第 15 章　创建 RESTFul 风格的 Web 应用

在日常生活中，有些词汇在不同的环境中会有不用的含义。例如“唱歌”这个词，一般指敞开喉咙歌唱。但是，在某些情况下，“唱歌”还有其他的含义。例如有的导游为了让自己的语言显得更优雅，会和他的游客约定，如果在旅途中让游客下车去“唱歌”，是指去洗手间。

REST（Representational State Transfer，表述性状态转义）也是按照一词多义的思路，把在一种软件场景中已经存在的术语和概念运用到另一种软件场景中，并且给这些术语和概念赋予新的含义。本章介绍把 RESTFul 风格运用到 Spring MVC 框架中，对客户端的 HTTP 请求的请求方式和 URL 赋予了一些新的含义，使得 HTTP 请求在请求访问服务器端的数据库资源时，变得更加优雅规范。

本章范例是第 13 章的范例的更新版本，原版本和新版本的业务逻辑服务层和 DAO 层基本相同，主要是重新创建了 CustomerController 控制器类和视图层的 JSP 文件 hello.jsp。

15.1　RESTFul 风格的 HTTP 请求

根据 HTTP 协议的规定，客户端发出的 HTTP 请求可以使用多种请求方式，主要包括：

- GET：这种请求方式最为常见，客户端通过这种请求方式访问服务器上的一个文档，服务器把文档发送给客户端。
- POST：客户端可通过这种方式发送大量信息给服务器。在 HTTP 请求中除了包含要访问的文档的 URL，还包括大量的请求正文，这些请求正文中通常会包含 HTML 表单数据。
- PUT：客户端通过这种方式把文档上传给服务器。
- DELETE：客户端通过这种方式来删除远程服务器上的某个文档。客户端可以利用 PUT 和 DELETE 请求方式来管理远程服务器上的文档。

另一方面，从第 12 章、13 章和 14 章的内容可以看出，Web 应用响应客户端的请求时，经常会执行针对数据库的 CRUD 操作。

为了让请求服务器端执行 CRUD 操作的 HTTP 请求更加优雅、规范和简洁，RESTFul 风格应运而生，它能巧用 HTTP 请求方式来指定 CRUD 操作，也就是说，为 HTTP 请求方式赋予了新的含义：

- GET：执行查询操作。
- POST：执行新增操作。
- PUT：执行更新操作。
- DELETE：执行删除操作。

下面以对 Customer 对象进行 CRUD 操作为例，举例说明符合 RESTFul 风格的 URL 和请求方式：

```
//新增一个Customer对象
```

```
http://localhost:8080/helloapp/customer  请求方式为 POST

//更新 id 为 100 的 Customer 对象
http://localhost:8080/helloapp/customer/100  请求方式为 PUT

//删除 id 为 100 的 Customer 对象
http://localhost:8080/helloapp/customer/100  请求方式为 DELETE

//查询 id 为 100 的 Customer 对象
http://localhost:8080/helloapp/customer/100  请求方式为 GET

//查询所有的 Customer 对象
http://localhost:8080/helloapp/customer  请求方式为 GET
```

按照 HTTP 协议的规定，HTTP 请求的 URL 指定服务器端的特定资源的路径，例如以下 URL 中的“100”按照常规的理解，指的是“helloapp/customer”路径下的一个子路径：

```
http://localhost:8080/helloapp/customer/100
```

而 RESTFul 风格为以上“100”赋予了新的含义，把它当作是 id 变量的值。本章 15.2 节的范例 CustomerController 类会通过@PathVariable 注解来读取这个 id 变量的值：

```
@RequestMapping(value ="/customer/{id}", method =RequestMethod.GET)
public Customer findById(@PathVariable("id")Long id) {……}
```

确切地说，HTTP 请求由 URI（Uniform Resource Identifier，统一资源标识符）、请求方式、请求头和请求正文数据等组成。URL 是 URI 的子集。本书为了简化起见，采用了“HTTP 请求的 URL”这样的说法。

15.2 控制器类处理 RESTFul 风格的 HTTP 请求

本书第 9 章的 9.3 节（ 利用 Ajax 和 JSON 实现前后端分离）介绍了如何在控制器类中读取 JSON 格式的请求数据，以及发送 JSON 格式的响应数据。本节介绍的 CustomerController 类所处理的 HTTP 请求的请求方式和 URL 符合 RESTFul 风格，并且请求正文数据采用 JSON 格式。

例如以下是一个要求更新 ID 为 100 的 Customer 对象的 HTTP 请求的原文，请求方式为 PUT，请求的 URL 为“/customer/100”，请求正文为 JSON 格式的 Customer 对象的数据：

```
PUT /customer/100 HTTP/1.1
Content-Type: application/json;charset=UTF-8
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0(Windows NT 10.0; WOW64;Trident/7.0; rv:11.0)
Host: localhost
Content-Length: 36
Connection: Keep-Alive
Cache-Control: no-cache

{"id":"100","name":"Tom","age":"21"}
```

CustomerController 类为了处理 RESTFul 风格的 HTTP 请求，用@RestController 注解来标识。@RestController 注解是@Controller 和@ResponseBody 注解的缩写。这意味着 CustomerController 类中的所有请求处理方法，在默认情况下，都返回 JSON 格式的响应数

据。

如果 CustomerController 类的请求处理方法不需要返回 JSON 格式的响应数据，而是需要把请求转发给一个 JSP 文件，该怎么办呢？此时可以把返回类型设为 ModelAndView，例如：

```
@RequestMapping("/input")
public ModelAndView input() {
  return new ModelAndView("hello");  //把请求转发给 hello.jsp
}
```

以下例程 15-1 是 CustomerController 类的源代码。除了 input()方法，其他的 insert()、update()、delete()、findById()和 findAll()方法都处理 RESTFul 风格的 HTTP 请求，执行 CRUD 操作。

例程 15-1 CustomerController.java

```
@RestController
public class CustomerController{
  @Autowired
  CustomerService customerService;

  @RequestMapping("/input")
   public ModelAndView input() {
     return new ModelAndView("hello");
   }

  @RequestMapping(value = "/customer", method = RequestMethod.POST)
  public StringResult insert(@RequestBody Customer customer)  {
    boolean isSuccess=customerService.insertCustomer(customer);
    if(isSuccess)
      return new StringResult("对象插入成功");
    else
      return new StringResult("对象插入失败");
  }

  @RequestMapping(value ="/customer/{id}",method=RequestMethod.PUT)
  public StringResult update(@PathVariable("id")Long id,
                              @RequestBody Customer customer) {
    Customer currentCustomer=customerService.findCustomerById(id);
    if(currentCustomer==null)
       return new StringResult("ID 为"+id+"的对象不存在");
    currentCustomer.setName(customer.getName());
    currentCustomer.setAge(customer.getAge());
     boolean isSuccess=
             customerService.updateCustomer(currentCustomer);
    if(isSuccess)
      return new StringResult("对象更新成功");
    else
      return new StringResult("对象更新失败");
  }

  @RequestMapping(value = "/customer/{id}",
                  method = RequestMethod.DELETE)
  public StringResult delete(@PathVariable("id")Long id) {
```

```
        Customer customer=customerService.findCustomerById(id);
        Customer currentCustomer=customerService.findCustomerById(id);
        if(currentCustomer==null)
           return new StringResult("ID 为"+id+"的对象不存在");

        boolean isSuccess=customerService.deleteCustomer(customer);
        if(isSuccess)
          return new StringResult("对象删除成功");
        else
          return new StringResult("对象删除失败");
      }

      @RequestMapping(value ="/customer/{id}", method =RequestMethod.GET)
      public Customer findById(@PathVariable("id") Long id) {
        Customer customer=customerService.findCustomerById(id);
        return customer;
      }

      @RequestMapping(value = "/customer", method = RequestMethod.GET)
      public List<Customer> findAll()  {
        List<Customer> customers=customerService.findAllCustomers();
        return customers;
      }

      @ExceptionHandler(Exception.class)
      public String exHandle(HttpServletRequest request,
                                    Exception exception) {
        request.setAttribute("exception", exception);
        return "error";
      }
    }
```

本书第 3 章的 3.3.3 节（@GetMapping 和@PostMapping 等简化形式的注解）介绍了@RequestMapping 注解的好几种简写形式，CustomerController 类中的@RequestMapping 注解也可以改为简写形式，例如：

```
    @RequestMapping(value = "/customer", method = RequestMethod.GET)
    等价于：@GetMapping("/customer")

    @RequestMapping(value = "/customer/{id}",
                      method = RequestMethod.DELETE)
    等价于：@DeleteMapping("/customer/{id}")
```

15.2.1 读取客户请求中的 RESTFul 风格的 URL 变量

当客户端发出的 HTTP 请求为：

```
    http://localhost:8080/helloapp/customer/100  请求方式为 PUT
```

以上请求由 CustomerController 类的 update()方法来处理。update()方法的 id 参数用@PathVariable("id")注解来标识，@PathVariable("id")注解能够把 URL 中的 100 赋值给 id 参数。本书第 3 章的 3.4.5 节（ 用@PathVariable 注解绑定 RESTFul 风格的 URL 变量）已经介绍了@PathVariable 注解的用法。

15.2.2 读取客户请求中的 JSON 格式的 Java 对象的数据

CustomerController 类的 insert()方法用于向数据库保存一个 Customer 对象。客户端通过 POST 方式请求访问 insert()方法，客户端发送的 JSON 格式的 Customer 对象的数据会作为 HTTP 请求中的请求正文。

在 insert()方法中，customer 参数用@RequestBody 注解标识。@RequestBody 注解的作用是把客户端发送的 JSON 格式的 Customer 对象的数据转换为 Customer 对象。

15.2.3 请求处理方法的返回类型

CustomerController 类的 findById()方法返回 Customer 对象，而其他的 insert()、update()、delete()和 findAll()方法返回 StringResult 对象。StringResult 类是自定义的符合 JavaBean 风格的 String 类的包装类，参见例程 15-2。

例程 15-2　StringResult.java

```
package mypack;
public class StringResult  {
  private String result;
  public StringResult(){}
  public StringResult(String result){ this.result=result; }

  public String getResult() { return this.result; }
  public void setResult(String result){ this.result = result; }
  public String toString(){ return result; }
}
```

insert()、update()、delete()和 findAll()方法也可以返回一个 Map 类型的对象，在这个 Map 对象中包含了 String 类型的返回数据。例如 insert()方法可以按以下方式改写：

```
@RequestMapping(value = "/customer", method = RequestMethod.POST)
public Map<String,String> insert(@RequestBody Customer customer)  {
  boolean isSuccess=customerService.insertCustomer(customer);
  Map<String,String> map=new HashMap<String,String>();
  if(isSuccess)
    map.put("result","对象插入成功");
  else
    map.put("result","对象插入失败");
  return map;
}
```

不论 CustomerController 类返回 Customer 对象、StringResult 对象或 Map 对象，服务器端的 JSON 引擎都会把它们转换为 JSON 格式的响应数据，再发送到客户端。

在客户端的 Ajax 代码中，以下代码会读取响应数据中的 StringResult 对象的 result 属性值或者 Map 对象中 Key 为 result 的值：

```
//读取成功响应的结果
success : function(data) {  //data 表示响应数据
  if (data != null) {
    $("input[type=reset]").trigger("click");//触发 reset 按钮
    alert(data.result);
  }
}
```

在客户端的 Ajax 代码中，以下代码会读取响应数据中的 Customer 对象的各个属性：

```
//读取成功响应的结果
success : function(data) {  //data 表示响应数据
  if (data != null && data !="") {
    alert("用户名: "+data.name
           + ",  年龄: "+data.age);
  }
}
```

15.3 客户端发送 RESTFul 风格的 HTTP 请求

在本范例中，客户端的 hello.jsp 通过 Ajax 来发送 RESTFul 风格的 HTTP 请求，并且请求正文数据采用 JSON 格式，Ajax 还会处理 JSON 格式的响应结果，参见例程 15-3。

例程 15-3 hello.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>Hello</title>
<script type="text/javaScript"
  src="${pageContext.request.contextPath}
          /resource/js/jquery-3.2.1.min.js">
</script>
</head>
<body>
  <form action="">
    ID: <input type="text" name="id" id="id"/><p>
    用户名: <input type="text" name="name" id="name"/><p>
    年龄: <input type="age" name="age" id="age" /><p>

    <input type="button" value="新增" onclick="insert()" />
    <input type="button" value="更新" onclick="update()" />
    <input type="button" value="删除" onclick="remove()" />
    <input type="button" value="根据 ID 查询" onclick="findById()" />
    <input type="button" value="查询所有" onclick="findAll()" />
    <input type="reset" name="重置" style="display: none;" />

  </form>
</body>
<script type="text/javaScript">
  function insert() {
    //获取输入的表单数据
    var v_id = $("#id").val();
    var v_name = $("#name").val();
    var v_age = $("#age").val();

    $.ajax({
      //请求路径
      url : "${pageContext.request.contextPath }/customer",

      //请求方式
      type : "post",
```

```
      //请求正文的数据格式为 JSON 字符串
      contentType : "application/json;charset=UTF-8",

      //data 表示发送的 JSON 格式的请求数据
      data : JSON.stringify({
        id :  v_id,
        name : v_name,
        age: v_age
      }),

      //响应正文的数据格式为 JSON 字符串
      dataType : "json",

      //读取成功响应的结果
      success : function(data) {  //data 表示响应数据
         if (data != null) {
          $("input[type=reset]").trigger("click");//触发 reset 按钮
          alert(data.result);
        }
      }
    });
  }

  function update() {
    //获取输入的表单数据
    var v_id = $("#id").val();
    var v_name = $("#name").val();
    var v_age = $("#age").val();

    $.ajax({
      //请求路径
      url : "${pageContext.request.contextPath }/customer/"+v_id ,

      //请求方式
      type : "put",

      //请求正文的数据格式为 JSON 字符串
      contentType : "application/json;charset=UTF-8",

      //data 表示发送的 JSON 格式的请求数据
      data : JSON.stringify({
        id :  v_id,
        name : v_name,
        age: v_age
      }),

      //响应正文的数据格式为 JSON 字符串
      dataType : "json",

      //读取成功响应的结果
      success : function(data) {  //data 表示响应数据
         if (data != null) {
```

```
        $("input[type=reset]").trigger("click");//触发 reset 按钮
        alert(data.result);
      }
    }
  });
}

function remove() {……}

function findById() {
  //获取输入的表单数据
  var v_id = $("#id").val();
  $.ajax({
    //请求路径
    url : "${pageContext.request.contextPath }/customer/"+v_id,

    //请求方式
    type : "get",

      //响应正文的数据格式为 JSON 字符串
    dataType : "json",

    //读取成功响应的结果
    success : function(data) {  //data 表示响应数据
      if (data != null && data !="") {
        alert("用户名："+data.name
                + "， 年龄："+data.age);
      }
    }
  });
}

function findAll() {
  //获取输入的表单数据
  var v_id = $("#id").val();
  $.ajax({
    //请求路径
    url : "${pageContext.request.contextPath }/customer",

    //请求方式
    type : "get",

      //响应正文的数据格式为 JSON 字符串
    dataType : "json",

    //读取成功响应的结果
    success : function(data) {  //data 表示响应数据
       if (data != null) {
         var info="";
         for(var i=0;i<data.length;i++) //遍历所有 Customer 对象
           info+=data[i].id+","+data[i].name+","+data[i].age+"\r\n";
         alert(info);
      }
```

```
            }
          });
        }
      </script>
      </html>
```

通过浏览器访问 http://localhost:8080/helloapp/input，就会显示 hello.jsp 的页面，参见图 15-1。

图 15-1　hello.jsp 生成的网页

在图 15-1 的网页上选择“更新”按钮，会执行 hello.jsp 的 update()方法，update()方法的以下代码指定 RESTFul 风格的 HTTP 请求的 URL 和请求方式：

```
        //请求路径
        url : "${pageContext.request.contextPath }/customer/"+v_id ,

        //请求方式
        type : "put",
```

15.4　通过 RestTemplate 类模拟客户程序

在实际软件开发过程中，前端与后端的开发团队分别开发各自的软件模块。对于采用 RESTFul 风格的 Web 应用，如果要等到前端与后端均开发完成自己的模块后，才能整体调试和运行程序，这会大大影响软件开发的效率。

Spring API 提供了一个 org.springframework.web.client.RestTemplate 类，它支持在服务器端单独测试负责处理 RESTFul 风格的 HTTP 请求的控制器类。

RestTemplate 类提供了一些方法，模拟客户端发出 RESTFul 风格的 HTTP 请求：

- 新增操作：postForEntity(String url, Object request, Class<T> responseType, Object... uriVariables)
- 更新操作：put(String url, Object request, Object... uriVariables)
- 删除操作：delete(String url, Object... uriVariables)
- 查询操作：getForObject(String url, Class<T> responseType, Object... uriVariables)

以上 postForEntity()、put()、delete()和 getForObject()方法分别采用 POST、PUT、DELETE 和 GET 请求方式，请求服务器端执行新增、更新、删除和查询操作。以上方法的 url 参数指定请求访问的 URL，request 参数表示请求正文数据，responseType 参数指定返回数据的类型，uriVariables 参数设定 URL 变量。

以下例程 15-4 的 TestController 类通过 RestTemplate 类来访问 CustomerController 类的

各种请求处理方法。

例程 15-4　TestController.java

```
@Controller
public class TestController {
  @Autowired
  private RestTemplate restTemplate;
  private String SERVICE_URL =
              "http://localhost:8080/helloapp/customer";

  @RequestMapping("/insert")
  public String insert() {
    Customer customer=new Customer();
    customer.setId(null);
    customer.setName("Linda");
    customer.setAge(18);

    // 发送 POST 请求
    StringResult result= restTemplate.postForObject(
              SERVICE_URL , customer,StringResult.class);
    System.out.println(result);
    return "result";
  }

  @RequestMapping("/update")
  public String update() {
    Customer customer=new Customer();
    customer.setId(Long.valueOf(1L));
    customer.setName("Linda");
    customer.setAge(18);

    // 发送 PUT 请求
    restTemplate.put(SERVICE_URL+"/1" , customer);
    System.out.println("更新完毕");
    return "result";
  }

  @RequestMapping("/delete")
  public String delete() {
    // 发送 DELETE 请求
    restTemplate.delete(SERVICE_URL+"/1");
    System.out.println("删除完毕");
    return "result";
  }

  @RequestMapping("/findone")
  public String findById() {
    // 发送 GET 请求
    Customer customer=restTemplate.getForObject(SERVICE_URL+"/1",
                                   Customer.class);
    if(customer!=null)
      System.out.println(customer.getId()+","
                +customer.getName()+","+customer.getAge());
```

```
      return "result";
    }

    @RequestMapping("/findall")
    public String findAll() {
      // 发送 GET 请求
      Customer[] customers=restTemplate
                   .getForObject(SERVICE_URL,Customer[].class);
      System.out.println("查询到"+customers.length+"个 Customer 对象");
      for(Customer c : customers)
        System.out.println(c.getId()+","+c.getName()+","+c.getAge());
      return "result";
    }
  }
```

TestController 类的 restTemplate 实例变量由 Spring 框架提供，在 Spring 的配置文件 applicationContext.xml 中配置了 restTemplate Bean 组件：

```
<bean id="restTemplate"
      class = "org.springframework.web.client.RestTemplate" />
```

通过浏览器访问以下 URL：

```
http://localhost:8080/helloapp/insert
http://localhost:8080/helloapp/update
http://localhost:8080/helloapp/findone
http://localhost:8080/helloapp/findall
http://localhost:8080/helloapp/delete
```

以上 URL 会访问 TestController 类的相关方法，而这些方法又会通过 RestTemplate 类发出 RESTFul 风格的 HTTP 请求，请求访问 CustomerController 类的相关方法。

15.5 小结

在 RESTFul 风格的 HTTP 请求中，GET、POST、 PUT 和 DELETE 请求方式被赋予了新的含义，是指向数据库执行查询、新增、更新和删除操作。在 URL 中，还可以包含变量。

这种新的含义由谁来解读呢？是控制器类。在控制器类的请求处理方法中，会根据特定的请求方式来调用业务逻辑服务层的相关方法，执行 CRUD 操作。请求处理方法还会通过 @PathVariable 注解读取 URL 中的变量。

Spring API 提供的 RestTemplate 类能够模拟客户程序，发出 RESTFul 风格的 HTTP 请求，从而测试服务器端的控制器类的功能。以下图 15-2 显示了本章范例中各个组件之间的调用关系。

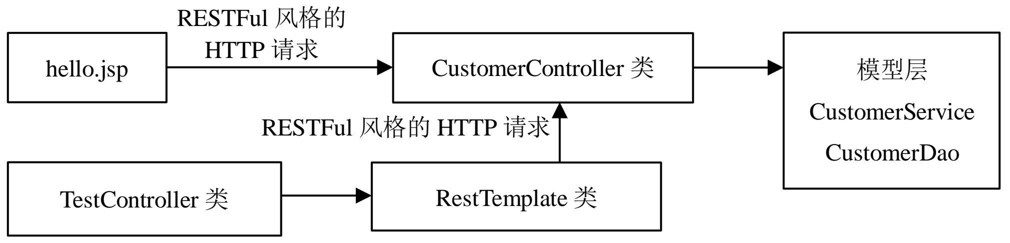

图 15-2 本章范例中各个组件之间的调用关系

15.6　思考题

1．对于 RESTFul 风格，以下哪种 HTTP 请求方式对应 CRUD 中的更新操作？（单选）

（a）GET　（b）PUT　（c）POST　（d）DELETE

2．以下哪个选项提供了符合 RESTFul 风格的 URL 以及请求方式，表示查询 ID 为 50 的 Customer 对象？（单选）

（a）http://localhost:8080/helloapp/customer/50　请求方式为 POST

（b）http://localhost:8080/helloapp/customer?id=50　请求方式为 GET

（c）http://localhost:8080/helloapp/customer/50　请求方式为 DELETE

（d）http://localhost:8080/helloapp/customer/50　请求方式为 GET

3．@RestController 注解是哪两个注解的缩写？（单选）

（a）@Controller 和@RequestBody　（b）@ResponseBody 和@RequestBody

（c）@Controller 和@ResponseBody　（d）@Controller 和@PathVariable

4．以下选项中的 restTemplate 变量是 RestTemplate 类型，customer 变量引用 ID 为 1 的 Customer 对象，哪个选项用于更新 ID 为 1 的 Customer 对象？（单选）

（a）restTemplate.put("http://localhost:8080/helloapp/customer/1" , customer);

（b）restTemplate.getForObject("http://localhost:8080/helloapp/customer/1" , customer);

（c）restTemplate.postForEntity("http://localhost:8080/helloapp/customer/1" , customer);

（d）restTemplate.put("http://localhost:8080/helloapp/customer?id=1" , customer);

5．以下哪些选项说法正确？（多选）

（a）RESTFul 是 HTTP 协议的最新版本中的规范。

（b）RESTFul 对一些 HTTP 请求方式赋予了 CRUD 的含义。

（c）Servlet 容器提供了专门的 REST 引擎，负责解析 RESTFul 风格的 HTTP 请求。

（d）控制器类通过@PathVariable 注解来读取 RESTFul 风格的 URL 中的变量。

6．客户端发送了一个 RESTFul 风格的 HTTP 请求，要求新增一个 Customer 对象，在控制器类中应该如何声明处理该请求的方法？（单选）

（a）

```
@RequestMapping(value = "/customer", method = RequestMethod.POST)
public StringResult insert(@RequestBody Customer customer){……}
```

（b）

```
@RequestMapping(value = "/customer", method = RequestMethod.POST)
public StringResult insert(@RequestParam Customer customer){……}
```

（c）

```
@RequestMapping(value = "/customer", method = RequestMethod.INSERT)
public StringResult insert(Customer customer){……}
```

（d）

```
@PostMapping("/customer")
public StringResult insert(@RequestBody Customer customer){……}
```

参考答案

答案：1．b　2．d　3．c　4．a　5．b,d　6．a,d