种 AngularJS 路由框架，不是核心 AngularJS 的一部分。当用户点击 SPA 中的任何链接时，UI-Router 不仅改变路由 URL，它还更改应用程序的状态。因为 UI-Router 还可以进行状态更改，可以在不更改 URL 的情况下更改页面的视图。由于应用程序状态由 UI-Router 管理，而使这变得可能。

如果我们把 SPA 当作一个状态机，那么其状态是应用程序的当前状态。当我们创建路由链接时，将在 HTML 链接标签使用 **ui-sref** 属性。链接中的属性 **href** 将从这生成并

指向在 `app.js` 中创建的应用程序的特定状态。

我们在 HTML `div` 中使用 `ui-view` 属性来使用 UI-Router: 例如, `<div ui-view></div>`。

OTRS 功能的开发

正如你所知, 我们正在开发 SPA。因此, 一旦应用程序加载, 你就可以执行所有操作而无须刷新页面。与服务器的所有交互都是使用 AJAX 调用执行的。现在, 我们会使用第一节介绍的 AngularJS 的概念, 会包括以下场景:

- 一个将显示餐馆列表的页面, 这也将是我们的主页。
- 搜索餐馆。
- 餐馆详细信息和预订选项。
- 登录 (不登录到服务器上, 而是用于显示流程)。
- 预订确认。

对于主页, 我们将创建 `index.html` 和一个模板, 它将包含在中间部分或内容区域中列出的餐馆列表。

主页/餐馆列表页

主页是任何 web 应用程序的主页面。为了设计主页, 我们打算使用 Angular-UI 引导程序, 而不是实际的引导程序。Angular-UI 是引导程序的 Angular 版本。主页将分为三个部分:

- 页眉部分将包含应用程序名称、搜索餐馆表单, 以及在右上角的用户名。
- 内容或中间部分将包含餐馆清单, 它将用餐馆名称作为链接。此链接将指向餐馆详细信息和预订页面。
- 页脚部分将包含应用程序名称与版权标记。

你一定有兴趣在设计或实现主页之前查看它。因此, 让我们首先来看看, 一旦我们准备好了内容, 它看起来的样子。

The screenshot shows a web application titled "Online Table Reservation System". At the top right, it says "Welcome Guest!". Below the title, there is a search bar with the placeholder "Search Restaurants" and a "Go" button. The main content area is titled "Famous Gourmet Restaurants in Paris" and displays a table with 10 rows of restaurant information. The columns are labeled "#Id", "Name", and "Address". The data is as follows:

#Id	Name	Address
1	Le Meurice	228 rue de Rivoli, 75001, Paris
2	L'Ambroisie	9 place des Vosges, 75004, Paris
3	Arpège	84, rue de Varenne, 75007, Paris
4	Alain Ducasse au Plaza Athénée	25 avenue de Montaigne, 75008, Paris
5	Pavillon LeDoyen	1, avenue Dutuit, 75008, Paris
6	Pierre Gagnaire	6, rue Balzac, 75008, Paris
7	L'Astrance	4, rue Beethoven, 75016, Paris
8	Pré Catelan	Bois de Boulogne, 75016, Paris
9	Guy Savoy	18 rue Troyon, 75017, Paris
10	Le Bristol	112, rue du Faubourg St Honoré, 8th arrondissement, Paris

© 2016 Online Table Reservation System

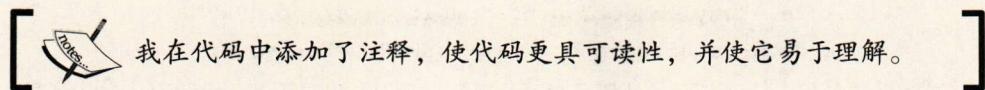
包含餐馆清单的OTRS主页

现在，来设计我们的主页，我们需要添加以下四个文件：

- `index.html`: 我们的主 HTML 文件
- `app.js`: 我们的主 AngularJS 模块
- `restaurants.js`: 餐馆模块，它还包含餐馆 Angular 服务
- `restaurants.html`: 将显示餐馆列表的 HTML 模板

`index.html`

首先，我们会把`./app/index.html`添加到我们的项目工作区。`Index.html`的内容将从这里开始描述。



`index.html` 被分成许多部分，我们将讨论一些关键的部件。首先，我们将看到如何处理 Internet Explorer 浏览器的旧版本。如果想要针对版本大于 8 或 IE 9 起的 Internet Explorer 浏览器，那么我们需要添加以下块，这会阻止 JavaScript 的呈现，并把不含 js 的输出呈现给最终用户。

```
<!--[if lt IE 7]>      <html lang="en" ng-app="otrsApp" class="no-js"
lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]>        <html lang="en" ng-app="otrsApp" class="no-js"
lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]>        <html lang="en" ng-app="otrsApp" class="no-js"
lt-ie9"> <![endif]-->
<!--[if gt IE 8]><!--> <html lang="en" ng-app="otrsApp" class="no-js">
<!--<![endif]-->
```

然后，在添加几个 `meta` 标记和应用程序的标题后，我们还会定义重要的 `meta` 标记视区 (**viewport**)。视区用于响应 UI 设计。

在内容属性中定义的 `width` 属性控制视区的大小。它可以设置为特定数量的像素宽度，如 `width = 600` 或特定的设备宽度值，即 100% 比例的 CSS 像素屏幕的宽度。

`initial-scale` 属性控制第一次加载网页时的缩放级别。`maximum-scale`、`minimum-scale` 和 `user-scalable` 属性控制允许用户如何放大或缩小页面。

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

在接下来的几行中，我们将定义应用程序的样式表。我们将添加 HTML5 样板代码中的 `normalize.css` 和 `main.css`，也加入应用程序的自定义 CSS `app.css`。最后，我们添加 `bootstrap 3` CSS。除了自定义的 `app.css`，其他 CSS 都被它引用。这些 CSS 文件都没有变化。

```
<link rel="stylesheet" href="bower_components/html5-boilerplate/dist/
css/normalize.css">
<link rel="stylesheet" href="bower_components/html5-
boilerplate/dist/css/main.css">
```

```
<link rel="stylesheet" href="public/css/app.css">
<link data-require="bootstrap-css@*" data-server="3.0.0"
rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/
bootstrap.min.css" />
```

然后我们将使用 `script` 标记定义脚本。我们正在添加 `modernizer`、`Angular`、`Angular-route` 和我们自己开发的自定义 `JavaScript` 文件 `app.js`。我们已经讨论过 `Angular` 和 `ngular-UI`, `app.js` 将在下一节讨论。

`modernizer` 使 `web` 开发人员可以使用新的 `CSS3` 和 `HTML5` 功能, 同时对并不支持它们的浏览器保持精细级别的控制权。基本上, `modernizer` 在页面加载到浏览器时执行下一代功能检测(检查这些功能的可用性), 并报告结果。基于这些结果, 可以检测浏览器中有哪些最新功能可用, 可以基于此向最终用户提供一个接口。如果浏览器不支持一些功能, 那么就把备用流程或 UI 提供给最终用户。

我们还使用 `ui-bootstrap-tpls` `javascript` 文件把用 `JavaScript` 写的引导程序模板添加进去。

```
<script src="bower_components/html5-boilerplate/dist/js/
vendor/modernizr-2.8.3.min.js"></script>
<script src="bower_components/angular/angular.min.js"></script>

<script src="bower_components/angular-route/angular-route.min.js">
</script>
<script src="app.js"></script>
<script data-require="ui-bootstrap@0.5.0" data-semver="0.5.0"
src="http://angular-ui.github.io/bootstrap/ui-bootstrap-tpls-
0.6.0.js"></script>
```

我们还可以向 `head` 标记添加样式, 如下所示。这个样式允许下拉菜单正常工作。

```
<style>
div.navbar-collapse.collapse {
    display: block;
    overflow: hidden;
    max-height: 0px;
    -webkit-transition: max-height .3s ease;
```

```

        -moz-transition: max-height .3s ease;
        -o-transition: max-height .3s ease;
        transition: max-height .3s ease;
    }
    div.navbar-collapse.collapse.in {
        max-height: 2000px;
    }
</style>

```

在 body 标记中我们使用 ng-controller 属性定义应用程序的控制器，在页面加载时，它把应用程序到 Angular 的名称告诉控制器。

```
<body ng-controller="otrsAppCtrl">
```

然后，我们定义主页的 header 部分。在 header 部分，我们将定义应用程序的标题，Online Table Reservation System。此外，我们还将定义用来搜索餐馆的搜索表单。

```

<!-- BEGIN HEADER -->
<nav class="navbar navbar-default" role="navigation">

    <div class="navbar-header">
        <a class="navbar-brand" href="#">
            Online Table Reservation System
        </a>
    </div>
    <div class="collapse navbar-collapse" ng-
class="!navCollapsed && 'in'" ng-click="navCollapsed = true">
        <form class="navbar-form navbar-left" role="search"
ng-submit="search()">
            <div class="form-group">
                <input type="text" id="searchedValue" ng-
model="searchedValue" class="form-control" placeholder="Search
Restaurants">
            </div>
            <button type="submit" class="btn btn-default" ng-
click="">Go</button>
        </form>
    <!-- END HEADER -->

```

然后，在下一部分，即中间部分，包括我们实际上绑定不同的视图，它使用实际的内容注释做标记。在 div 中的 ui-view 属性从 Angular 动态地获取其内容，如餐馆详细信息、餐馆列表，等等。我们还在中间部分添加了一个警告对话框和微调框，它们将在有必要时显示。

```
<div class="clearfix"></div>
<!-- 开始容器 -->
<div class="page-container container">
    <!-- 开始内容 -->
    <div class="page-content-wrapper">
        <div class="page-content">
            <!-- 开始实际内容 -->
            <div ui-view class="fade-in-up"></div>
            <!-- 结束实际内容 -->
        </div>
    </div>
    <!-- 结束内容 -->
</div>
<!-- 加载微调框 -->
<div id="loadingSpinnerId" ng-show="isSpinnerShown()"
style="top:0; left:45%; position:absolute; z-index:999">
    <script type="text/ng-template" id="alert.html">
        <div class="alert alert-warning" role="alert">
            <div ng-transclude></div>
        </div>
    </script>
    <uib-alert type="warning" template-url="alert.
html"><b>Loading...</b></uib-alert>
</div>
<!-- 结束容器 -->
```

index.html 的最后一部分是页脚。可以添加你想要的任何内容，在这里，我们只添加静态内容和版权文本。

```
<!-- 开始页脚 -->
<div class="page-footer">
    <hr/><div style="padding: 0 39%">&copy; 2016 Online Table
```

```
Reservation System</div>
  </div>
  <!-- 结束页脚 -->
</body>
</html>
```

app.js

`app.js` 是我们的主应用程序文件。因为我们在 `index.html` 中定义了它，所以只要 `index.html` 被调用，它就会被加载。



我们需要小心地不要把路由（URI）与 REST 端点混合。路由表示 SPA 的状态/视图。

由于我们使用了边缘服务器（代理服务器），一切都将通过它来访问，包括我们的 REST 端点。外部应用程序，包括用户界面将使用边缘服务器主机访问应用程序。可以在一些全局常量文件中配置它，然后在任何需要的地方使用它。这将允许你在一个单独的地方配置 REST 主机并在其他地方使用它。

```
'use strict';
/*
此调用初始化我们的应用程序并注册所有模块，作为数组中第二个参数传递。
*/
var otrsApp = angular.module('otrsApp', [
  'ui.router',
  'templates',
  'ui.bootstrap',
  'ngStorage',
  'otrsApp.httperror',
  'otrsApp.login',
  'otrsApp.restaurants'
])
/*
然后我们定义了默认路由 /restaurants
*/
.config([

```

```
'$stateProvider', '$urlRouterProvider',
    function ($stateProvider, $urlRouterProvider) {
        $urlRouterProvider.otherwise('/restaurants');
    })
/*  
此函数控制应用程序的流程并处理事件。  
*/
.controller('otrsAppCtrl', function ($scope, $injector,
restaurantService) {
    var controller = this;

    var AjaxHandler = $injector.get('AjaxHandler');
    var $rootScope = $injector.get('$rootScope');
    var log = $injector.get('$log');
    var sessionStorage = $injector.get('$sessionStorage');
    $scope.showSpinner = false;
/*
当用户搜索任何一家餐馆时，此函数就被调用。它使用Angular餐馆服务，我们将在下一节中搜索给定的搜索字符串时定义它。
*/
    $scope.search = function () {
        $scope.restaurantService = restaurantService;
        restaurantService.async().then(function () {
            $scope.restaurants = restaurantService.
search($scope.searchedValue);
        });
    }
/*
当状态更改时，新的控制器基于视图和配置控制流程，而现有的控制器被销毁。发生destroy事件时，此函数被调用。
*/
    $scope.$on('$destroy', function destroyed() {
        log.debug('otrsAppCtrl destroyed');
        controller = null;
        $scope = null;
    });
}
```

```

        $rootScope.fromState;
        $rootScope.fromStateParams;
        $rootScope.$on('$stateChangeSuccess', function (event,
toState, toParams, fromState, fromStateParams) {
            $rootScope.fromState = fromState;
            $rootScope.fromStateParams = fromStateParams;
        });

        // 实用程序方法
        $scope.isLoggedIn = function () {
            if (sessionStorage.session) {
                return true;
            } else {
                return false;
            }
        };

        /* 微调框状态 */
        $scope.isSpinnerShown = function () {
            return AjaxHandler.getSpinnerStatus();
        };

    })

/*
此对象加载时，此函数被执行。在这里我们正在设置为根作用域定义的用户对象。
*/
.run(['$rootScope', '$injector', '$state', function
($rootScope, $injector, $state) {
    $rootScope.restaurants = null;
    // 自引用
    var controller = this;
    // 注入外部的引用
    var log = $injector.get('$log');
    var sessionStorage = $injector.get('$sessionStorage');
    var AjaxHandler = $injector.get('AjaxHandler');

    if (sessionStorage.currentUser) {

```

```
$rootScope.currentUser = $sessionStorage.currentUser;  
} else {  
    $rootScope.currentUser = "Guest";  
    $sessionStorage.currentUser = ""  
}  
})
```

restaurants.js

`restaurants.js` 表示我们的应用程序 Angular 服务，我们将把它用于餐馆。我们知道服务有两种常见的用途——组织代码和在应用程序之间共享代码。因此，我们已经创建了一个餐馆服务，它将用在不同的模块，如搜索、列表、详细信息，等等。



服务是单例对象，由 AngularJS 服务工厂懒惰地实例化。

下面这段代码初始化餐馆服务模块，并加载必需的依赖项。

```
angular.module('otrsApp.restaurants', [  
    'ui.router',  
    'ui.bootstrap',  
    'ngStorage',  
    'ngResource'  
])
```

在配置信息中，我们使用 UI-Router 定义了 `otrsApp.restaurants` 模块的路由和状态。

首先我们通过传递 JSON 对象来定义 `restaurants` 状态，这个 JSON 对象包含指向路由 URI 的 URL、指向显示 `restaurants` 状态的 HTML 模板的模板 URL，以及将处理 `restaurants` 视图上的事件的控制器。

在 restaurants 视图 (route-/restaurants) 之上，也定义了一个嵌套的状态 restaurants.profileis，它将表示具体的餐馆。例如，/restaurant/1 会打开并显示 Id 为 1 的餐馆的概要资料（详细信息）页。用户在 restaurants 模板中的某个链接上单击时，此状态将会被调用。在此 ui sref ="restaurants.profile ({id: rest.id})" 中，rest 代表从 restaurantsview 获取的 restaurant 对象。

请注意，状态名称是'restaurants.profile'，这告诉 AngularJS UI Router，概要资料是一种 restaurants 状态的嵌套状态。

```
.config([
    '$stateProvider', '$urlRouterProvider',
    function ($stateProvider, $urlRouterProvider) {
        $stateProvider.state('restaurants', {
            url: '/restaurants',
            templateUrl: 'restaurants/restaurants.html',
            controller: 'RestaurantsCtrl'
        })
        // 餐馆显示页面
        .state('restaurants.profile', {
            url: '/:id',
            views: {
                '@': {
                    templateUrl: 'restaurants/restaurant.html',
                    controller: 'RestaurantCtrl'
                }
            }
        });
    }
]);
```

在接下来的这段代码中，我们使用 Angular 工厂服务类型来定义餐馆服务。餐馆服务加载时使用 REST 调用从服务器获取餐馆的列表。它提供了一个列表、搜索餐馆操作和餐馆数据。

```
.factory('restaurantService', function ($injector, $q) {
    var log = $injector.get('$log');
    var ajaxHandler = $injector.get('AjaxHandler');
    var deffered = $q.defer();
    var restaurantService = {};
    restaurantService.restaurants = [];
    restaurantService.orignalRestaurants = [];
    restaurantService.async = function () {
        ajaxHandler.startSpinner();
        if (restaurantService.restaurants.length === 0) {
```

```
ajaxHandler.get('/api/restaurant')
    .success(function (data, status, headers, config) {
        log.debug('Getting restaurants');
        sessionStorage.apiActive = true;
        log.debug("if Restaurants --> " +
restaurantService.restaurants.length);
        restaurantService.restaurants = data;
        ajaxHandler.stopSpinner();
        deffered.resolve();
    })
    .error(function (error, status, headers, config) {
        restaurantService.restaurants = mockdata;
        ajaxHandler.stopSpinner();
        deffered.resolve();
    });
    return deffered.promise;
} else {
    deffered.resolve();
    ajaxHandler.stopSpinner();
    return deffered.promise;
}
};

restaurantService.list = function () {
    return restaurantService.restaurants;
};
restaurantService.add = function () {
    console.log("called add");
    restaurantService.restaurants.push(
    {
        id: 103,
        name: 'Chi Cha\'s Noodles',
        address: '13 W. St., Eastern Park, New County, Paris',
    });
};
restaurantService.search = function (searchedValue) {
    ajaxHandler.startSpinner();
    if (!searchedValue) {
```

```
if (restaurantService.originalRestaurants.length > 0) {
    restaurantService.restaurants =
    restaurantService.originalRestaurants;
}
deffered.resolve();
ajaxHandler.stopSpinner();
return deffered.promise;
} else {
    ajaxHandler.get('/api/restaurant?name=' + searchedValue)
        .success(function (data, status, headers, config) {
            log.debug('Getting restaurants');
            sessionStorage.apiActive = true;
            log.debug("if Restaurants --> " +
            restaurantService.restaurants.length);
            if (restaurantService.
originalRestaurants.length < 1) {
                restaurantService.
originalRestaurants = restaurantService.restaurants;
            }
            restaurantService.restaurants = data;
            ajaxHandler.stopSpinner();
            deffered.resolve();
        })
        .error(function (error, status, headers, config) {
            if (restaurantService.
originalRestaurants.length < 1) {
                restaurantService.
originalRestaurants = restaurantService.restaurants;
            }
            restaurantService.restaurants = [];
            restaurantService.restaurants.push(
            {
                id: 104,
                name: 'Gibsons - Chicago Rush St.',
                address: '1028 N. Rush
St., Rush & Division, Cook County, Paris'
            });
        });
}
```

```

        restaurantService.restaurants.push(
        {
            id: 105,
            name: 'Harry Caray\'s Italian Steakhouse',
            address: '33 W. Kinzie
St., River North, Cook County, Paris',
        });
        ajaxHandler.stopSpinner();
        deffered.resolve();
    });
    return deffered.promise;
}
};

return restaurantService;
})

```

在 restaurants.js 模块的下一部分，我们将为路由配置中定义的餐馆和 restaurants.profile 状态添加两个控制器。这两个控制器是 RestaurantsCtrl 和 RestaurantCtrlthat，它们分别处理 restaurants 状态和 restaurants.profiles 状态。

RestaurantsCtrl 很简单，因为它使用餐馆服务的列表方法来加载餐馆的数据。

```

.controller('RestaurantsCtrl', function ($scope, restaurantService) {
    $scope.restaurantService = restaurantService;
    restaurantService.async().then(function () {
        $scope.restaurants = restaurantService.list();
    });
})

```

RestaurantCtrl 负责显示一个给定 ID 的餐馆的详细信息，它也负责在显示的餐馆上执行预订操作。当我们设计餐馆详细信息页面与预订选项时，将使用此控件。

```

.controller('RestaurantCtrl', function ($scope, $state,
stateParams, $injector, restaurantService) {
    var $sessionStorage = $injector.get('$sessionStorage');
    $scope.format = 'dd MMMM yyyy';
    $scope.today = $scope.dt = new Date();
})

```

```

$scope.dateOptions = {
    formatYear: 'yy',
    maxDate: new Date(). setDate($scope.today.getDate() + 180),
    minDate: $scope.today.getDate(),
    startingDay: 1
};

$scope.popup1 = {
opened: false
};

$scope.altInputFormats = ['M!/d!/yyyy'];
$scope.open1 = function () {
    $scope.popup1.opened = true;
};

$scope.hstep = 1;
$scope.mstep = 30;

if ($sessionStorage.reservationData) {
    $scope.restaurant = $sessionStorage.reservationData.
    t;                                                 restaurant
    $scope.dt = new Date($sessionStorage.reservationData.tm);
    $scope.tm = $scope.dt;
} else {
    $scope.dt.setDate($scope.today.getDate() + 1);
    $scope.tm = $scope.dt;
    $scope.tm.setHours(19);
    $scope.tm.setMinutes(30);
    restaurantService.async().then(function () {
        angular.forEach(restaurantService.list(), function
key) {
            if (value.id === parseInt($stateParams.id)) {
                $scope.restaurant = value;
            }
        });
    });
}

$scope.book = function () {

```

```
var tempHour = $scope.tm.getHours();
var tempMinute = $scope.tm.getMinutes();
$scope.tm = $scope.dt;
$scope.tm.setHours(tempHour);
$scope.tm.setMinutes(tempMinute);
if ($sessionStorage.currentUser) {
    console.log("$scope.tm --> " + $scope.tm);
    alert("Booking Confirmed!!!!");
    $sessionStorage.reservationData = null;
    $state.go("restaurants");
} else {
    $sessionStorage.reservationData = {};
    $sessionStorage.reservationData.restaurant = $scope.restaurant;
    $sessionStorage.reservationData.tm = $scope.tm;
    $state.go("login");
}
}
})
```

我们也在 `restaurants.js` 模块中添加了一些过滤器来设置日期和时间格式。这些过滤器对输入数据执行以下格式化操作：

- date1: 以'MMM dd yyyy'格式返回输入的日期, 例如 13-Apr-2016
 - time1: 以'HH:mm:ss'格式返回输入的时间, 例如 11:55:04
 - dateTime1: 以'dd MMM yyyy HH:mm:ss'格式返回输入的日期和时间, 例如
13-Apr-2016 11:55:04

在下面的代码片段中，我们已经应用了这三个过滤器：

```
.filter('date1', function ($filter) {
  return function (argDate) {
    if (argDate) {
      var d = $filter('date')(new Date(argDate), 'dd MMM yyyy');
      return d.toString();
    }
    return "";
  }
})
```

```

    };
  })
.filter('time1', function ($filter) {
  return function (argTime) {
    if (argTime) {
      return $filter('date')(new Date(argTime), 'HH:mm:ss');
    }
    return "";
  };
})
.filter('datetimel', function ($filter) {
  return function (argDateTime) {
    if (argDateTime) {
      return $filter('date')(new Date(argDateTime), 'dd MMM yyyy
HH:mm a');
    }
    return "";
  };
});

```

restaurants.html

我们需要添加已经为 `restaurants.profile` 状态定义的模板。正如你可以在模板中看到的，我们使用 `ng-repeat` 指令来迭代由 `restaurantService.restaurants` 返回的对象列表。`RestaurantService` 作用域变量是在控制器中定义的，'RestaurantsCtrl' 在餐馆状态中与此模板相关联。

```

<h3>Famous Gourmet Restaurants in Paris</h3>
<div class="row">
  <div class="col-md-12">
    <table class="table table-bordered table-striped">
      <thead>
        <tr>
          <th>#Id</th>
          <th>Name</th>
          <th>Address</th>
        </tr>
      </thead>

```

```
</thead>
<tbody>
    <tr ng-repeat="rest in restaurantService.restaurants">
        <td>{{rest.id}}</td>
        <td><a ui-sref="restaurants.profile({id: rest.id})">
{{rest.name}}</a></td>
        <td>{{rest.address}}</td>
    </tr>
</tbody>
</table>
</div>
</div>
```

搜索餐馆

我们已在主页 index.html 的 header 部分中添加搜索表单，使我们能够搜索餐馆。搜索餐馆功能将使用如前所述的相同文件。它利用 app.js（搜索表单处理程序）、restaurants.js（餐馆服务）和 restaurants.html 来显示搜索到的记录。

The screenshot shows the homepage of the "Online Table Reservation System". At the top, there is a navigation bar with the title "Online Table Reservation System", a search input field containing the letter "C", a "Go" button, and a welcome message "Welcome Guest!". Below the navigation bar, the main content area has a heading "Famous Gourmet Restaurants in Paris". A table displays two records:

#Id	Name	Address
104	Gibsons - Chicago Rush St.	1028 N. Rush St., Rush & Division, Cook County, Paris
105	Harry Caray's Italian Steakhouse	33 W. Kinzie St., River North, Cook County, Paris

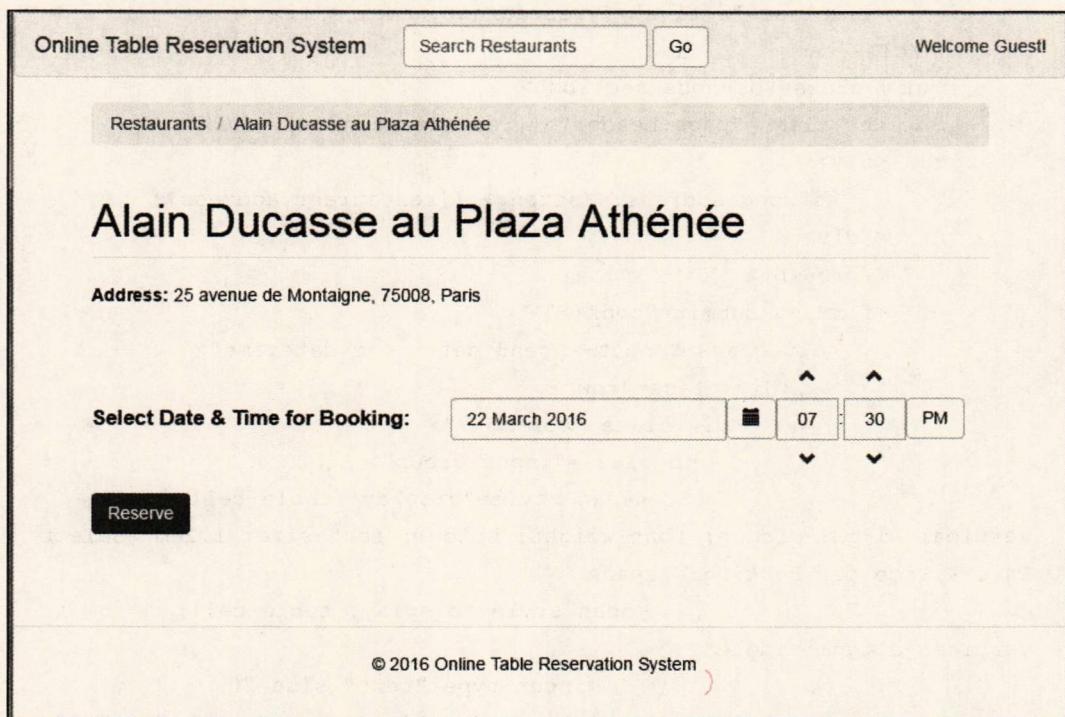
At the bottom of the page, there is a copyright notice: "© 2016 Online Table Reservation System".

OTRS主页与餐馆清单

餐馆详细信息与预订选项

餐馆详细信息与预订选项将是内容区域（页面的中间部分）的一部分。这将包含顶部的痕迹导航，以及链接到餐馆清单页面的 Restaurants，后面跟着餐馆的名称和地址。最后一部分将包含预订部分，其中包含日期时间选择框和预订按钮。

此页面将看起来像下面的屏幕截图。



餐馆详细信息页面与预订选项

在这里，我们将使用与 `restaurants.js` 中声明的同一个餐馆服务。如状态 `restaurants.profile` 所述，唯一的变化就是模板。此模板将使用 `restaurant.html` 来定义。

restaurant.html

正如你所看到的，痕迹导航使用由 `ui-sref` 属性定义的餐馆路由。在此模板中设计

的预订表格调用 book() 函数，此函数在控制器 RestaurantCtrl 中用表单提交的指令 ng-submiton 定义。

```
<div class="row">
<div class="row">
    <div class="col-md-12">
        <ol class="breadcrumb">
            <li><a ui-sref="restaurants">Restaurants</a></li>
            <li class="active">{{restaurant.name}}</li>
        </ol>
        <div class="bs-docs-section">
            <h1 class="page-header">{{restaurant.name}}</h1>
            <div>
                <strong>Address:</strong> {{restaurant.address}}
            </div>
            <br><br>
            <form ng-submit="book()">
                <div class="input-append date form_datetime">
                    <div class="row">
                        <div class="col-md-7">
                            <p class="input-group">
                                <span style="display: table-cell;
vertical-align: middle; font-weight: bolder; font-size: 1.2em">Select
Date & Time for Booking:</span>
                                <span style="display: table-cell;
vertical-align: middle">
                                    <input type="text" size=20
class="form-control" uib-datepicker-popup="{{format}}" ng-model="dt"
is-open="popup1.opened" datepicker-options="dateOptions" ng-
required="true" close-text="Close" alt-input-formats="altInputFormats"
/>
                                </span>
                            </p>
                            <span class="input-group-btn">
                                <button type="button" class="btn
btn-default" ng-click="open1()"><i class="glyphicon glyphicon-
calendar"></i></button>
                            </span>
                        </div>
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>
```

```
<uib-timepicker ng-model="tm" ng-
change="changed()" hour-step="hstep" minute-step="mstep"></uib-
timepicker>
    </p>
</div>
</div></div>
<div class="form-group">
    <button class="btn btn-primary"
type="submit">Reserve</button>
</div>
</form><br><br>
</div>
</div>
</div>
```

登录页面

当用户选择了预订的日期和时间后，点击餐馆详细信息页面上的 **Reserve** 按钮时，餐馆详细信息页面会检查用户是否已登录。如果用户未登录，那么会显示登录页面。它看起来像下面的屏幕截图。

The screenshot shows a web browser window for the "Online Table Reservation System". At the top, there is a navigation bar with the system name and a search bar labeled "Search Restaurants" with a "Go" button. Below the navigation bar, the main content area has a title "Login". It contains two input fields: one for "username" and one for "password". At the bottom of the form are two buttons: "Login" and "Cancel". In the bottom right corner of the main content area, there is a copyright notice: "© 2016 Online Table Reservation System".

登录页面

 我们不从服务器上对用户进行身份验证。相反，我们只把用户名
填充在会话存储和实现工作流中的根作用域中。

一旦用户登录成功，用户将被重定向回相同的预订页面并保持原来的状态，然后用户可以继续处理预订。**Login** 页面基本上使用如下两个文件：login.html 和 login.js。

login.html

Login.html 模板由两个输入字段（用户名和密码），以及 **Login** 按钮和 **Cancel** 链接组成。**Cancel** 链接重置表单，而 **Login** 按钮提交登录表单。

在这里，我们使用 LoginCtrl 和 ng-controller 指令。**Login** 表单使用调用 LoginCtrl 的 submit 函数的 ng-submit 指令提交。输入的值首先使用 ng-model 指令收集，然后使用其各自的属性-_email 和 _password 提交。

```
<div ng-controller="LoginCtrl as loginC" style="max-width: 300px">
    <h3>Login</h3>
    <div class="form-container">
        <form ng-submit="loginC.submit(_email, _password)">
            <div class="form-group">
                <label for="username" class="sr-only">Username</label>
                <input type="text" id="username" class="form-control" placeholder="username" ng-model="._email" required autofocus />
            </div>
            <div class="form-group">
                <label for="password" class="sr-only">Password</label>
                <input type="password" id="password" class="form-control" placeholder="password" ng-model="._password" />
            </div>
            <div class="form-group">
                <button class="btn btn-primary" type="submit">Login</button>
                <button class="btn btn-link" ng-click="loginC.cancel()">Cancel</button>
            </div>
        </form>
    </div>
</div>
```

```
</div>
</div>
```

login.js

登录模块是在 `login.js` 中定义的，它包含并加载使用此模块功能的依赖项。状态 `login` 是在 `config` 函数的帮助下定义的，此函数会取得包含 `url`、控制器和 `templateUrl` 属性的 JSON 对象。

在控制器里面，我们定义取消和提交操作，它们是从 `login.html` 模板调用的。

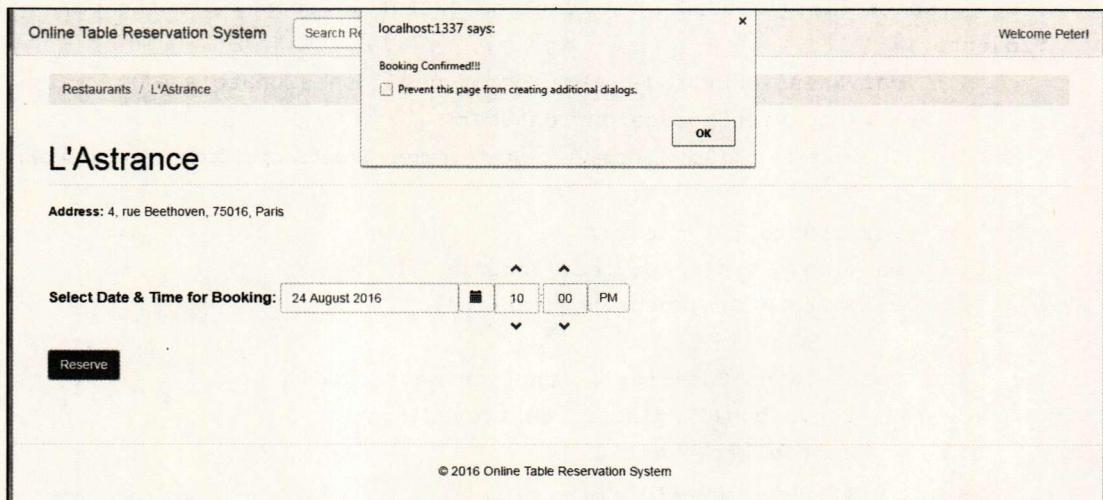
```
angular.module('otrsApp.login', [
  'ui.router',
  'ngStorage'
])
.config(function config($stateProvider) {
  $stateProvider.state('login', {
    url: '/login',
    controller: 'LoginCtrl',
    templateUrl: 'login/login.html'
  });
})
.controller('LoginCtrl', function ($state, $scope, $rootScope,
$injector) {
  var $sessionStorage = $injector.get('$sessionStorage');
  if ($sessionStorage.currentUser) {
    $state.go($rootScope.fromState.name, $rootScope.fromStateParams);
  }
  var controller = this;
  var log = $injector.get('$log');
  var http = $injector.get('$http');

  $scope.$on('$destroy', function destroyed() {
    log.debug('LoginCtrl destroyed');
    controller = null;
    $scope = null;
  });
})
```

```
        this.cancel = function () {
            $scope.$dismiss;
            $state.go('restaurants');
        }
        console.log("Current --> " + $state.current);
        this.submit = function (username, password) {
            $rootScope.currentUser = username;
            $sessionStorage.currentUser = username;
            if ($rootScope.fromState.name) {
                $state.go($rootScope.fromState.name,
$rootScope.fromStateParams);
            } else {
                $state.go("restaurants");
            }
        };
    });
});
```

预订确认

一旦用户登录成功并点击 **Reservation** 按钮，餐馆控制器就显示确认警告框，如以下屏幕截图所示。



餐馆详细信息页面与预订确认

设置 web 应用程序

我们打算为用户界面应用程序开发使用最新的技术栈，我们将使用 Node.js 和 npm（Node.js 包管理器），它们为开发服务器端 JavaScript web 应用程序提供开放源码的运行时环境。

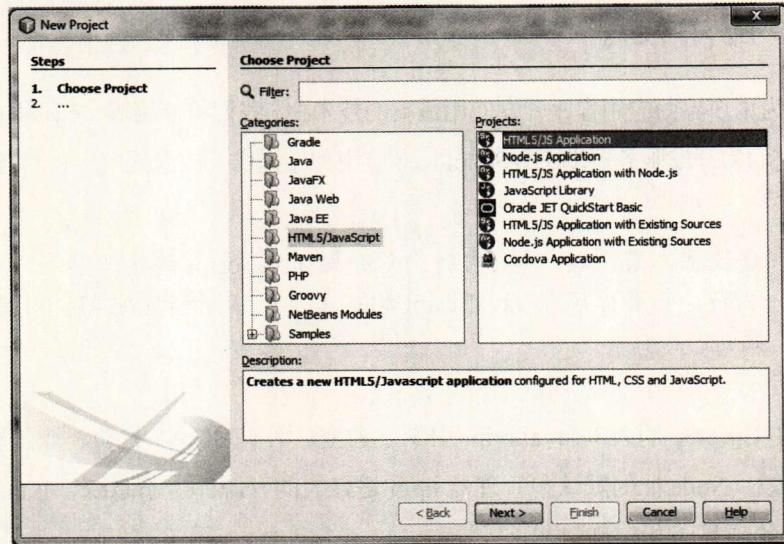


建议把本节通读一遍，它将向你介绍 JavaScript 构建工具和环境。

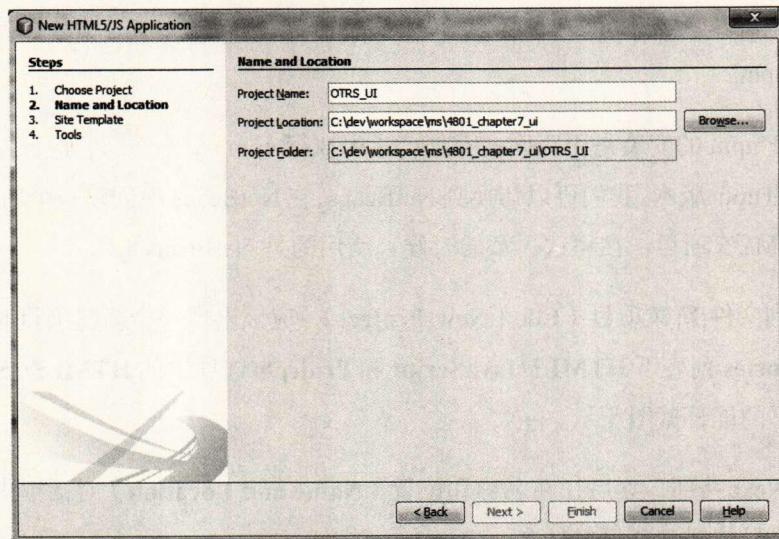
然而，如果你了解 JavaScript 构建工具或不想研究它们，则可以跳过。

Node.js 在 Chrome 的 V8 JavaScript 引擎上建立，并使用事件驱动、非阻塞 I/O，这使得它轻便而高效。Node.js 的默认包管理器 npm 是最大的开源库生态系统，它允许安装 Node 程序并使得依赖项易于指定和链接。

1. 如果还没有安装 npm，我们需要首先安装它。它是一个先决条件。可以检查位于 <https://docs.npmjs.com/getting-started/installing-node> 的链接来安装 npm。
2. 要检查 npm 的安装是否正确，请在 CLI 中执行 npm -v 命令。它应该在输出中返回安装的 npm 版本。我们可以切换到 NetBeans，在 NetBeans 中创建一个新的 AngularJS JS HTML 5 项目。在写这一章的时候，使用的是 NetBeans 8.1。
3. 导航到文件|新建项目（File | New Project），应该出现一个新建项目对话框。选择 Categories 列表下 HTML5/JavaScript 和 Projects 选项中的 HTML5/JS Application，如下页的屏幕截图所示。
4. 单击 Next 按钮。然后在名称和位置（Name and Location）对话框中输入项目名称、项目位置和项目文件夹，单击 Next 按钮。

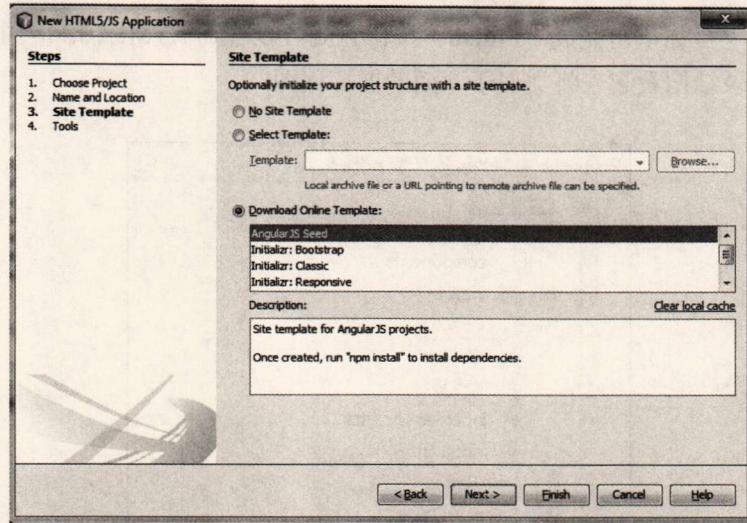


NetBeans——新建HTML5/JavaScript项目



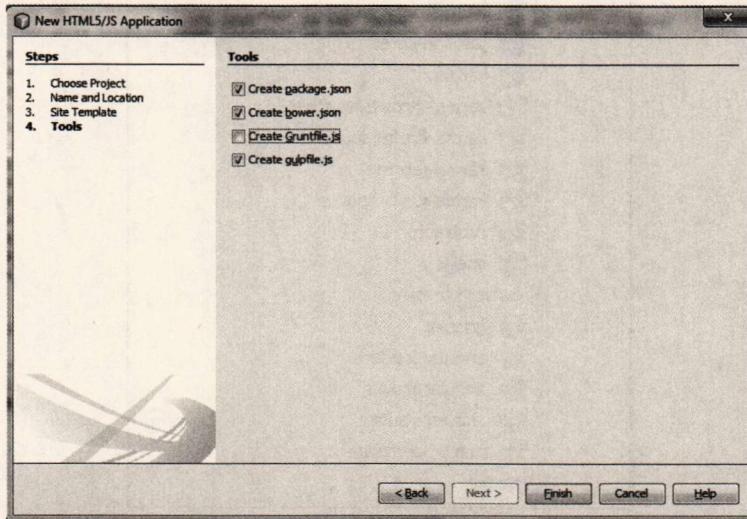
NetBeans新建项目——名称和位置

5. 在网站模板(Site Template)对话框中，在下载在线模板(Download Online Template)下选择 AngularJS Seed 条目：选择并单击 Next 按钮。AngularJS Seed 项目都可在<https://github.com/angular/angular-seed>获取。



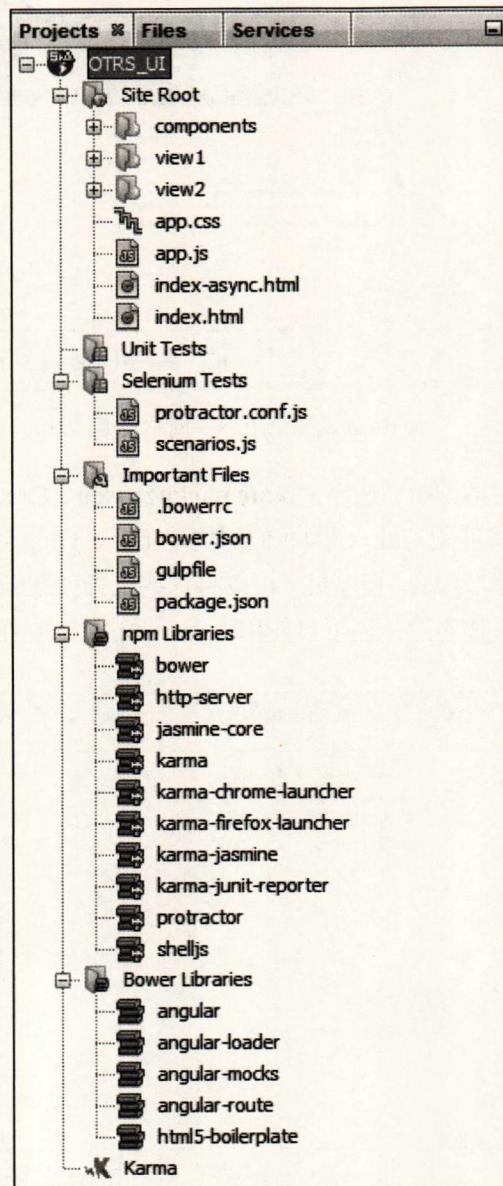
NetBeans新建项目——网站模板

6. 在工具 (Tools) 对话框中, 选择 **Create package.json**、**Create bower.json** 和 **Create gulpfile.js**。我们将使用 Gulp 作为我们的构建工具。Gulp 和 Grunt 是两种最受欢迎的 JS 构建框架。作为 Java 程序员, 可以将这些工具关联到 ANT。这两种工具都有各自的强项。如果你愿意, 还可以使用 Gruntfile.js 作为构建工具。



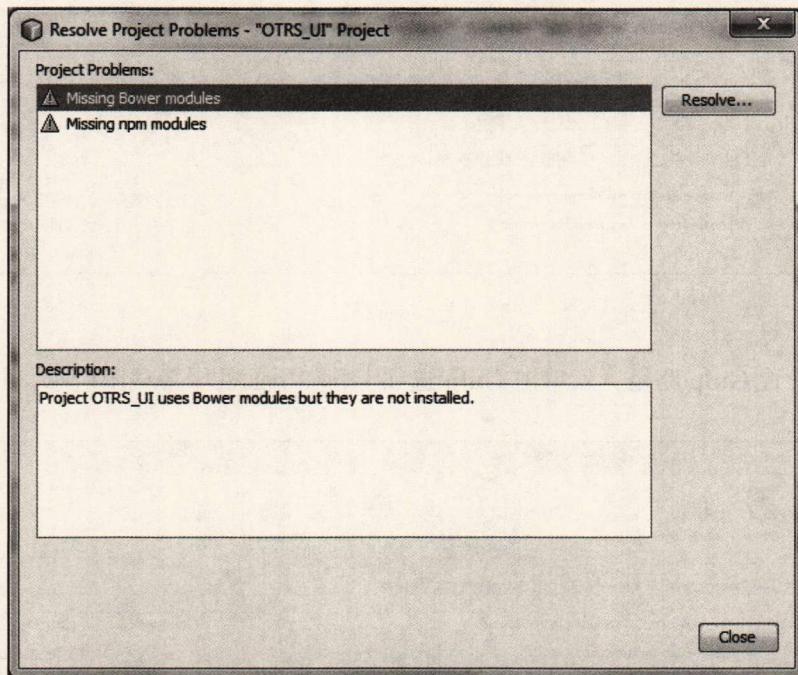
Netbeans新建项目——工具

7. 现在，一旦你单击完成（Finish）按钮，就可以看到 HTML5/JS 应用程序目录和文件。目录结构看起来将类似于下面的屏幕截图。



AngularJS种子目录结构

- 如果所需的依赖项的配置有任何不正确，还将在项目中看到感叹号标记。可以使用鼠标右键单击此项目，然后选择解决项目问题（Resolve Project Problems）选项来解决项目问题。



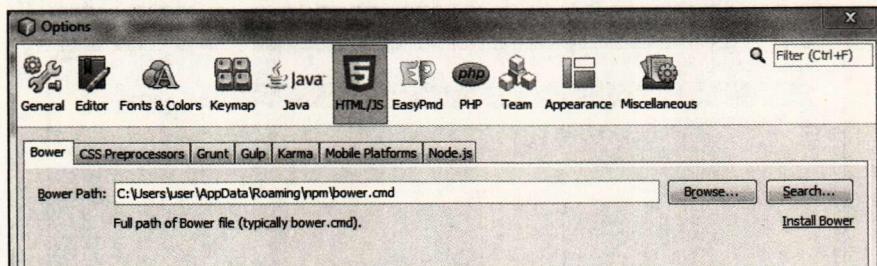
解决项目问题对话框

- 理想的情况是，如果单击 **Resolve...** 按钮，NetBeans 解决了项目存在的问题。
- 也可以通过给出一些 JS 模块，如 bower、Gulp 和 node 的正确路径来解决一些问题：
 - Bower**: OTRS 的应用程序管理 JavaScript 库所需要的
 - Gulp**: 构建项目所需的任务执行器，类似于 ANT
 - Node**: 用于执行 OTRS 应用程序的服务器端



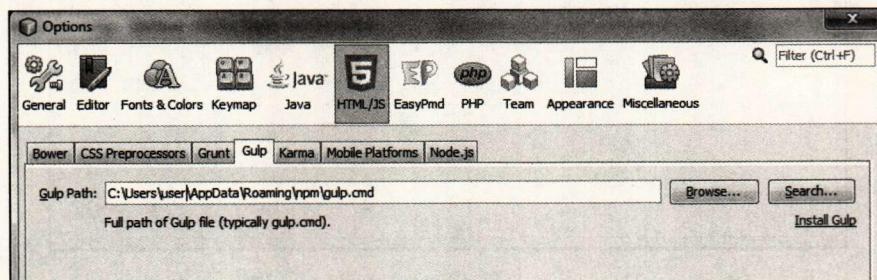
Bower 是一个依赖项管理工具，类似于 npm。npm 用于安装 Node.js 模块，而 Bower 用于管理 web 应用程序的库/组件。

11. 单击工具 (Tools) 菜单并选择选项 (Options)。现在，在 HTML/JS 工具 (顶部横栏的图标) 中设置 bower、Gulp 和 node.js 路径，如下面的屏幕截图所示。要设置 bower 路径，就单击 bower 选项卡，如下面的屏幕截图所示，并更新路径。



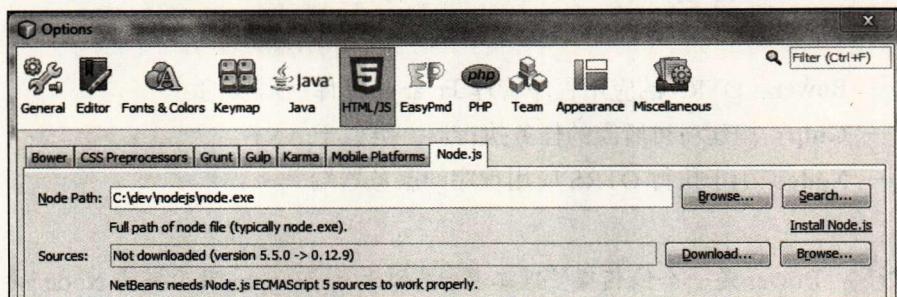
设置bower路径

12. 要设置 Gulp 路径，就单击 Gulp 选项卡，如下面的屏幕截图所示，并更新路径。



设置Gulp路径

13. 要设置 Node 路径，就单击 Node.js 选项卡，如下面的屏幕截图所示，并更新路径。



设置Node路径

14. 一旦完成这些, **package.json** 看起来将类似于下面这样。我们已修改少数条目, 如名称、描述、依赖关系等的值:

```
{  
  "name": "otrs-ui",  
  "private": true,  
  "version": "1.0.0",  
  "description": "Online Table Reservation System",  
  "main": "index.js",  
  "license": "MIT",  
  "dependencies": {  
    "coffee-script": "^1.10.0",  
    "gulp-AngularJS-templatecache": "^1.8.0",  
    "del": "^1.1.1",  
    "gulp-connect": "^3.1.0",  
    "gulp-file-include": "^0.13.7",  
    "gulp-sass": "^2.2.0",  
    "gulp-util": "^3.0.7",  
    "run-sequence": "^1.1.5"  
  },  
  "devDependencies": {  
    "coffee-script": "*",  
    "gulp-sass": "*",  
    "bower": "^1.3.1",  
    "http-server": "^0.6.1",  
    "jasmine-core": "^2.3.4",  
    "karma": "~0.12",  
    "karma-chrome-launcher": "^0.1.12",  
    "karma-firefox-launcher": "^0.1.6",  
    "karma-jasmine": "^0.3.5",  
    "karma-junit-reporter": "^0.2.2",  
    "protractor": "^2.1.0",  
    "shelljs": "^0.2.6"  
  },  
  "scripts": {  
    "postinstall": "bower install",  
    "prestart": "npm install",  
  }  
}
```

```

    "start": "http-server -a localhost -p 8000 -c-1",
    "pretest": "npm install",
    "test": "karma start karma.conf.js",
    "test-single-run": "karma start karma.conf.js --single-run",
    "preupdate-webdriver": "npm install",
    "update-webdriver": "webdriver-manager update",
    "preprotractor": "npm run update-webdriver",
    "protractor": "protractor e2e-tests/protractor.conf.js",
    "update-index-async": "node -e \"require('shelljs/global');
global'; sed('-i', '/\\\\\\\\@@NG_LOADER_START@@
[\\s\\S]*\\\\\\\\@@NG_LOADER_END@@', '//@@NG_LOADER_
START@@\\n' + sed(/sourceMappingURL=AngularJS-loader.min.
js.map/, 'sourceMappingURL=bower_components/AngularJS-loader/
AngularJS-loader.min.js.map', 'app/bower_components/AngularJS-
loader/AngularJS-loader.min.js') + '\\n//@@NG_LOADER_END@@', 'app/
index-async.html');\""
}
}

```

15. 然后，我们会更新 bower.json，如下所示：

```
{
  "name": "OTRS-UI",
  "description": "OTRS-UI",
  "version": "0.0.1",
  "license": "MIT",
  "private": true,
  "dependencies": {
    "AngularJS": "~1.5.0",
    "AngularJS-ui-router": "~0.2.18",
    "AngularJS-mocks": "~1.5.0",
    "AngularJS-bootstrap": "~1.2.1",
    "AngularJS-touch": "~1.5.0",
    "bootstrap-sass-official": "~3.3.6",
    "AngularJS-route": "~1.5.0",
    "AngularJS-loader": "~1.5.0",
    "ngstorage": "^0.3.10",
    "AngularJS-resource": "~1.5.0",
  }
}
```

```

    "html5-boilerplate": "~5.2.0"
  }
}

```

16. 下一步，我们将修改 `.bowerrc` 文件，指定 Bower 用来存储在 `bower.json` 中定义的组件的目录。我们将把 bower 组件存储在 `app` 目录下。

```

{
  "directory": "app/bower_components"
}

```

17. 下一步，我们将设置 `gulpfile.js`。我们会使用 CoffeeScript 来定义 Gulp 任务。因此，我们在 `gulpfile.js` 中只定义 CoffeeScript，而实际的任务将在 `gulpfile.coffee` 中定义。让我们看看 `gulpfile.js` 的内容：

```

require('coffee-script/register');
require('./gulpfile.coffee');

```

18. 在这一步，我们将定义 `gulp` 配置。我们使用 CoffeeScript 来定义 `gulp` 文件。写在 CoffeeScript 中的 Gulp 文件名是 `gulpfile.coffee`。默认的任务定义为 `default_sequence`：

```
default_sequence = ['connect', 'build', 'watch']
```

根据定义的默认顺序任务，首先它将连接到服务器，然后构建 web 应用程序，并监控（`watch`）所做的更改。监控将有助于呈现我们对代码的修改，并将立即显示在 UI 上。

在此脚本中最重要的部分是 `connect` 和 `watch`，别的内容都一目了然。

- `gulp-connect`: 这是一个 `gulp` 插件来运行 web 服务器，它还允许现场重新加载。
- `gulp-watch`: 这是一个文件监控程序，使用 `chokidar` 并发出 `vinyl` 对象（描述此文件的对象——其路径和内容）。简单地说，我们可以认为，`gulp-watch` 监控文件的更改并触发任务。

gulpfile.coffee:

```
gulp          = require('gulp')
```

```
gutil      = require('gulp-util')
del        = require('del');
clean      = require('gulp-clean')
connect    = require('gulp-connect')
fileinclude = require('gulp-file-include')
runSequence = require('run-sequence')
templateCache = require('gulp-AngularJS-templatecache')
sass       = require('gulp-sass')

paths =
  scripts:
    src: ['app/src/scripts/**/*.js']
    dest: 'public/scripts'
  scripts2:
    src: ['app/src/views/**/*.js']
    dest: 'public/scripts'
  styles:
    src: ['app/src/styles/**/*.scss']
    dest: 'public/styles'
  fonts:
    src: ['app/src/fonts/**/*']
    dest: 'public/fonts'
  images:
    src: ['app/src/images/**/*']
    dest: 'public/images'
  templates:
    src: ['app/src/views/**/*.html']
    dest: 'public/scripts'
  html:
    src: ['app/src/*.html']
    dest: 'public'
  bower:
    src: ['app/bower_components/**/*']
    dest: 'public/bower_components'

#把bower 模块复制到public目录
gulp.task 'bower', ->
```

```
gulp.src(paths.bower.src)
  .pipe gulp.dest(paths.bower.dest)
  .pipe connect.reload()

#把scripts复制到public目录
gulp.task 'scripts', ->
  gulp.src(paths.scripts.src)
  .pipe gulp.dest(paths.scripts.dest)
  .pipe connect.reload()

#把scripts2复制到public目录
gulp.task 'scripts2', ->
  gulp.src(paths.scripts2.src)
  .pipe gulp.dest(paths.scripts2.dest)
  .pipe connect.reload()

#把styles复制到public目录
gulp.task 'styles', ->
  gulp.src(paths.styles.src)
  .pipe sass()
  .pipe gulp.dest(paths.styles.dest)
  .pipe connect.reload()

#把images复制到public目录
gulp.task 'images', ->
  gulp.src(paths.images.src)
  .pipe gulp.dest(paths.images.dest)
  .pipe connect.reload()

#把fonts复制到public目录
gulp.task 'fonts', ->
  gulp.src(paths.fonts.src)
  .pipe gulp.dest(paths.fonts.dest)
  .pipe connect.reload()

#把html复制到public目录
gulp.task 'html', ->
```

```
gulp.src(paths.html.src)
  .pipe gulp.dest(paths.html.dest)
  .pipe connect.reload()

#在一个单独的 js 文件中编译AngularJS 模板
gulp.task 'templates', ->
  gulp.src(paths.templates.src)
  .pipe(templateCache({standalone: true}))
  .pipe(gulp.dest(paths.templates.dest))

#从公共目录删除内容
gulp.task 'clean', (callback) ->
  del ['./public/**/*'], callback;

#Gulp 连接任务，部署公共目录
gulp.task 'connect', ->
  connect.server
  root: ['./public']
  port: 1337
  livereload: true

gulp.task 'watch', ->
  gulp.watch paths.scripts.src, ['scripts']
  gulp.watch paths.scripts2.src, ['scripts2']
  gulp.watch paths.styles.src, ['styles']
  gulp.watch paths.fonts.src, ['fonts']
  gulp.watch paths.html.src, ['html']
  gulp.watch paths.images.src, ['images']
  gulp.watch paths.templates.src, ['templates']

gulp.task 'build', ['bower', 'scripts', 'scripts2', 'styles',
  'fonts', 'images', 'templates', 'html']

default_sequence = ['connect', 'build', 'watch']

gulp.task 'default', default_sequence

gutil.log 'Server started and waiting for changes'
```

19. 一旦我们完成前面的更改，我们会使用以下命令安装 Gulp：

```
npm install --no-optional gulp
```

20. 此外，我们使用下面的命令安装其他 Gulp 库，如 gulp-clean、gulp-connect，等等：

```
npm install --save --no-optional gulp-util gulp-clean gulp-connect
gulp-file-include run-sequence gulp-AngularJS-templatecache gulp-
sass
```

21. 现在，我们可以使用下面的命令安装在 bower.json 文件中定义的 bower 依赖项：

```
bower install - save
```

```
$ bower install --save
bower angular-route#~1.4.0 not-cached git://github.com/angular/bower-angular-route.git#~1.4.0
bower angular-route#~1.4.0 resolve git://github.com/angular/bower-angular-route.git#~1.4.0
bower angular#~1.4.0 not-cached git://github.com/angular/bower-angular.git#~1.4.0
bower angular#~1.4.0 resolve git://github.com/angular/bower-angular.git#~1.4.0
bower angular-loader#~1.4.0 not-cached git://github.com/angular/bower-angular-loader.git#~1.4.0
bower angular-loader#~1.4.0 resolve git://github.com/angular/bower-angular-loader.git#~1.4.0
bower angular-mocks#~1.4.0 not-cached git://github.com/angular/bower-angular-mocks.git#~1.4.0
bower angular-mocks#~1.4.0 resolve git://github.com/angular/bower-angular-mocks.git#~1.4.0
bower html5-boilerplate#~5.2.0 not-cached git://github.com/h5bp/html5-boilerplate.git#~5.2.0
bower html5-boilerplate#~5.2.0 resolve git://github.com/h5bp/html5-boilerplate.git#~5.2.0
bower html5-boilerplate#~5.2.0 download https://github.com/h5bp/html5-boilerplate/archive/5.2.0.tar.gz
bower angular#~1.4.0 download https://github.com/angular/bower-angular/archive/v1.4.9.tar.gz
bower angular-loader#~1.4.0 download https://github.com/angular/bower-angular-loader/archive/v1.4.9.tar.gz
bower angular-mocks#~1.4.0 download https://github.com/angular/bower-angular-mocks/archive/v1.4.9.tar.gz
bower angular-loader#~1.4.0 extract archive.tar.gz
bower angular-loader#~1.4.0 resolved git://github.com/angular/bower-angular-loader.git#1.4.9
bower html5-boilerplate#~5.2.0 extract archive.tar.gz
bower angular-route#~1.4.0 extract archive.tar.gz
bower angular-route#~1.4.0 resolved git://github.com/angular/bower-angular-route.git#1.4.9
bower html5-boilerplate#~5.2.0 invalid-meta html5-boilerplate is missing "main" entry in bower.json
bower html5-boilerplate#~5.2.0 invalid-meta html5-boilerplate is missing "ignore" entry in bower.json
bower html5-boilerplate#~5.2.0 resolved git://github.com/h5bp/html5-boilerplate.git#5.2.0
bower angular-mocks#~1.4.0 extract archive.tar.gz
bower angular-mocks#~1.4.0 resolved git://github.com/angular/bower-angular-mocks.git#1.4.9
bower angular#~1.4.0 progress received 0.3MB of 0.9MB downloaded, 33%
bower angular#~1.4.0 progress received 0.3MB of 0.9MB downloaded, 60%
bower angular#~1.4.0 progress received 0.4MB of 0.9MB downloaded, 77%
bower angular#~1.4.0 progress received 0.4MB of 0.9MB downloaded, 88%
bower angular#~1.4.0 progress received 0.5MB of 0.9MB downloaded, 98%
bower angular#~1.4.0 extract archive.tar.gz
bower angular-loader#~1.4.0 resolved git://github.com/angular/bower-angular.git#1.4.9
bower angular-route#~1.4.0 install angular-loader#1.4.9
bower angular-route#~1.4.0 install angular-route#1.4.9
bower html5-boilerplate#~5.2.0 install html5-boilerplate#5.2.0
bower angular-mocks#~1.4.0 install angular-mocks#1.4.9
bower angular#~1.4.0 install angular#1.4.9

angular-loader#1.4.9 app\bower_components\angular-loader
  \-- angular#1.4.9

angular-route#1.4.9 app\bower_components\angular-route
  \-- angular#1.4.9

html5-boilerplate#5.2.0 app\bower_components\html5-boilerplate

angular-mocks#1.4.9 app\bower_components\angular-mocks
  \-- angular#1.4.9

angular#1.4.9 app\bower_components\angular
```

示例输出——bower install --save

22. 这是安装程序的最后一步。在这里，我们将确认目录结构应该像下面这样。我们
会把 `src` 和发布的工件(在`./public` 目录)作为单独的目录保留。因此，以下的
目录结构不同于默认 AngularJS 种子项目：

```
+---app
|   +---bower_components
|   |   +---AngularJS
|   |   +---AngularJS-bootstrap
|   |   +---AngularJS-loader
|   |   +---AngularJS-mocks
|   |   +---AngularJS-resource
|   |   +---AngularJS-route
|   |   +---AngularJS-touch
|   |   +---AngularJS-ui-router
|   |   +---bootstrap-sass-official
|   |   +---html5-boilerplate
|   |   +---jquery
|   |   \---ngstorage
|   +---components
|   |   \---version
|   +---node_modules
|   +---public
|   |   \---css
|   \---src
|       +---scripts
|       +---styles
|       +---views
+---e2e-tests
+---nbproject
|   \---private
+---node_modules
+---public
|   +---bower_components
|   +---scripts
|   +---styles
\---test
```

一些好的阅读参考资料：

- *AngularJS by Example*, Packt 出版社 (<https://www.packtpub.com/webdevelopment/angularjs-example>)
- *AngularSeed Project* (<https://github.com/angular/angular-seed>)
- *Angular UI* (<https://angular-ui.github.io/bootstrap/>)
- *Gulp* (<http://gulpjs.com/>)

小结

在本章中，我们学习了新的动态 web 应用程序开发。多年来，它已经完全改变了。web 应用程序前端是完全用纯 HTML 和 JavaScript 开发的，而不是使用 JSP、servlet、ASP 等任何服务器端技术开发的。使用 JavaScript UI 的应用程序开发现在有自己的开发环境，如 npm、bower 等。我们探讨了用来开发 web 应用程序的 AngularJS 框架。它通过提供对引导和处理 AJAX 调用的 \$https 服务的内置功能和支持，使得这项工作变得更容易。

希望你掌握 UI 开发概述、现代应用程序的开发，以及与服务器端微服务集成的方式。在下一章，我们将学习微服务设计的最佳做法和一般原则，将提供有关使用微服务开发的行业做法和范例的详细信息，它还将包含微服务实现在哪里出错了和如何才能避免这类问题的例子。

8

最佳做法和一般原则

在为获得微服务示例项目开发经验付出这么多辛苦的工作后，你一定会想如何避免常见的错误，并改进基于微服务的产品和服务的开发全过程。我们可以按照这些原则或准则，来简化微服务的开发过程，并避免或减少潜在的局限性。我们将在这一章着重介绍这些关键概念。

这一章分为以下三个部分：

- 概述和心态
- 最佳做法和原则
- 微服务框架和工具

概述和心态

无论对新的和现有的产品和服务，都可以实现基于微服务的设计。与从零开始开发和设计新的系统较容易，而对已经投入运行的系统做更改较难的观点相反，每种方法各有其自身的挑战和优势。

例如，由于新产品或服务没有现有的系统设计，在设计系统时有自由和灵活性，而无须对其影响给予任何考虑。但是，对于新的系统，没有明确的功能和系统需求，这些具备成熟形状需要时间。另一方面，对于成熟的产品和服务，对功能和系统需求有详细的知识和信息。不过，会遇到减轻设计变化的影响带来的风险的挑战。因此，在把整体式的生产系统更新到微服务的时候，比起从零开始构建一个系统，将需要更好的规划。

有经验的成功软件设计专家和架构师总是评估利弊，并对现存的系统做出任何改变采取审慎的态度。人们不应该因为可能会很酷或时尚而对现有系统设计进行更改。因此，如果想要把现有生产系统的设计更新为微服务，需要在做这个决定之前评估所有的利弊。

我相信整体系统为升级到一个成功的基于微服务的设计提供了很好的平台。很显然，在这里我们不讨论成本。现有系统和功能的足够知识，使你能够基于现有系统的功能和这些功能之间的交互方式对其进行划分，并在此基础上建立微服务。此外，如果整体式产品已经是以某种方式模块化的，那么通过公开的 API，而不是应用程序二进制接口（**Application Binary Interface, ABI**）直接转化微服务可能是最简单的实现微服务架构的方法。一个基于微服务的系统的成功更依赖于微服务及其交互协议而不是别的。

话虽如此，这并不意味着如果你从零开始就不能开发出一个成功的基于微服务的系统。然而，建议以整体设计为基础开始一个新的项目，它会为你提供功能与系统的角度和理解。它使你能够快速找到瓶颈，并指导你找出任何可以使用微服务开发的潜在功能。在这里，我们还没有讨论项目的大小，这是另一个重要因素。我们将在下一节讨论这个主题。

在当今的云时代和敏捷开发世界中，从做出任何改变到变化生效只需要用一个小时的时间。在今天的竞争环境中，每个组织都想具有迅速地向用户提供功能的优势。持续开发、集成和部署都是生产交付过程这个完全自动化过程的一部分。

如果你提供基于云环境的产品或服务，这么做就更有意义。然后，基于微服务的系统使团队能够灵活应对来解决任何问题，或向用户提供一个新的功能。

因此，在做出从头开始一个新的基于微服务的项目或计划把现有的整体系统的设计升级到一个基于微服务的系统的决定之前，需要评估所有利弊。必须倾听和理解团队中存在的不同想法和观点，并且需要采取审慎的态度。

最后，我想要分享具有一个更好的流程和高效的系统对于生产系统成功的重要性。在当今的时代，有一个基于微服务的系统并不能保证生产系统的成功，而独立应用程序并不意味着你不能有一个成功的生产系统。Netflix 这个基于微服务的云视频租赁服务和 Etsy 这个整体式电子商务平台，这两者都是成功的实际生产系统的例子（参见本章后面参考资料一节中一个有趣的 Twitter 讨论链接）。因此，流程和敏捷性也是生产系统成功的关键。

最佳做法和原则

正如我们从第一章学到的，微服务是实现面向服务的架构（Service Oriented Architecture, SOA）的轻量级的风格。最重要的是，微服务的定义是不严格的，这使你可以按想要的方式和根据需要灵活地开发微服务。同时，需要确保遵循几条标准的做法和原则，以使你的工作更容易一些，并成功实施基于微服务的架构。

Nanoservice（不推荐）、规模和整体性

在你的项目中，每个微服务都应体积微小并执行一种功能或特性（例如，用户管理），且足够独立地自主执行功能。

以下两条引文来自 Mike Gancarz(X windows 系统的设计成员)，它定义了 UNIX 哲学的首要戒律之一，它也适合微服务范式：

“小即是美。”

“让每个程序做好一件事。”

那么，在当今时代，当你有一个能减少代码行（lines of code, LOC）的框架（例如 Finagle）时，如何定义规模呢？此外，许多现代编程语言，如 Python 和 Erlang，都不那么烦琐。这使得是否要使此代码成为微服务变得难以决定。

显然，你可能会实现一个LOC很少的微服务，但它实际上不是微服务，而是nanoservice。

Arnon Rotem-Gal-Oz 对 nanoservice 的定义如下：

“Nanoservice 是一种反模式，它的服务的粒度太细。Nanoservice 是一个（通信、维护，等等）开销超过其用途的服务。”

因此，基于功能来设计微服务很有意义。领域驱动设计使得在领域级别定义功能变得很容易。

如前文所述，项目规模是决定是否为项目实施微服务，或确定你想要的微服务数量时的一个关键因素。在一个简单的小型项目中，使用整体式架构是有意义的。例如，基于我们在第 3 章中学到的领域设计，你会清楚地了解功能需求，了解可用来绘制边界之间各种

功能或特性的事实。例如，在我们已实现的示例项目(OTRS)中，假如你不想把 API 公开给客户，或不想把它作为 SaaS 使用，或者做决定之前有很多你想要评估的相似参数，使用整体设计来开发同一项目是很容易的。

可以在以后需要时把整体项目迁移到微服务设计。因此，至关重要的是，应该用模块化方式开发整体项目并在每个层之间都具有松耦合，并确保不同的功能和特性之间有预定的接触点和边界。此外，还应相应地设计你的数据源，如数据库。即使你不打算迁移到一个基于微服务的系统，它也将使 bug 修复和增强易于实现。

当你迁移到微服务时，注意上述各点将减轻你可能会遇到的任何困难。

一般来说，如前几章所述，开发大型或复杂的项目应该使用基于微服务的架构，因为它提供了许多优点。

尽管我建议把初始项目开发为整体式的，但一旦你更好地了解了项目功能和项目的复杂性，那么你就可将其迁移到微服务。理想情况下，一个已开发出的初始原型应该给你提供功能的边界，使你能够做出正确的选择。

持续集成和部署

必须具备一个持续的集成和部署过程，它给你更快提供更改和及早发现 bug 的好处。因此，每个服务都应该有其自身的集成和部署过程。此外，它必须被自动化。这个任务有很多工具可用，例如 Teamcity、Jenkins，等等，它们都得到广泛使用。它可以帮助你自动生成过程——及早发现构建的失败，尤其是当你将更改集成到主线时。

也可以将测试集成到每个自动的集成和部署过程中。**集成测试 (Integration Testing)** 负责测试系统不同部分的交互，如两个接口（API 提供者和使用者）之间，或系统不同的组件或模块，如 DAO 与数据库之间，等等。集成测试是重要的，因为它测试模块之间的接口。单个模块首先进行隔离测试，然后集成测试，以检查联合的行为并验证需求被正确实现。因此，在微服务中，集成测试是一个验证 API 的关键工具。在下一节，我们将介绍更多关于它的内容。

最后，在此过程在其中部署构建完成的软件的 DIT 计算机上，可以看到更新主线发生更改。

这个过程不会在这里结束，可以制作一个容器，如 docker，并把它交给 WebOps 团队，或拥有一个单独的进程，将它传递到已配置的位置，或将它部署到 WebOps 阶段性环境。从这里，一旦由指定的主管部门批准，它就可以被直接部署到生产系统上。

系统/端到端测试自动化

测试是交付任何产品和服务的重要部分。你不想要交付给客户的应用程序有很多错误。过去，在瀑布模型很流行的时候，一个组织经常在交付给客户之前采取一至六个月或更长时期的测试阶段。近年来，在敏捷过程变得流行后，自动化得到更多重视。类似于前面的测试，自动化也是强制性的。

无论你是否遵循测试驱动开发(**Test Driven Development, TDD**)，我们都必须具备系统或端到端的自动化测试。对于测试你的业务场景，它是非常重要的，对于可能开始从你的 REST 调用数据库检查，或从 UI 应用程序到数据库检查的端到端测试，也是如此。

此外，如果你有公共 API，测试 API 也是很重要的。

这样做会确保任何更改都不会破坏任何功能，并确保交付无缝、无缺陷的产品。如上节所述，每个模块都使用单元测试进行单独测试，以检查一切都在按预期运行，然后，无论实现得正确与否，集成测试都在不同的模块之间执行，以检查预期的联合行为并校验需求。在集成测试后，还会执行验证功能和功能需求的功能测试。

所以，如果单元测试确保单个模块独立工作情况良好，集成测试确保不同模块之间的交互均按预期方式工作。如果单元测试工作情况良好，它意味着集成测试失败的概率大大降低。同样，集成测试确保功能测试很有可能获得成功。

[ 据推测，人们始终会保持所有类型的测试更新，无论这些是单元级测试还是端到端测试场景。]

自我监控和记录

微服务应当提供服务本身的信息和它所依赖的各种资源的状态。服务信息包括处理一个请求的平均、最小和最大时间，成功和失败的请求的数量，能够跟踪请求、内存使用情况等的统计信息。

Adrian Cockcroft 在 2015 年的 **Glue 会议 (Glue Conference, Glue Con)** 上强调的下面几个做法对于监控微服务是非常重要的。它们中的大部分对任何监控系统都有效：

- 分析指标含义的代码，要比收集、移动、存储和显示度量指标的代码花更多时间来开发。
这不仅有助于提高生产效率，而且还提供了重要参数来微调微服务和提高系统效率。这个观点是开发更多的分析工具，而不是开发更多的监控工具。
- 用来显示延迟的指标需要小于人类的注意力跨度。根据 Adrian 的说法，这意味着少于 10 秒。
- 验证你的测量系统具有足够的准确度与精度。收集响应时间的直方图。
- 准确的数据能使决策更快，并允许你微调直到所需的精度水平。他还建议最好用的显示响应时间的图形是直方图。
- 监控系统需要比被监控的系统有更高的可用性和可扩展性。
- 总而言之，你不能依靠本身并不稳定或 24/7 可用的系统。
- 针对分布式、短暂性、云本机、容器化的微服务进行优化。
- 用模型来拟合指标以理解相互关系。

监控是微服务架构的关键组成部分。根据项目的大小，你可能有十几个到数以千计的微服务（对于大企业的大项目）。即使对于扩展性和高可用性，组织为每个微服务创建集群或负载均衡池/仓，甚至基于每个版本的微服务创建单独的池。最终，它增加了你需要监控的资源，包括微服务的每个实例。此外，重要的一点是，你应该具备一个进程，每当有什么闪失，你就立即知道它，或更好的是，在出了什么差错前前提到一个警报通知。因此，有效且高效的监控对构建和使用微服务架构至关重要。Netflix 使用情况安全监控使用诸如 Netflix Atlas（实时业务监控，处理 12 亿指标）、Security Monkey（用于监控基于 AWS 的环境的安全）、Scumblr（情报收集工具）和 FIDO（对事件的分析和自动化事件报告）的工具。

日志记录是微服务不应被忽视的另一个重要方面。具有有效的日志记录关系重大。因为可能有 10 个或更多的微服务，所以管理日志记录是一项艰巨的任务。

对于我们的示例项目，我们用 MDC 日志记录，在某种程度上，对单个微服务日志记录，这就足够了。然而，我们还需要对一个完整的系统的日志记录或集中式日志记录。我们还需要日志的汇总统计数据。做这项工作的工具，包括 Loggly 和 Logspout。



请求和生成的相关事件提供了请求的总体视角。为追踪任何事件和请求，分别用服务 ID 和请求 ID 来关联事件和请求是至关重要的。也可以将事件的内容，例如消息、严重程度、类名等关联到服务 ID。

每个微服务都使用独立的数据存储区

如果你还记得，你可以发现有关微服务的最重要特征是，微服务与其他微服务隔离的运行方式，通常是作为独立应用程序运行的。

遵守这项规则，建议你不要跨多个微服务使用相同的数据库或任何其他数据存储区。在大型项目中，你可能会有不同的团队在同一个项目中工作，并且你想要能够灵活地为每个微服务选择最适合此微服务的数据库。

现在，这也带来了一些挑战。

例如，对可能为同一项目中的不同微服务工作的团队，如果此项目共享相同的数据库结构，会产生以下问题：存在一个微服务的变化可能会影响其他微服务模型的可能性。在这种情况下，一个微服务的变化可能影响依赖于它的微服务，所以你也需要改变依赖模型的结构。

要解决此问题，微服务应该基于 API 驱动的平台开发。每个微服务都将公开其 API，它们可以由其他微服务使用。因此，你也需要开发 API，这是集成不同的微服务的需要。

同样，由于有不同的数据存储区，实际项目数据也分散在多个数据存储区，这使数据管理更复杂，因为单独的存储系统更容易变得不同步或变得不一致，并且外键会发生意外的变化。要解决这种问题，你需要使用主数据管理（**Master Data Management, MDM**）工具。MDM 工具在后台运行，并且如果它发现任何不一致，就修复它们。对于 OTRS 示例，它可能会检查存储预订请求 ID 的每个数据库，以验证相同的 ID 存在于所有的数据库中（换句话说，在任何一个数据库中都不存在任何丢失的 ID 或额外的 ID）。市场上的 MDM 工具包括 Informatica、IBM MDM 高级版、Oracle Siebel UCM、Postgres（主数据流复制）、mariadb（主/主配置），等等。

如果没有满足你的要求的现有产品，或者你对任何专有的产品都不感兴趣，那么你可

以编写你自己的 MDM 工具。目前，API 驱动的开发和平台减少这些问题的复杂性，因此，至关重要的是，微服务应在 API 平台上开发。

事务边界

我们已经在第 3 章学过了领域驱动设计的概念。如果你还没有彻底掌握这些概念，请复习它们，因为这使你垂直地理解状态。因为我们专注于基于微服务的设计，结果是，我们有一个系统的系统，其中每个微服务都表示一个系统。在这种环境中，找到任何给定时刻的整个系统的状态是非常具有挑战性的。如果你熟悉分布式应用程序，可能会很好地适应在这种环境中的状态。

具有描述在任何给定时间哪个微服务拥有一条消息的事务边界是非常重要的。你需要能参与事务的方式或者过程、事务化的路由和错误处理程序、等效使用者和补偿操作。确保跨异构系统的事务性行为不是件容易的事，但有工具可为你做这个工作。

例如，Camel 有很强的事务性功能，可帮助开发人员轻松地创建具有事务性行为的服务。

微服务框架和工具

不去重新发明轮子总是更好的。因此，我们想探讨都有哪些已经可用的工具，能提供使微服务的开发和部署更容易的平台、框架和功能。

在本书中，我们已经广泛地使用了 Spring Cloud，由于同样的原因，它提供使微服务非常容易地开发所需的所有工具和平台。Spring Cloud 使用 Netflix 开放源码软件（OSS）。让我们探讨 Netflix OSS——一个完整的软件包。

我还补充了每个工具将如何有助于建立良好的微服务架构的简要概述。

Netflix 开放源码软件（OSS）

Netflix 开放源码软件中心是基于 Java 的微服务开放源码项目最流行和最广泛使用的开放源码软件。世界上最成功的视频租赁服务依赖于它。Netflix 已经有超过 4 000 万用户，他们在全球各地使用其服务。Netflix 是一个纯粹的基于云平台的解决方案，在微服务架构

的基础上开发。可以说，每当有人谈到微服务时，Netflix 都是进入你脑海的第一个名字。让我们讨论它提供的各种工具。在开发示例 OTRS 应用程序时，我们已经讨论了其中的很多工具。然而，有几个工具我们还未探讨过。在这里，我们将只对每个工具进行概述，而不是详细讲解。这将给你带来微服务架构的实际特点和它在云平台中使用的总体思路。

构建——Nebula

Netflix Nebula 是一种使你更容易使用 Gradle（类似 Maven 的构建工具）来生成微服务的 Gradle 插件集合。对于我们的示例项目，由于我们已使用了 Maven，因此我们没有机会在本书中详细探讨 Nebula。然而，研究它会很有趣。对于开发人员来说，最重要的 Nebula 功能是消除 Gradle 生成文件中的许多样板代码，这使得开发人员能够把重点放在编码上面。



有一个很好的构建环境，尤其是 CI/CD（持续集成和持续部署）是微服务开发和与敏捷开发保持一致必备的。Netflix Nebula 使你的构建过程更轻松、更高效。

部署和交付——Spinnaker 与 Aminator

一旦你生成的软件已准备就绪，你会想要将此软件移动到亚马逊网络服务（Amazon Web Services, AWS）EC2 中。Aminator 使用亚马逊机器映像（Amazon Machine Image, AMI）的形式来创建生成的软件并将其打包成映像文件。Spinnaker 然后将这些 AMI 部署到 AWS。

Spinnaker 是高速并高效地发布代码更改的持续交付平台。Spinnaker 还支持其他云服务，例如 Google Computer Engine 和 Cloud Foundry。



你想要将最新的微服务软件部署于类似 EC2 的云环境中，Spinnaker 和 Aminator 可以帮助你自动地完成这件事。

服务注册和发现——Eureka

正如我们已在本书中探讨的，Eureka 提供了负责微服务注册和发现的服务。最重要的是，Eureka 也用于中间层（承载不同的微服务的进程）负载均衡。Netflix 也使用 Eureka 以

及其他工具，像 Cassandra 或 memcached，以提高其整体可用性。



服务注册和发现是微服务架构所必备的。Eureka 的用途就是这个。

请参阅第 4 章获取有关 Eureka 的详细信息。

服务沟通——Ribbon

如果没有进程间或服务间的通信，微服务架构就没有用。功能区应用程序提供该功能。Ribbon 与 Eureka 结合实现负载均衡，与 Hystrix 结合实现容错或电路断路器操作。

除 HTTP 之外，Ribbon 还支持 TCP 和 UDP 协议。它对这些协议同时提供了异步和反应式模型的支持，它还提供缓存和批处理的功能。



因为你的项目中将会有很多微服务，你需要一种使用进程间或服务间通信的方法来处理信息。Netflix 公司为此提供了 Ribbon 工具。

电路断路器——Hystrix

Hystrix 工具用来执行电路断路器操作，也就是，容忍延迟和容错。因此，Hystrix 会停止连锁故障。Hystrix 执行实时的服务监控和属性更改操作，并支持并发。

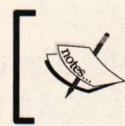


电路断路器或容错，是用于任何项目，包括微服务的一个重要概念。一个微服务的故障不应停止你的整个系统；Netflix Hystrix 的任务是防止这一点，并在出故障时，向用户提供有意义的信息。

边缘（代理）服务器——Zuul

Zuul 是边缘服务器或代理服务器，它用来为外部应用程序如 UI 客户端、Android/iOS 应用程序或任何产品或服务提供的第三方使用者的 API 发出的请求提供服务。从概念上讲，它是一扇面向外部应用程序的门。

Zuul 允许动态路由和监控请求。它还执行安全操作，如身份验证。可以确定每个资源的身份验证要求，并拒绝任何不能满足这些要求的请求。



你需要为微服务提供边缘服务器或 API 网关。Netflix Zuul 提供此功能，请参阅第 5 章获取详细信息。

业务监控——Atlas

1

的候选者。如果任何规则确定某资源是被清理的候选者，看守猴子就对此资源做标记，并安排时间去清理它。特殊情况下，当你想要把未使用的资源保留更长的时间，在看守猴子删除资源前，资源的所有者将在清理时间前几天收到通知，天数是可配置的。

- **符合猴子 (Conformity Monkey)**：是一种在 AWS 云中运行的服务，它寻找不符合最佳做法的预定义规则的实例。它可以扩展来用于其他云提供商和云资源。这个服务的时间表是可配置的。

如果确定该实例不符合任何一条规则，猴子就向实例的所有者发送电子邮件通知。可能在有的例外情况下，对于某些应用程序要忽略关于符合特定规则的警告。

- **安全猴子 (Security Monkey)**：安全猴子监控策略的更改并对某个 AWS 账户上没有安全感的配置进行提醒。安全猴子的主要目的是保证安全性，但它也是用于跟踪潜在问题的有用工具，因为它本质上是一个更改跟踪系统。
- 成功的微服务架构可以确保你的系统始终是运行的，并且单个云组件失败不会停止整个系统。Simian Army 使用许多服务来实现高可用性。

AWS 资源监控——Edda

在云环境中，没有什么是静态的。例如，虚拟宿主机实例经常发生变化，通常情况下，IP 地址可以由各种应用程序重复使用，防火墙或相关的变化也可能发生。

Edda 是跟踪这些动态的 AWS 资源的服务。Netflix 将其命名为 Edda(即北欧神话故事)，它记录云管理和部署的故事。Edda 使用 AWS API 轮询 AWS 资源并记录结果。这些记录允许搜索和查看云已经随着时间的推移发生了哪些变化。例如，如果任何 API 服务器的主机正在造成任何问题，你需要找出此主机是什么，哪支团队要为它负责。

它提供了这些功能：

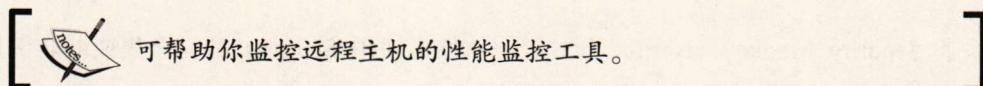
- **动态查询**：Edda 提供 REST API，并且它支持矩阵参数并提供让你仅检索所需的数据的字段选择器。
- **历史的变化**：Edda 维护所有 AWS 资源的历史记录。此信息可帮助你分析资源中断的原因和影响。Edda 还可以提供有关资源的当前和历史信息的不同视图。在撰写本文时，它在 MongoDB 中存储信息。

- **配置：**Edda 支持多个配置选项。一般情况下，可以从多个账户和多个区域轮询信息，还可以使用账户的组合和这些账户指向的区域。同样的，它提供 AWS、Crawler、Elector 和 MongoDB 的不同配置。
- 如果采用 AWS 来承载基于微服务的产品，那么 Edda 可用于对 AWS 资源进行监控。

主机性能监控——Vector

Vector 是一个静态的 web 应用程序，在 web 浏览器内运行。它可以用来监控安装了 Performance Co-Pilot(PCP)的主机的性能。Vector 支持 PCP 3.10 及以上版本。PCP 收集各种指标并提供给 Vector。

它根据需要提供高分辨率的正确指标。这可以帮助工程师了解系统的行为和正确地解决性能问题。



可帮助你监控远程主机的性能监控工具。

分布式配置管理——Archaius

Archaius 是一个分布式的配置管理工具，它允许你执行以下操作：

- 使用动态和类型化的属性。
- 执行线程安全的配置操作。
- 使用轮询框架检查属性更改。
- 在有序的层次结构的配置中使用回调机制。
- 使用 JConsole 检查属性并对其执行操作，因为 Archaius 提供了 JMX MBean。
- 当你有一个基于微服务的产品时，需要有一个良好的配置管理工具。Archaius 可以帮助在一个分布式的环境中配置不同类型的属性。

Apache Mesos 调度器——Fenzo

Fenzo 是用 Java 编写的用于 Apache Mesos 框架的一个调度程序库。Apache Mesos 框架查找匹配的资源，并将其分配到挂起的任务上。其主要特点如下：

- 支持长时间运行的服务风格的任务和批处理。
- 可以基于资源需求自动缩放执行主机集群。
- 支持插件，可以基于需求创建它们。
- 可以监控资源分配的故障，允许调试故障根源。

成本和云利用率——Ice

Ice 从成本和使用的角度提供云资源的全景图。它提供调配云资源分配到不同团队的最新信息，为云计算资源的最优利用增加价值。

Ice 是一个圣杯项目。用户与 Ice UI 组件交互，后者显示通过 Ice 阅读器组件发送的信息。阅读器从 Ice 处理器组件所生成的数据中提取信息。Ice 处理器组件从详细的云计算费文件中读取数据信息，并将它转换成 Ice 阅读器组件可读的数据。

其他安全工具——Scumblr 和 FIDO

除了 **Security Monkey**，**Netflix** 开放源码软件也使用 **Scumblr** 和完全集成的防御操作（**Fully Integrated Defense Operation, FIDO**）工具。



为了跟踪你的微服务，并保护它不受经常的威胁和攻击，你需要以自动化的方式来对你的微服务进行保护和监控。**Netflix Scumblr** 和 **FIDO** 为你做这份工作。

Scumblr

Scumblr 是一个基于 Ruby on Rails 的 web 应用程序，它允许你执行定期搜索并对识别的结果执行存储/采取行动。基本上，它会利用全互联网有针对性的搜索来收集情报，从而揭露特定安全问题用于调查。

Scumblr 利用可以流程化的宝贵信息，允许对不同类型的结果设置灵活的工作流。**Scumblr** 利用称为 **Search Providers**（搜索提供程序）的插件进行搜索，它会检查类似以下的异常。因为它是可扩展的，可以根据需要添加任意多的检查项目：

- 泄露的凭据

- 黑客漏洞/讨论
- 攻击讨论
- 社交媒体上的安全相关讨论

完全集成的防御操作（FIDO）

FIDO 是一种安全业务流程框架，用于分析事件和自动化事件响应。它通过评价、评估和应对恶意软件来使事件的响应过程变得自动化。FIDO 的主要目的是处理评估来自当今安全栈的威胁和它们所生成的大量警报所需要的大量手动工作。

作为业务流程平台，FIDO 通过大幅减少检测、通知和应对网络攻击所需要的手动工作，可以更高效、更准确地使用现有的安全工具。有关详细信息，可以参考下面的链接：

<https://github.com/Netflix/Fido> <https://github.com/Netflix>

参考资料

- 关于整体式(Etsy)与微服务(Netflix)的比较的 Twitter 讨论，<https://twitter.com/adrianco/status/441169921863860225>
- *Monitoring Microservice and Containers Presentation* (监控微服务和容器演示文稿)
Adrian Cockcroft 编：<http://www.slideshare.net/adriancockcroft/gluecon-monitoring-microservices-and-containers-a-challenge>
- Nanoservice 反模式：<http://arnon.me/2014/03/microservices-nanoservices/>
- 微服务的架构的 Apache Camel：<https://www.javacodegeeks.com/2014/09/apache-camel-for-microservices.html>
- Teamcity：<https://www.jetbrains.com/teamcity/>
- Jenkins：<https://jenkins-ci.org/>
- Loggly：<https://www.loggly.com/>

小结

在这一章，我们探讨了最适合基于微服务的产品和服务的各种做法和原则。微服务架构是云环境的结果，它被广泛用于与基于整体式系统的对比。我们发现了少量与规模、敏捷性和测试有关的原则，它们必须具备才能成功实现微服务。

我们也概述了由 Netflix OSS 使用的不同工具，它们用于成功实现微服务架构产品和服务所需的各种关键功能。Netflix 成功地使用同样的工具提供视频租赁服务。

在下一章，读者可能会遇到各种问题，他们可能会被这些问题困住。下一章解释了在开发微服务的过程中所遇到的常见问题和它们的解决办法。

9

故障排除指南

到目前为止，我们已经学了这么多东西，我敢肯定你享受这个具有挑战性的快乐学习旅程的每时每刻。学完这一章后，我不愿意说这本书结束了，而宁愿说你正在完成第一个里程碑。跨过这个里程碑，我们就可以继续学习基于微服务的新设计范式并在云环境中实现它。我想重申，集成测试是测试微服务和 API 之间交互的重要途径。在你完成在线餐馆订座系统（OTRS）示例应用程序的过程中，我确信你面临许多挑战，尤其是在调试应用程序时。在这里，我们将介绍几种做法和工具，帮助你解决部署应用程序、Docker 容器和宿主机的故障。

本章包括以下三个主题：

- 日志记录和 ELK 环境
- 使用相关 ID 来进行服务调用
- 依赖项和版本

日志记录和 ELK 环境

你能想象在没有看到日志的情况下在生产系统上调试任何问题吗？简单地说，不行，因为时间是很难回去的。因此，我们需要日志记录。如果它们是按这种方式设计和编码的，日志也给我们提供有关系统的警告信号。日志记录和日志分析，对排除任何问题的故障和吞吐量、容量和系统的健康状况监控，都是一个重要的步骤。因此，有一种很好的日志平台和战略将使调试变得高效。日志是在软件开发最初的几天中最重要的关键部件之一。

微服务一般使用映像的容器，如 Docker 来部署，它们提供日志，并通过命令来帮助你读取部署在容器内的服务日志。Docker 和 Docker Compose 提供把容器内运行的、在所有容器中的服务日志分别输出的命令。请参阅以下 Docker 和 Docker Compose 的 logs 命令：

Docker logs 命令：

用法： docker logs [OPTIONS] < CONTAINER NAME >

获取容器的日志：

-f, --follow	跟踪日志输出
--help	打印用法
--since=""	显示时间截以来的日志
-t, --timestamps	显示时间截
timestamp	
--tail="all"	要从日志的末尾显示的行数



Docker Compose logs 日志命令：

用法： docker-compose logs [OPTIONS] [SERVICE...]

OPTIONS：

--no-color	产生单色输出
-f, --follow	跟踪日志输出
-t, --timestamps	显示时间截
--tail	要从每个容器日志的末尾显示的行数
[SERVICE...]	表示容器的服务——可以给出多个

这些命令帮助你检查微服务和其他容器内运行的进程的日志。正如你所看到的，当你有大量的服务时，使用上面的命令将具有很大的挑战性。例如，如果你有数十个或数百个微服务，将很难跟踪每个微服务的日志。同样，你可以想象，即使没有容器，监控单独的日志记录将是多么困难。因此，你可以想象，检查和关联数十到数百个容器的日志的难度有多大。它非常耗时，并带来很少的价值。

因此，我们将使用一种类似 ELK 环境的日志聚合器和可视化工具来辅助，它将用于集中日志记录。下一节，我们将探讨这个工具。

简要概述

Elasticsearch、Logstash、Kibana（ELK） 环境是一系列执行日志聚合、分析、可视化和监控的工具。ELK 环境提供完整日志记录平台，用来分析、可视化和监控所有日志，包括所有类型的产品日志和系统日志。如果你已经了解 ELK 环境，请跳过下一节。在这里，我们会提供 ELK 环境中的每个工具的简介。

Elasticsearch

Elasticsearch 是最受欢迎的企业全文搜索引擎之一。它是开放源码的软件，它是可分布式的，并支持多租户。一个单独的 Elasticsearch 服务器存储多个索引（每个索引都表示一个数据库），并且单个查询可以搜索多个索引的数据。它是一个分布式的搜索引擎并支持聚类。

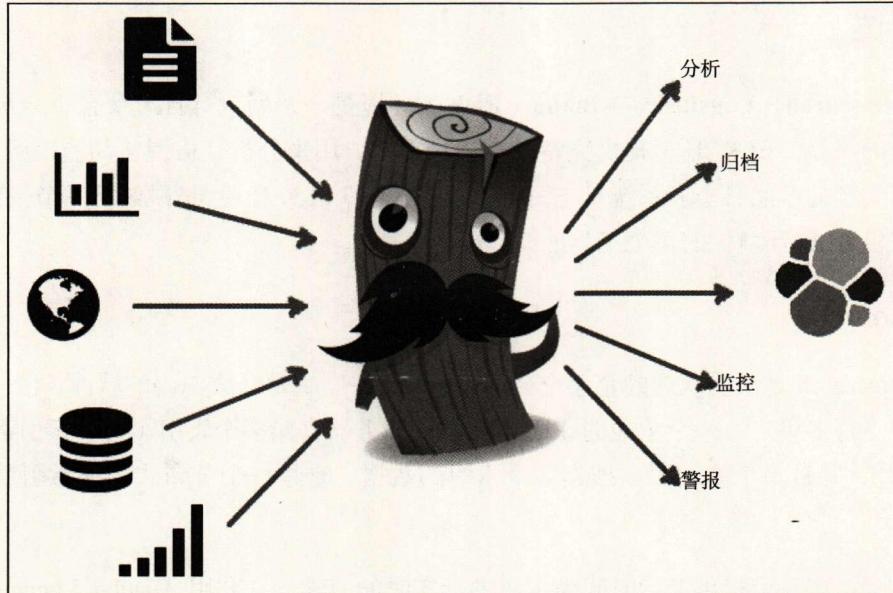
它易于扩展并可提供滞后时间为 1 秒的近实时的搜索。它利用 Apache Lucene 和 Java 开发。Apache Lucene 也是免费的，开放源码，它提供了 Elasticsearch 的核心，又名信息检索软件库。

Elasticsearch API 本质上广泛且非常精细。Elasticsearch 提供了一个基于 JSON 的架构，更少的存储，并用 JSON 表示数据模型。Elasticsearch API 使用 JSON 文档处理 HTTP 请求和响应。

Logstash

Logstash 是具有实时管道能力的开放源码信息收集引擎。简单地说，它收集、分析、处理并存储数据。由于 Logstash 具有数据管道功能，它能帮助你处理任何来自各种系统的事件数据，如日志。Logstash 作为代理来运行，它收集数据，解析、过滤它们，并将其输出发送到一个指定的应用程序，如 Elasticsearch 或在控制台上的简单的标准输出。

它也有一个很好的插件生态系统（图像来自 www.elastic.co）：



Logstash生态系统

Kibana

Kibana 是一个开源分析和可视化的 web 应用程序，它被设计为与 Elasticsearch 配合工作。Kibana 用于搜索、查看和与存储在 Elasticsearch 索引中的数据进行交互。

它是一个基于浏览器的 web 应用程序，可以用各种图表、表和地图执行高级的数据分析和可视化数据。此外，它是一个零配置的应用程序。因此，安装它以后不需要任何编码或额外的基础设施。

ELK 环境安装

一般来说，这些工具都是单独安装的，然后配置为相互通信。这些组件的安装是相当简单的。从指定的位置下载可安装的工件，并按照下一节所示的步骤安装。

下面提供的安装步骤是设置你想要运行的 ELK 环境所需的基本安装程序的一部分。由于此安装在我的本地主机上完成，我已用了主机 localhost。它可以容易地改为任何你想要的单独的主机名。

安装 Elasticsearch

我们可以按照如下步骤来安装 Elasticsearch：

1. 从 <https://www.elastic.co/downloads/elasticsearch> 下载最新的 Elasticsearch 软件。
2. 将其解压缩到你的系统中的所需位置。
3. 确保安装了最新的 Java 版本和设置了 `JAVA_HOME` 环境变量。
4. 转到 Elasticsearch 主目录，在基于 Unix 的系统上，运行 `bin/elasticsearch`，而在 Windows 上，运行 `bin/elasticsearch.bat`。
5. 打开任何浏览器，并输入 `http://localhost:9200/`。成功安装后应该会提供一个类似于如下所示的 JSON 对象：

```
{
  "name" : "Leech",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "2.3.1",
    "build_hash" : "bd980929010aef404e7cb0843e61d0665269fc39",
    "build_timestamp" : "2016-04-04T12:25:05Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

默认情况下，不安装 GUI。可以通过从 `bin` 目录执行下面的命令安装一个 GUI。
请确保系统连接到互联网：

```
plugin -install mobz/elasticsearch-head
```

6. 现在，可以使用 URL `http://localhost:9200/_plugin/head/` 来访问 GUI 界面。

可以把 `localhost` 和 `9200` 替换为你各自的主机名和端口号。

安装 Logstash

我们可以按给定的步骤安装 Logstash：

1. 从 <https://www.elastic.co/downloads/logstash> 下载最新的 Logstash 软件。
2. 将其解压缩到你的系统中的所需位置。

编写一个配置文件，如下所示。它指示 Logstash 从给定文件读取输入并将其传递给 Elasticsearch（见以下 config 文件；Elasticsearch 是用 localhost 和 9200 端口表示的）。它是最简单的配置文件。要添加过滤器，并了解更多关于 Logstash 的信息，你可以查看 Logstash 参考文档，它位于 <https://www.elastic.co/guide/en/logstash/current/index.html>。



正如你可以看到，OTRS service 日志和 edge-server 日志被作为输入添加。同样，你还可以添加其他的微服务日志文件。

```
input {  
    #### OTRS ####  
    file {  
        path => "\logs\otrs-service.log"  
        type => "otrs-api"  
        codec => "json"  
        start_position => "beginning"  
    }  
  
    #### 边缘 ####  
    file {  
        path => "/logs/edge-server.log"  
        type => "edge-server"  
        codec => "json"  
    }  
}  
  
output {
```

```

    stdout {
      codec => rubydebug
    }
    elasticsearch {
      hosts => "localhost:9200"
    }
}

```

3. 转到 Logstash 主目录，在基于 Unix 的系统上，运行 bin/logstash agent -f logstash.conf，而在 Windows 上，运行 bin/logstash.bat agent -f logstash.conf。在这里，Logstash 使用 agent 命令执行。Logstash 代理从配置文件输入字段中提供的来源收集数据，并将其输出发送到 Elasticsearch。在这里，我们不使用过滤器，否则它可能会在把输入的数据提供给 Elasticsearch 之前对它做处理。

安装 Kibana

我们可以通过下面给定的步骤安装 Kibana web 应用程序：

1. 从 <https://www.elastic.co/downloads/kibana> 下载最新 Kibana 软件。
2. 将其解压缩到你的系统中的所需位置。
3. 从 Kibana 主目录打开配置文件 config/kibana.yml，并把 elasticsearch.url 指向以前配置的 Elasticsearch 实例：

```
elasticsearch.url:"http://localhost:9200"
```

4. 转到 Kibana 主目录，在基于 Unix 的系统上，运行 bin/kibana agent -f logstash.conf；而在 Windows 上，运行 bin/kibana.bat agent -f logstash.conf。
5. 现在可以从你的浏览器使用 URL <http://localhost:5601> 访问 Kibana 应用程序。

为了解 Kibana 的更多内容，查看在 <https://www.elastic.co/guide/en/kibana/current/getting-started.html> 的 Kibana 参考文档。

在我们按照上述步骤操作时，你可能会注意到，这需要一定的工作量。如果你想要避免手动安装，可以 Docker 化它。如果你不想要投入精力创建 ELK 环境的 Docker 容器，可以从 Docker Hub 选择一个。在 Docker Hub 有许多现成的 ELK 环境的 Docker 映像，可以尝试不同的 ELK 容器并选择最适合的。`willdurand/elk` 是被下载得最多的容器，也容易启动，并良好地与 Docker Compose 配合工作。

有关 ELK 环境实现的提示

- 为了避免任何的数据丢失和处理突然激增的输入负载，建议在 Logstash 和 Elasticsearch 之间使用代理，如 Redis 或 RabbitMQ。
- 如果你正在使用集群，防止脑裂问题，请使用奇数个 Elasticsearch 的节点数。
- 在 Elasticsearch 中，对于给定的数据始终使用合适的字段类型。这将允许你执行不同的检查，例如，`int` 字段类型将允许你执行 (`"http_status: < 400"`) 或 (`"http_status:=200"`) 检查。同样，其他字段类型还允许你执行类似的检查。

服务调用关联 ID 的使用

当你对任何 REST 端点发出调用并有任何问题弹出时，很难跟踪问题和它的起源，因为向服务器发出每个调用时，此调用可能调用另一个，以此类推。这使得很难弄清楚一个特定的请求如何被变换，以及它调用了什么。通常情况下，由一个服务造成的问题可能会造成其他位置的服务出问题。这个问题很难跟踪，可能需要大量的工作。如果它是整体式的，你会知道正确的调查方向，但微服务使得难以理解问题的根源是什么和你应该在哪里得到你的数据。

让我们看看怎样解决这个问题

利用跨所有调用传递的关联 ID，能够跟踪每个请求，并轻松地跟踪路由，每个请求都具有其独特的关联 ID。因此，当我们调试任何问题时，关联 ID 都是我们的出发点。我们可以跟着它一路跟踪，就可以找出到底什么东西出错了。关联 ID 需要一些额外的开发工作，但这个工作非常值得做，从长远来看将有很大帮助。当一个请求在不同的微服务之间传输时，你将能够看到所有交互和哪个服务有问题。

这不是新东西或微服务的发明。许多受欢迎的产品，如 Microsoft SharePoint 都已经采用了这种模式。

依赖项和版本

在产品开发中，我们面临的两个常见的问题是循环依赖和 API 版本。我们将在微服务的基础架构中讨论它们。

循环依赖关系及其影响

一般来说，整体式架构具有典型的分层模型，而微服务带有图状模型。因此，微服务可能会有循环依赖。

因此，有必要对微服务的关系保持一个依赖项检查。

让我们看看以下两种情况：

- 如果你的微服务之间有一个循环依赖，当某一具体事务可能会被困在一个循环中时，你很容易遇到分布式的堆栈溢出错误。例如，当餐馆的餐桌由一个人预订时。在这种情况下，餐馆需要知道预订的人（`findBookedUser`），而在给定的时间，人也需要了解餐馆（`findBookedRestaurant`）。如果这些服务的设计不好，就可能在循环中互相调用。结果可能是由 JVM 生成的堆栈溢出。
- 如果两个服务共享一个依赖项，而你更新另一个服务的 API 的方式可能会影响它们，你会需要一次更新所有三个服务。这就引出了一些问题，如你应该首先更新哪个服务？此外，你怎么使这个成为一个安全事务？

设计系统时需要分析它

因此，在设计微服务时，需要在不同的服务间建立正确的关系，以避免任何循环依赖是非常重要的。它是一个设计问题，并且必须得到解决，即使这需要重构代码。

维护不同版本

当你有更多的服务时，这意味着每个服务有不同的发布周期，这将通过不同版本服务

的引入增加复杂性，相同的 REST 服务将有不同的版本。当某个问题在一个版本中已经消失并在一个新版本中复现的时候，再现问题的解决办法将是很难的。

让我们了解更多

API 版本控制之所以重要是因为随着时间的推移 API 会改变。随着时间的推移，你的知识和经验增加，并导致 API 的改变。改变的 API 可能会破坏现有客户端集成。

因此存在管理 API 版本的各种方法。其中一种方法是在路径中使用版本，就像我们在本书中已经使用的那样，还有一些则使用 HTTP 标头。HTTP 标头可以是自定义请求标头，或者你可以使用 Accept Header 表示调用 API 的版本。更多关于如何使用 HTTP 标头来处理版本的信息，请参阅 Packt 出版的 Bhakti Mehta 编写的《*RESTful Java Patterns and Best Practices*》一书：<https://www.packtpub.com/application-development/restful-java-patterns-and-best-practices>。

在实现你的微服务时，在对任何问题进行故障排除时，在日志中产生版本号是非常重要的。此外，理想情况下，应该避免你的任何微服务的任何实例有太多版本。

参考资料

如下链接将有更多的信息：

- Elasticsearch: <https://www.elastic.co/products/elasticsearch>
- Logstash: <https://www.elastic.co/products/logstash>
- Kibana: <https://www.elastic.co/products/kibana>
- willdurand/elk: ELK Docker 映像
- *Mastering Elasticsearch-Second Edition*: <https://www.packtpub.com/webdevelopment/mastering-elasticsearch-second-edition>

小结

在本章中，我们探讨了 ELK 环境的概念和安装。在 ELK 环境中，Elasticsearch 用来存放日志和来自 Kibana 的服务查询。Logstash 是一个希望从中收集来自每个服务器的运行日

志的代理。Logstash 可以读取日志、过滤/转换它们，并提供给 Elasticsearch。Kibana 读取/查询来自 Elasticsearch 的数据并以表格或图形的可视化效果呈现。

我们也了解了调试问题时拥有关联 ID 的实用性。在这一章的结束，我们也探讨了几种微服务设计的缺点。在本书中包括有关微服务的所有话题是具有挑战性的，所以我试图尽可能在相关章节中包含参考资料，使你可以探索更多的相关信息。现在我想让你开始在工作场所或在个人项目中实现我们在这一章已经学到的概念。这不但会给你亲身的体验，而且也可以让你掌握微服务。此外，学会了这些内容，你就会有能力参与当地的聚会和会议。

