第4章 Java Web开发

目录

前言

- 本章主要介绍了Java Web开发的相关知识，从最基础的HTTP协议讲起，然后对一系列Java Web开发中常用的工具如Tomcat、Spring Boot、MyBatis等进行了介绍，同时还对数据设计进行了讲解。

目标

- 学完本课程后，你可以
  - 了解HTTP协议的基本知识。
  - 掌握Tomcat和Servlet的基本原理。
  - 了解什么是REST风格，掌握JSON数据格式。
  - 掌握Spring Boot的基本使用。
  - 了解数据设计的相关概念和方法，并对SQL、JDBC、DAO和Mysql等概念有一个初步的了解。
  - 初步掌握Mybatis的基本思想和使用方法。
  - 通过一个完整的案例了解Mybatic和Spring Boot的结合使用方式。

目录

本节概述和学习目标

- 本节我们介绍协议栈、数据包、HTTP的基础知识。

- 了解HTTP协议的基础知识。

协议栈

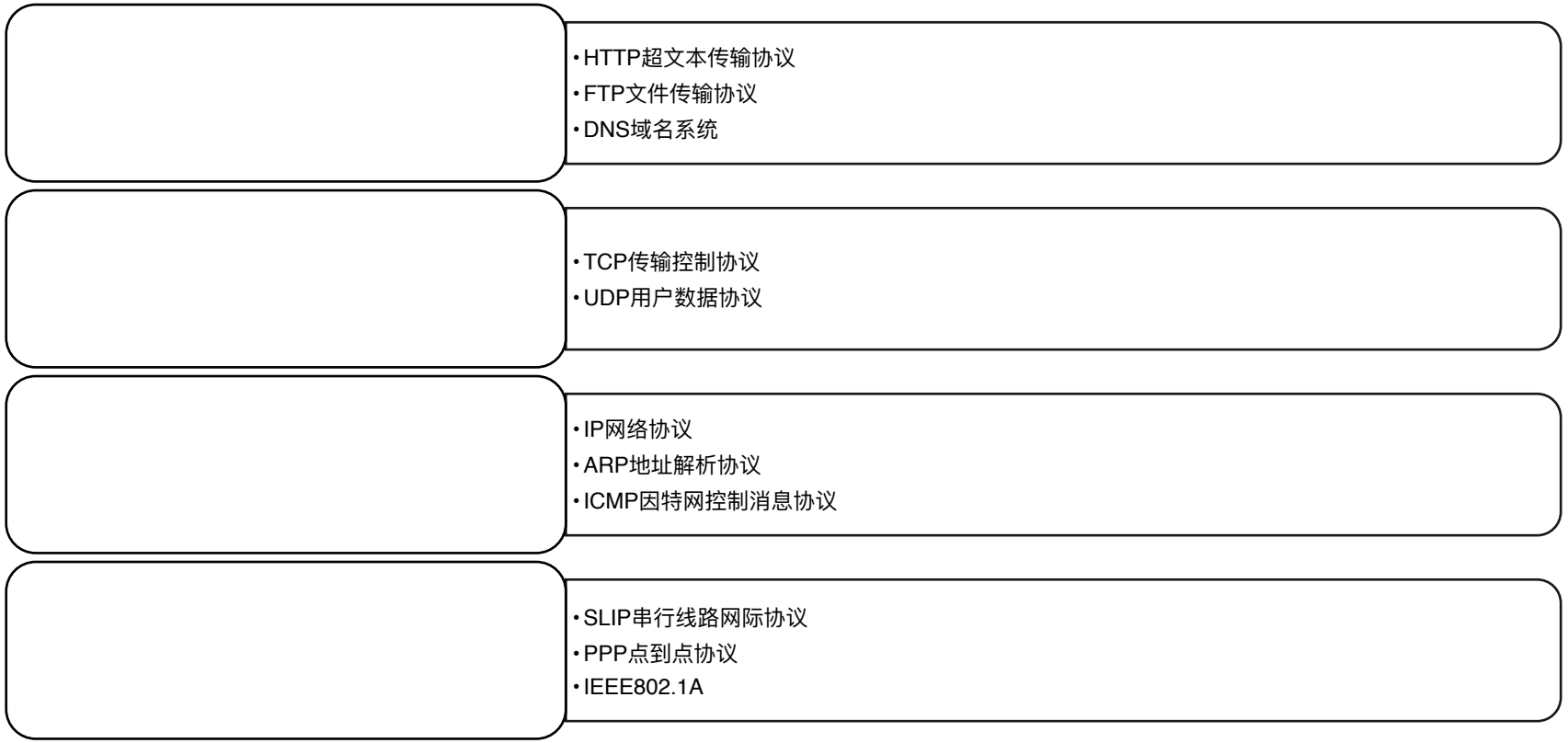
- 协议：网络中计算机之间为能够互相通信所制定一些规则
  - 协议可以规定怎么搜索到目标计算机，怎么开始和结束通信，怎么保证可靠性等内容。
  - TCP/IP是互联网相关的各类协议族的总称，计算机网络就是在TCP/IP协议族的基础上运作，通常抽象为4层的协议栈。

协议栈

- TCP/IP的四层协议栈
  - 应用层为操作系统或网络应用程序提供访问网络服务的接口。
    - HTTP属于它内部的一个子集，提供了超文本数据的传输
  - 传输层将上层数据分段并提供端到端的、可靠的或不可靠的传输以及端到端的差错控制和流量控制问题。
    - 主要用到的是TCP协议、UDP协议
  - 网络层对子网间的数据包进行路由选择，还可以实现拥塞控制，网络互连等功能。
  - 网络接口层则为网络层提供物理上的可靠的数据传输。

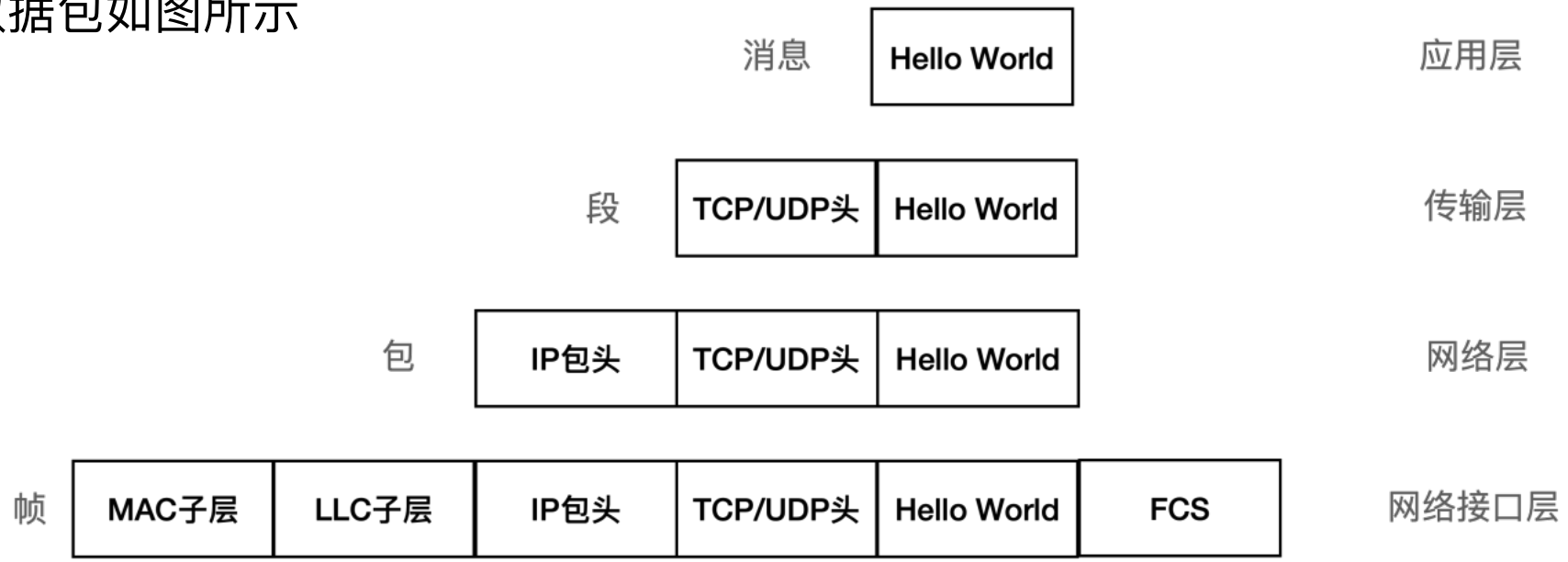
协议栈

- TCP/IP的四层协议栈



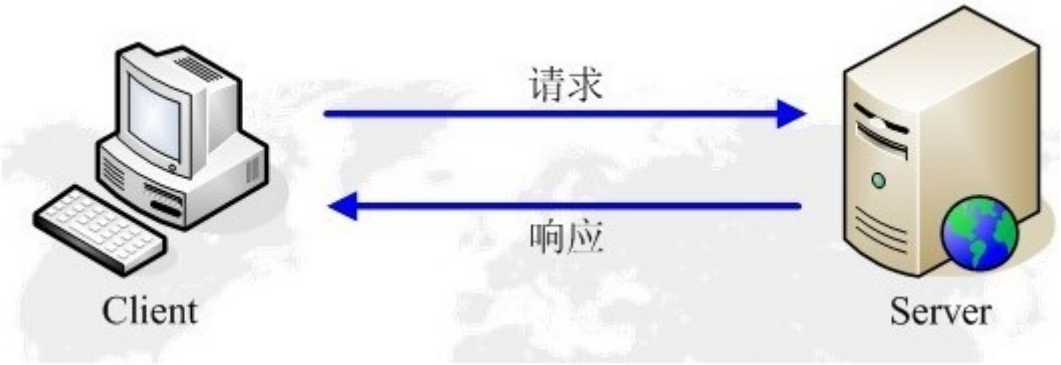
数据包

- 数据包：当应用层协议使用 TCP/IP 协议传输数据时，TCP/IP 协议簇可能会将应用层发送的数据分成多个包依次发送。
  - 数据的接收方收到的数据可能是分段的或者拼接的，所以它需要对接收的数据进行拆分或者重组
  - 各层数据包如图所示



HTTP协议 – 客户端服务器模式

- HTTP是TCP/IP协议栈应用层的一个协议，是一个客户端服务器模型（C/S）。
  - 客户端（Client）发起请求（Request），服务器（Server）回送响应（Response）。
  - HTTP是一个无状态的协议
    - 在非Keep-Alive模式下，每次建立连接都是完全独立的，当一个客户端向服务器端发出请求，然后服务器返回响应，连接就被关闭了，在服务器端不保留连接的有关信息。
    - 当使用Keep-Alive模式时，Keep-Alive功能使客户端到服务器端的连接持续有效。



HTTP协议 – HTTP工作过程

- 一次HTTP操作称为一个事务，其工作整个过程如下：
  - 地址解析
  - 封装HTTP请求数据包
  - 封装成TCP包，建立TCP连接（TCP的三次握手）
  - 客户机发送请求命令
  - 服务器响应
  - 服务器关闭TCP连接

HTTP协议 – HTTP工作过程

- 地址解析：
  - 如客户用浏览器请求某个网站页面，需要通过其URL来指定访问目标。
    - http://localhost.com:8080/index.htm
    - 我们从URL中分解结果如下：
      - 协议名：http、主机名：localhost.com、端口：8080、对象路径：/index.htm
    - 真实访问时，还需要通过域名系统DNS解析域名localhost.com，得到主机的IP地址。

HTTP协议 – HTTP工作过程

- 封装HTTP请求数据包：
  - 把以上部分结合本机自己的信息，封装成一个HTTP请求数据包。

HTTP协议 – HTTP工作过程

- 封装成TCP包，建立TCP连接（TCP的三次握手）
  - 在HTTP工作开始之前，客户机（Web浏览器）首先要通过网络与服务器建立连接，该连接是通过TCP来完成的，该协议与IP协议共同构建Internet，即著名的TCP/IP协议族。HTTP是比TCP更高层次的应用层协议，根据规则，只有低层协议建立之后，才能进行更高层协议的连接，因此，首先要建立TCP连接，一般TCP连接的端口号是80。这里是8080端口。

HTTP协议 – HTTP工作过程

- 客户机发送请求命令：
  - 建立连接后，客户机发送一个请求给服务器。例如：GET/sample/hello.jsp HTTP/1.1。

HTTP协议 – HTTP工作过程

- 服务器响应：
  - 服务器接到请求后，给予相应的响应信息。
    - 例如： HTTP/1.1 200 OK。服务器向浏览器发送头信息后，它会发送一个空白行来表示头信息的发送到此为结束，接着，它就以Content-Type应答头信息所描述的格式发送用户所请求的实际数据。

HTTP协议 – HTTP工作过程

- 服务器关闭TCP连接：
  - 一般情况下，一旦Web服务器向浏览器发送了响应数据，它就要关闭TCP连接。
  - 如果浏览器或者服务器在其头信息加入了这行代码：Connection:Keep-Alive，TCP连接在发送后将仍然保持打开状态，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

HTTP协议 – URL格式请求

- 统一资源定位符（Uniform Resource Locator，URL）
  - URL是因特网上标准的资源的地址，如同在网络上的门牌。
  - URL标准格式由三部分组成：协议类型、服务器地址(和端口号)、路径(Path)。
  - HTTP URL可以携带键值对形式的参数。参数以问号？开始并采用 name=value的格式。如果存在多个URL参数，则参数之间用一个&符隔开。
  - 具体格式和示例如下：
    - 格式： 协议类型://服务器地址[:端口号]路径
    - 示例： http://blog.csdn.com/users?gender=male

HTTP协议 – Request

- HTTP Request
  - 格式主要包含方法、路径、HTTP版本、头信息和Body

HTTP协议 – Request

- HTTP Request
  - 常用的方法有GET和POST
    - GET方法用于获取资源，不会对服务器数据进行修改，所以不发送 Body
      - GET /users/1 HTTP/1.1
      - Host: api.github.com
    - POST方法用于增加或修改资源 ， 它会将发送给服务器的内容写在Body里面
      - POST /users HTTP/1.1
      - Host: api.github.com
      - Content-Type: application/x-www-form-urlencoded
      - Content-Length: 13
      - name=rengwuxian&gender=male

HTTP协议 – Response

- HTTP Response
  - 格式包括HTTP版本、状态码、状态消息、头信息、空行、Body

HTTP协议 – Response

- HTTP Response常用状态码
  - 1xx：临时性消息。如：100（继续发送）、101（正在切换协议）；
  - 2xx：成功。最典型的是200（OK）、201（创建成功）；
  - 3xx：重定向。如：301（永久移动）302（暂时移动）、304（内容未改变）；
  - 4xx：客户端错误。如：400（客户端请求错误）、401（认证失败）、403（被禁止）、404（找不到内容）；
  - 5xx：服务器错误。如：500（服务器内部错误）。

目录

本节概述和学习目标

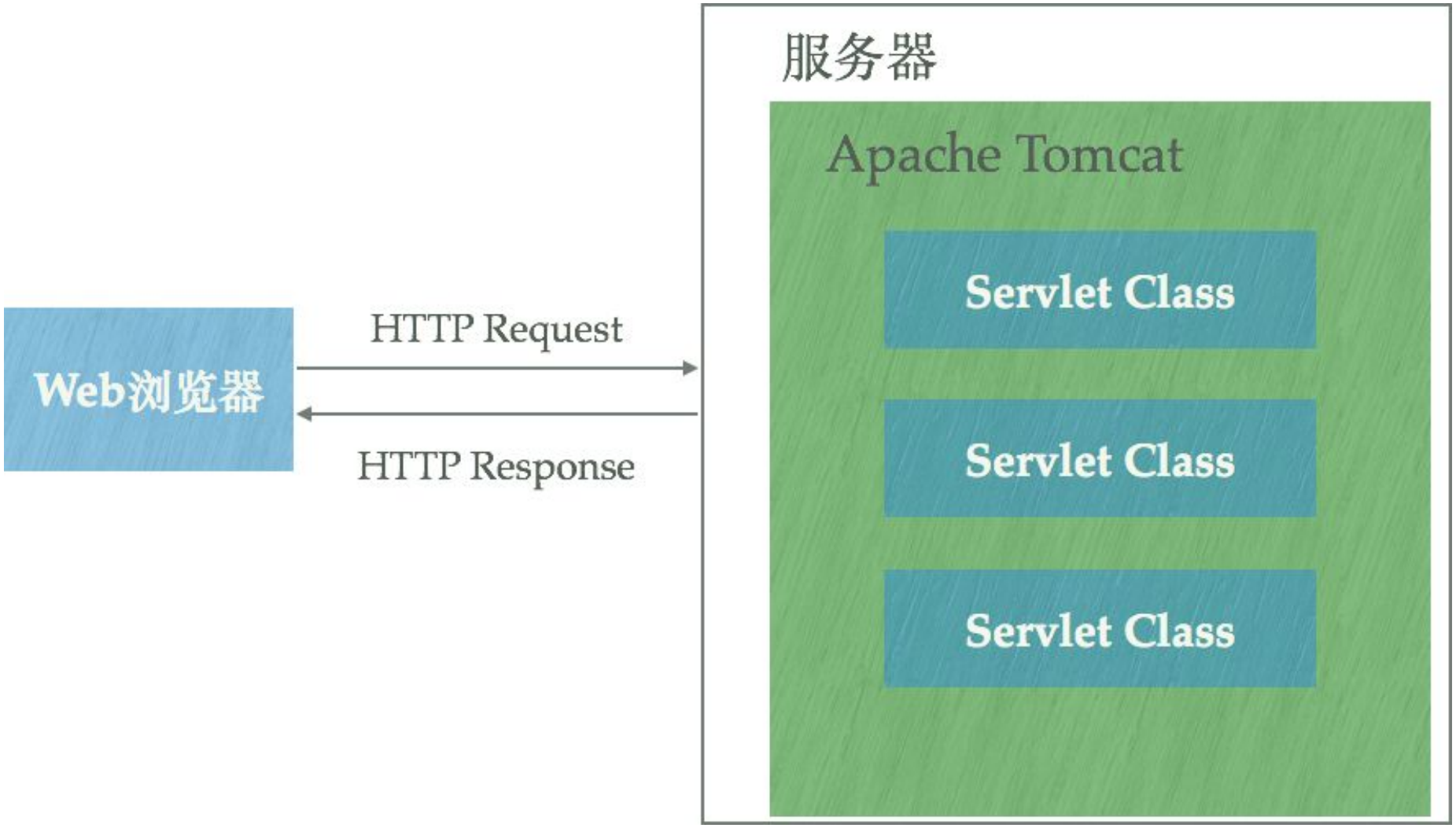
- 本节介绍了Tomcat的职责，以及Servlet在Tomcat中如何工作的。

- 掌握Tomcat和Servlet的基本原理。

Tomcat

- Apache Tomcat是一个Java语言编写的运行在JVM之上的Web服务器（Java Servlet容器），它和HTTP服务器一样，绑定IP地址并监听TCP端口，同时还包含以下职责：
  - 管理Servlet程序的生命周期
  - 将URL映射到指定的Servlet进行处理
  - 与Servlet程序合作处理由Web浏览器发来的HTTP请求：根据HTTP请求生成HttpServletRequest对象并传递给Servlet进行处理，将Servlet中的HttpServletResponse对象生成的内容返回给浏览器

Tomcat

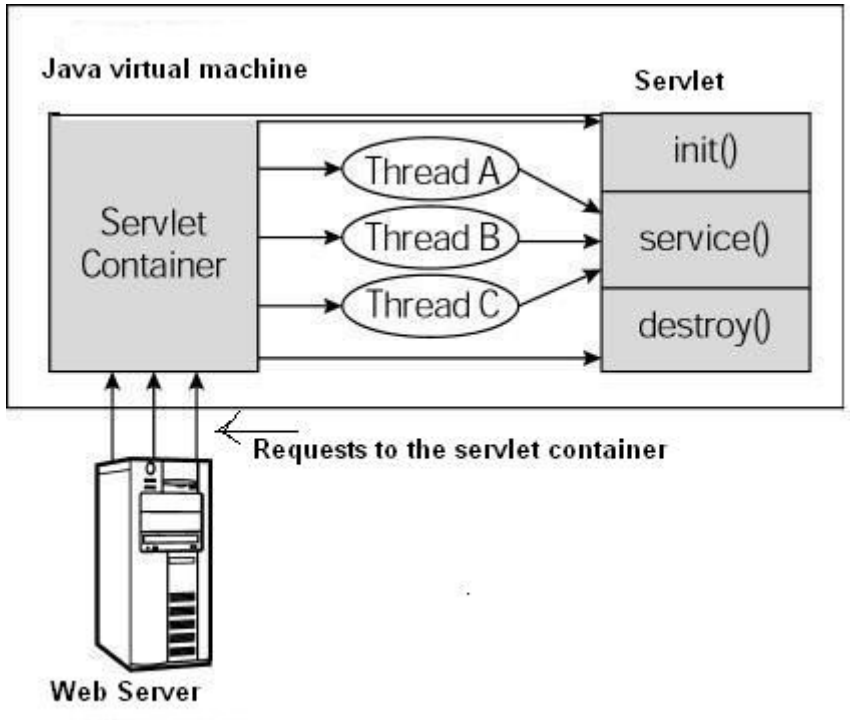

服务器
Apache Tomcat
Servlet Class
Servlet Class
Servlet Class
Web浏览器
HTTP Request
HTTP Response

Servlet

- Java Servlet是用Java编写的服务器端程序
  - 其主要功能在于交互式地浏览和修改数据，生成动态Web内容。
- Java Servlet运行在 Web 服务器或应用服务器上，作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。

Servlet – 生命周期

- 一个典型的 Servlet 生命周期方案如下：
  - 到达服务器的 HTTP 请求被委派到 Servlet 容器。
  - Servlet 容器在调用 service() 方法之前加载 Servlet类。
  - Servlet 容器处理由多个线程产生的多个请求，每个线程执行一个单一的 Servlet 对象的 service() 方法。



Servlet – service()方法

- service() 方法是执行实际任务的主要方法：
  - Servlet 容器（即 Web 服务器）调用 service() 方法来处理来自客户端（浏览器）的请求，并把格式化的响应写回给客户端。
  - 每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。
  - service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法。
  - 该方法的接口如下：

```
public void service(ServletRequest request,
                    ServletResponse response)
      throws ServletException, IOException{
}
```

Servlet – service()方法

- service() 方法由容器调用，service 方法在适当的时候调用 doGet、doPost、doPut、doDelete 等方法。
  - 不用对 service() 方法做任何动作，只需要根据来自客户端的请求类型来重写 doGet() 或 doPost() 即可。
  - doGet() 方法：GET 请求来自于一个 URL 的正常请求，或者来自于一个未指定 METHOD 的 HTML 表单，它由 doGet() 方法处理。
  - doPost() 方法：POST 请求来自于一个特别指定了 METHOD 为 POST 的 HTML 表单，它由 doPost() 方法处理。

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet 代码
}
```

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet 代码
}
```

Servlet – Hello World， Servlet

```
// 导入必需的 java 库
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// 扩展 HttpServlet 类
public class HelloWorld extends HttpServlet {

  private String message;

  public void init() throws ServletException
  {
     // 执行必需的初始化
     message = "Hello World";
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // 设置响应内容类型
     response.setContentType("text/html");

     // 实际的逻辑是在这里
     PrintWriter out = response.getWriter();
     out.println("<h1>" + message + "</h1>");
  }

  public void destroy()
  {
     //
  }
}
```

Servlet – Hello World，Servlet

- 通过如下命令编译 Servlet：javac HelloWorld.java
- 部署 Servlet
  - 默认情况下，Servlet 应用程序位于路径 /webapps/ROOT 下，且类文件放在 /webapps/ROOT/WEB-INF/classes
  - 把 HelloWorld.class 复制到 /webapps/ROOT/WEB-INF/classes 中
  - 在位于 /webapps/ROOT/WEB-INF/ 的 web.xml 文件中创建如右图所示条目
  - 启动 Tomcat 服务器，最后在浏览器的地址栏中输入 http://localhost:8080/HelloWorld

```
<web-app>
   <servlet>
      <servlet-name>HelloWorld</servlet-name>
      <servlet-class>HelloWorld</servlet-class>
   </servlet>

   <servlet-mapping>
      <servlet-name>HelloWorld</servlet-name>
      <url-pattern>/HelloWorld</url-pattern>
   </servlet-mapping>
</web-app>
```

Servlet – Hello World, Servlet

目录

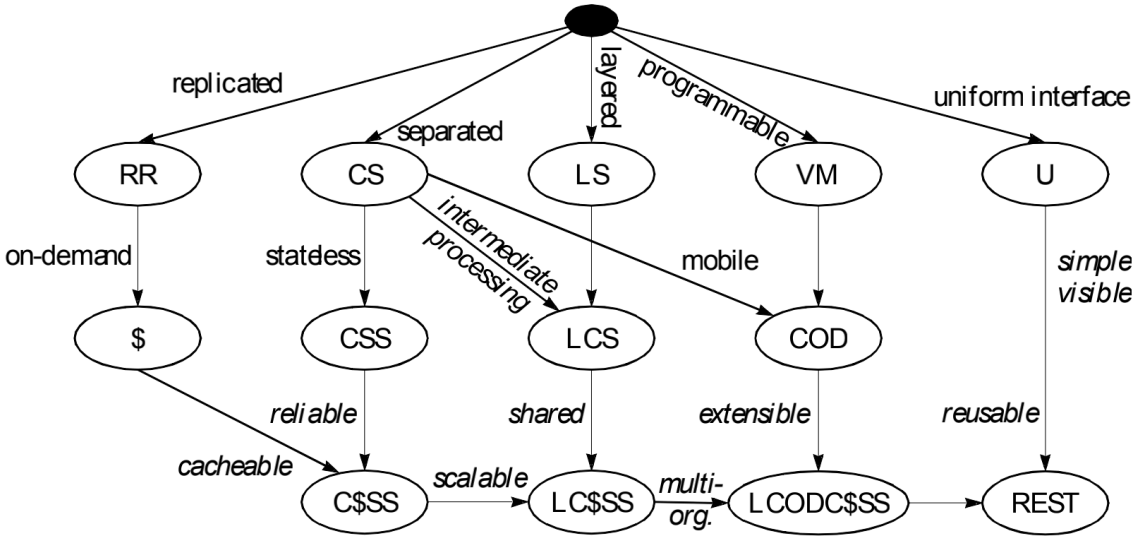
本节概述和学习目标

- 本节介绍了REST风格和JSON数据格式

- 了解REST风格的由来和规范，掌握JSON格式

REST风格 - REST风格的提出

- Roy Fielding博士在2000年他的博士论文中提出了REST（Representational State Transfer）风格的软件架构模式：
  - REST基本上迅速取代了复杂而笨重的SOAP，成为Web API的标准。
  - REST风格的软件架构模式满足的各种要求：可重复、分层、无状态、可靠、统一接口、可扩展、可重用等。

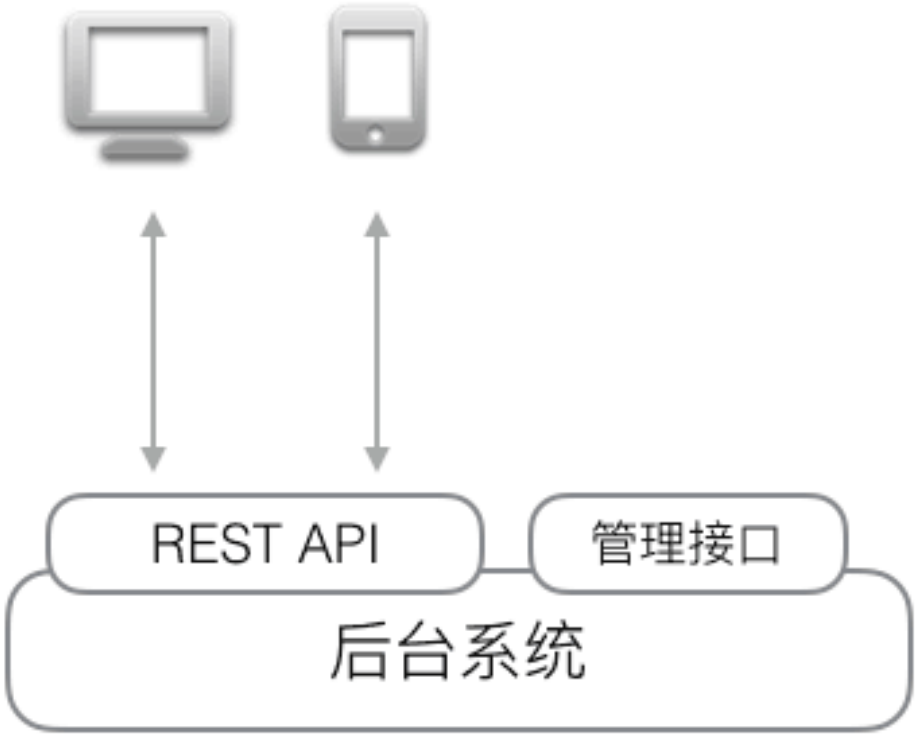


REST风格 - Web API 与前后端分离

- 如果一个URL返回的不是HTML，而是机器能直接解析的数据，这个URL就可以看成是一个Web API：
  - 比如，读取http://localhost:3000/api/products/123，如果能直接返回Product的数据，那么机器就可以直接读取。
  - REST风格就是一种设计Web API的风格
    - 由于JSON能直接被JavaScript读取，所以，以JSON格式编写的REST风格的Web API，即REST API，具有简单、易读、易用的特点。
    - 通过REST API操作数据，可以极大地把前端和后端的代码隔离，使得后端代码易于测试，前端代码编写更简单。
    - 如果我们把前端页面看作是一种用于展示的客户端，那么REST API就是为客户端提供数据、操作数据的接口。

REST风格 - Web API 与前后端分离

- 一当一个Web应用以REST API的形式对外提供功能时，整个应用的结构就扩展为：



REST风格 - REST API 规范

- REST请求仍然是标准的HTTP请求，
- 除了GET请求外，POST、PUT等请求的body是JSON数据格式，请求的Content-Type为application/json。

REST风格 - REST API 规范

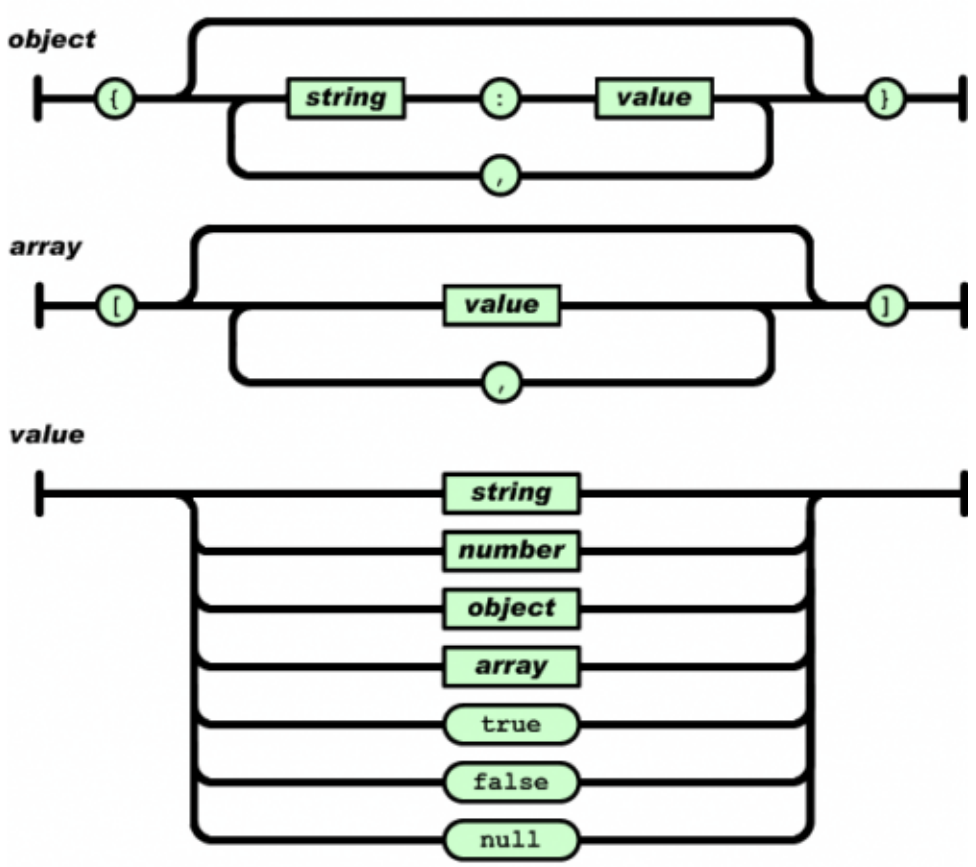
- REST规范定义了资源的通用访问格式，虽然它不是一个强制要求，但遵守该规范可以让人易于理解。
  - 假如商品Product就是一种资源。获取所有Product的URL是：GET /api/products
  - 获取某个指定的Product，例如，id为123的Product。URL是：GET /api/products/123
  - 新建一个Product使用POST请求，JSON数据包含在body中。URL是：POST /api/products
  - 更新一个Product使用PUT请求，例如，更新id为123的Product。URL是：PUT /api/products/123
  - 删除一个Product使用DELETE请求，例如，删除id为123的Product。URL是：DELETE /api/products/123
  - 资源还可以按层次组织。例如，获取某个Product的所有评论，使用的URL是：GET /api/products/123/reviews
  - 当我们只需要获取部分数据时，可通过参数限制返回的结果集，例如，返回第2页评论，每页10项，按时间排序，使用的URL是：
    - GET /api/products/123/reviews?page=2&size=10&sort=time

JSON

- JSON（JavaScript Object Notation，JavaScript对象表示法，读作/ˈdʒeɪsən/）是一种由道格拉斯·克罗克福特构想和设计、轻量级的数据交换语言
  - 该语言以易于让人阅读的文字为基础，用来传输由属性值或者序列性的值组成的数据对象。
  - JSON是JavaScript的一个子集，但JSON是独立于语言的文本格式，并且采用了类似于C语言家族的一些习惯。
- JSON 数据格式与语言无关
  - 当前很多编程语言都支持 JSON 格式数据的生成和解析。
  - JSON 的官方 MIME 类型是 application/json，文件扩展名是 .json。

JSON

- JSON本质是一个正则表达式，JSON的parser是一个有限状态机



JSON

- JSON示例

```
{
    "firstName": "John",
    "lastName": "Smith",
    "sex": "male",
    "age": 25,
    "address":
    {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber":
    [
        {
          "type": "home",
          "number": "212 555-1234"
        },
        {
          "type": "fax",
          "number": "646 555-4567"
        }
    ]
}
```

目录

本节概述和学习目标

- 本节介绍Spring Boot的特色和Hello World项目

- 了解Spring的特性，且能够搭建最基础的Spring Boot项目

Spring Boot简介

- Spring　Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。其具有如下特色：
  - 简化依赖，提供整合的依赖项，告别逐一添加依赖项的烦恼。
  - 简化配置，提供约定俗成的默认配置，告别编写各种配置的繁琐。
  - 简化部署，内置 Servlet 容器，开发时一键即运行。可打包为 jar 文件，部署时一行命令即启动。
  - 简化监控，提供简单方便的运行监控方式。

Hello World – POM文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   …
   <parent>…</parent>
   <properties>…</properties>

   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>

      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-test</artifactId>
         <scope>test</scope>
      </dependency>
   </dependencies>

   <build>
      <plugins>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
         </plugin>
      </plugins>
   </build>
</project>
```

Hello World – 程序入口

- Spring Boot要求main()方法所在的启动类必须放到根package下，命名不做要求。
- 启动Spring Boot应用程序只需要一行代码加上一个注解@SpringBootApplication。

```
// Application.java
package hello;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

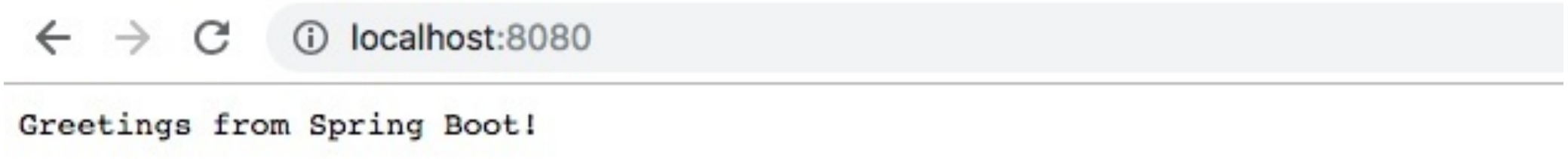
Hello World – Controller

- 启动之后，对URL“/”路径的响应是通过HelloController类来实现的：
  - 需要添加@RestController和@RequestMapping(“/”)注解来说明。
    - 如果是其它路径，修改RequestMapping注解的参数为对应的路径即可。

```
// HelloController.java
package hello;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Spring Boot!";
    }

}
```

Hello World –启动和浏览器访问

- 启动项目有三种方式：
  - 在IDE运行Application的main方法。
  - 使用命令 mvn spring-boot:run”在命令行启动该应用。
  - 运行“mvn package”进行打包时，会打包成一个可以直接运行的 JAR 文件，使用“java -jar”命令就可以直接运行。



目录

本节概述和学习目标

- 本节介绍服务器如何通过关系型数据库进行数据持久

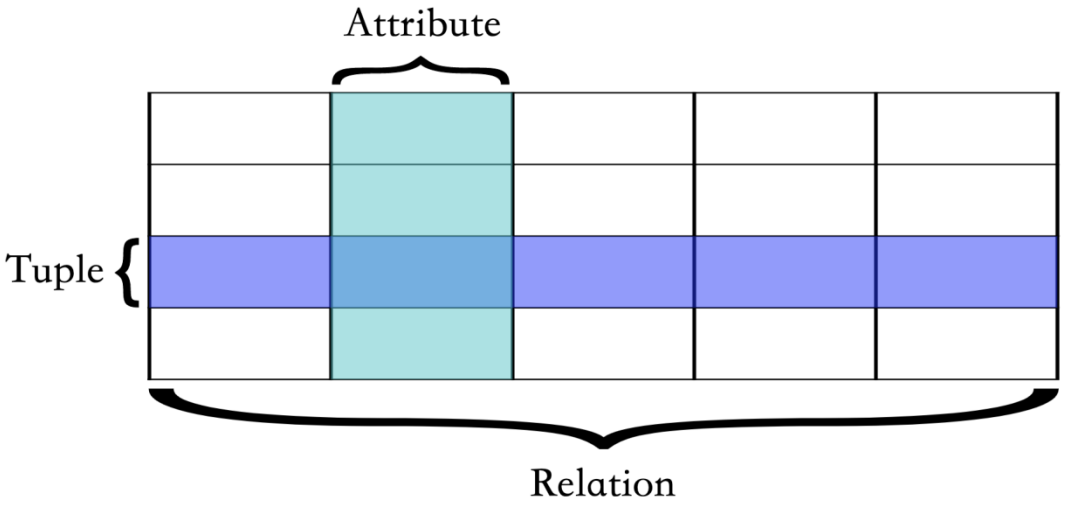
- 了解持久化的意义，掌握关系型数据库的基本概念，了解SQL重要命令，掌握JDBC原理，理解DAO相关概念，了解Mysql数据库在Spring Boot中的基本用法

数据持久化

- 为什么持久化?
  - 因为数据只有能够被记录（record）、存储（storage）、回忆（recall），才能成为知识，才能被我们人类所利用。
- 在人类漫长的历史中出现了很多持久化的方法：
  - 实物记录（绳结、泥板、竹片、绢布、纸张）、电子文件、以及数据库管理系统。

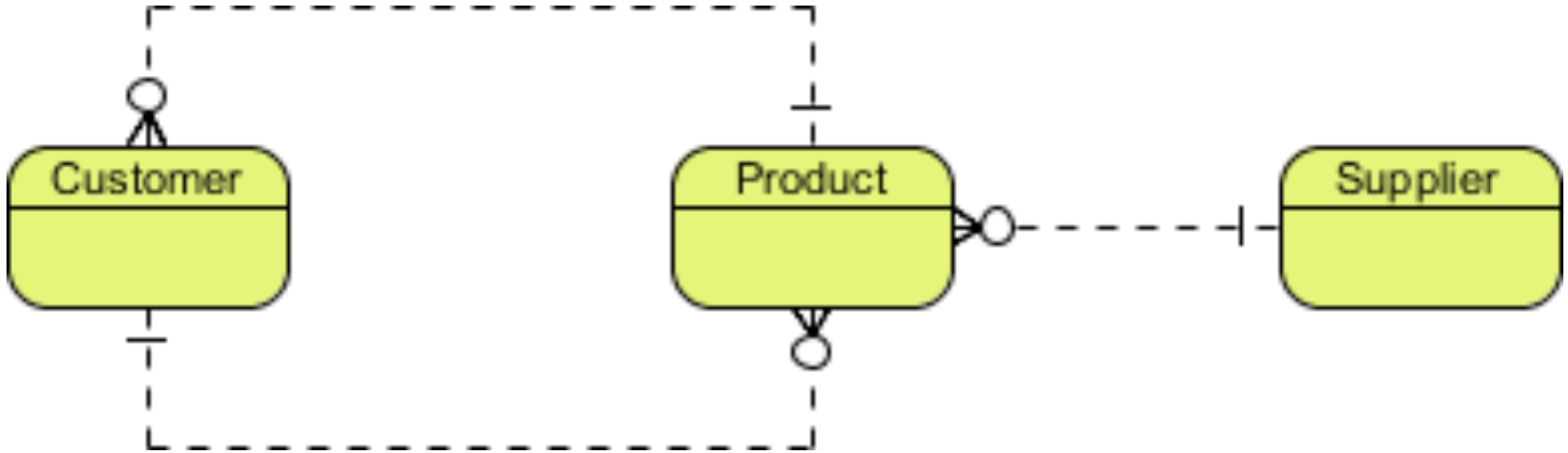
关系型数据库 – 关系

- 在关系型数据库中，我们用一张表表达一个关系
  - 表（关系Relation）是以行（值组Tuple）和列（属性Attribute）的形式组织起来的数据的集合。
  - 一个数据库包括一个或多个表（关系Relation）
    - 例如，可能有一个有关作者信息的名为authors的表（关系Relation）。每列（属性Attribute）都包含特定类型的信息，如作者的姓氏。每行（值组Tuple）都包含有关特定作者的所有信息：姓、名、住址等等。



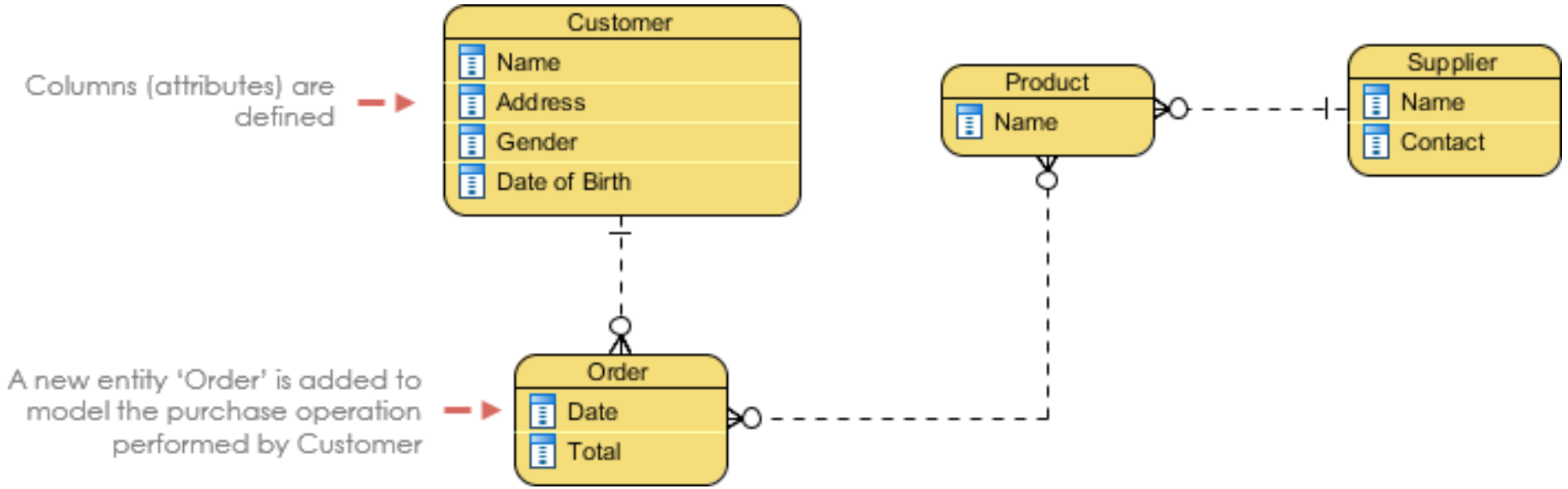
关系型数据库 – 关系

- 概念模式是从用户的需求及其业务领域出发，进过分析和总结，提炼出来用以描述用户业务的概念类，以及概念类之间的关系。
  - 如图所示，一个顾客可能买了多个商品。这时候，可以不关注概念的属性，主要关注的是类和类之间的关系。



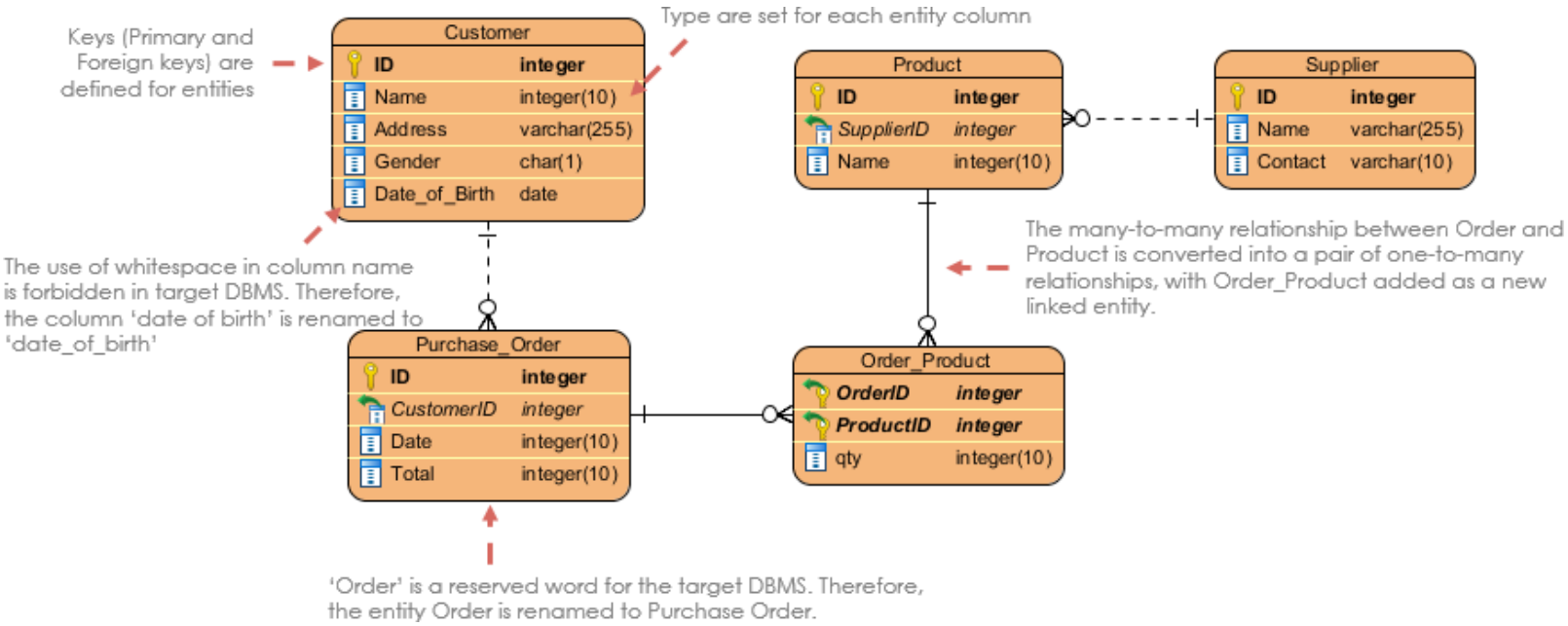
关系型数据库 – 逻辑模型

- 模型就是将概念模型具体化，要实现概念的功能时候具体需要的实体和属性。
  - 如图所示，新的订单实体和客户的属性被加入到逻辑模型中。概念模型只表明系统要实现什么，但怎样实现，用什么工具实现还没有讲。



关系型数据库 – 物理模型

- 物理模型则是对逻辑模型在真实数据库中实现的描述：
  - 物理模型包括数据表、主键、外键、字段、数据类型、长度、是否为空、默认值等。
  - 如图所示，订单类被转换为Purchase_Order表。其中ID是主键，是Integer类型的。



关系型数据库 – 将类图转换为关系表

- 通常的转换过程如下：
  - (1)通常一个类/对象映射为一张表
    - 类的属性映射为表的列名
    - 类的实例对象就是表的行
  - (2)要为每个转换后的表建立主键
    - 类/对象通过引用来唯一标识自己
    - 关系/表通过主键类唯一标识自己
  - (3)处理关联
    - 类/对象通过链接实现关联
    - 关系/表通过主键/外键对实现关联
      - 1:1关联：其中一个表的主键，作为另一个表的外键；两端等价
      - 1:N关联：将1端表的主键，作为N端表的外键
      - M:N关联：建立中间表，将M的主键和N的主键都作为中间表的外键
      - 在包含/聚合关联中，将整体的主键放在部分中作为外键
      - 关联的最小基数为0，意味着相应外键可以为NULL

关系型数据库 – 将类图转换为关系表

- (3)处理关联
  - a)类/对象通过链接实现关联
  - b)关系/表通过主键/外键对实现关联
    - i.1:1关联：其中一个表的主键，作为另一个表的外键；两端等价
    - ii.1:N关联：将1端表的主键，作为N端表的外键
    - iii.M:N关联：建立中间表，将M的主键和N的主键都作为中间表的外键
    - iv.在包含/聚合关联中，将整体的主键放在部分中作为外键
    - v.关联的最小基数为0，意味着相应外键可以为NULL

SQL语句

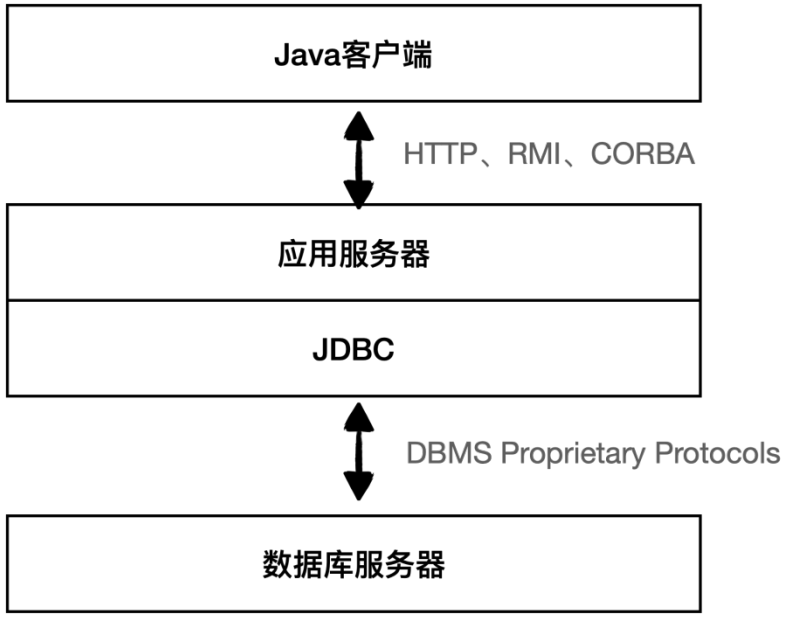
- 结构化查询语言（Structured Query Language，SQL） 是用于管理关系数据库管理系统的一套标准语言。
  - SQL 的范围包括数据插入、查询、更新和删除，数据库模式创建和修改，以及数据访问控制。

SQL语句

- 一些重要的SQL命令如下所示：
  - SELECT - 从数据库中提取数据
  - UPDATE - 更新数据库中的数据
  - DELETE - 从数据库中删除数据
  - INSERT INTO - 向数据库中插入新数据
  - CREATE DATABASE - 创建新数据库
  - ALTER DATABASE - 修改数据库
  - CREATE TABLE - 创建新表
  - ALTER TABLE - 变更（改变）数据库表
  - DROP TABLE - 删除表
  - CREATE INDEX - 创建索引（搜索键）
  - DROP INDEX - 删除索引

JDBC原理

- Java数据库连接（Java Database Connecitivity，JDBC）是Java语言中用来规范客户端程序如何访问数据库管理系统的应用程序接口，提供了诸如查询和更新数据库中工属具的方法
  - 如图所示，我们客户端通过各种方式访问Java应用服务器，我们通过ORM将对业务对象的操作，转换为统一的JDBC方式，再由JDBC和数据库服务器进行通信。其中主要的步骤是：
    - 登记并加载JDBC驱动器
    - 建立与SQL数据库的连接
    - 传送一个SQL操作
    - 获得数据结果



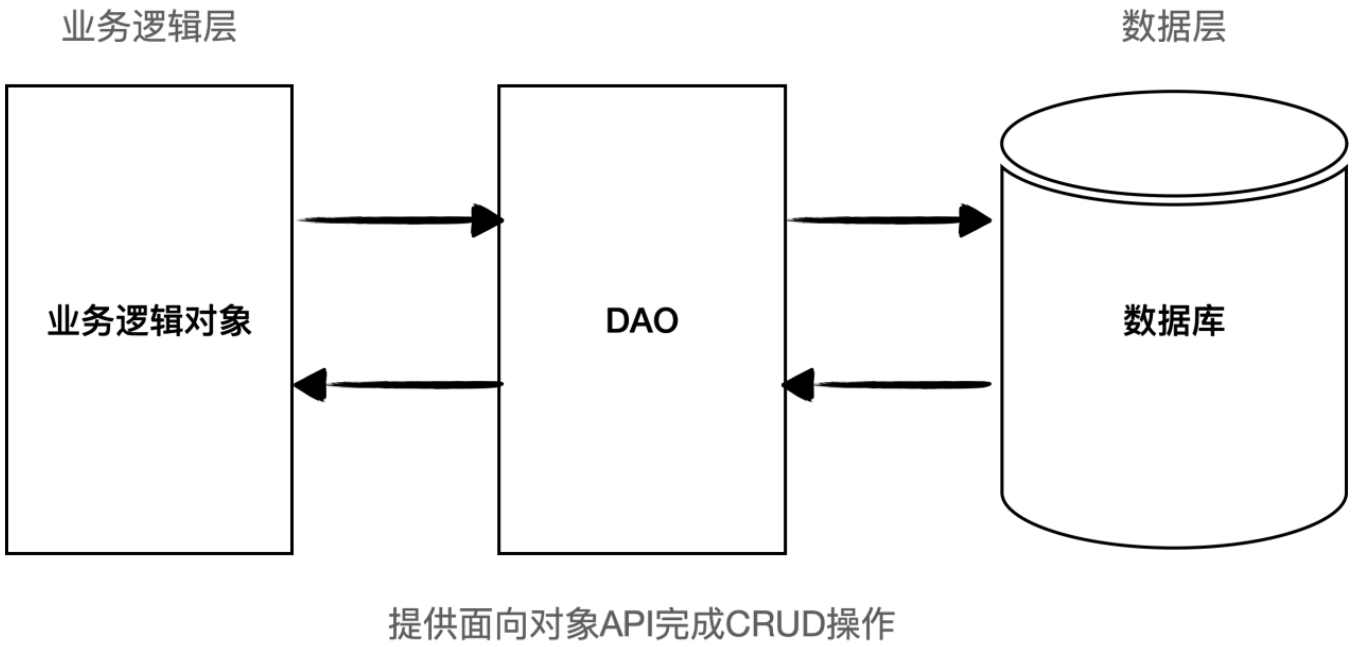
JDBC原理

- 使用JDBC完成一个SQL操作的示例代码

```
//1.加载驱动程序
Class.forName("com.mysql.jdbc.Driver");
//2. 获得数据库连接
Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
//3.操作数据库，实现增删改查
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT user_name, age FROM imooc_goddess");
//如果有数据，rs.next()返回true
while(rs.next()){
  System.out.println(rs.getString("user_name")+" 年龄："+rs.getInt("age"));
}
```

DAO - DAO

- 数据存取对象（Data Access objects，DAO）是指位于业务逻辑层和数据层之间实现以面向对象API为形式的对持久化数据进行CRUD操作的对象。
  - 通俗来讲，就是将数据库操作都封装起来。这样对逻辑对象的操作，就直接转换为对数据库中记录的操作。



DAO - DAO

- DAO的优势就在于：
  - 提供了数据访问的DAO接口：
    - 隔离了数据访问代码和业务逻辑代码。业务逻辑代码直接调用DAO方法即可，完全感觉不到数据库表的存在，都是对逻辑对象的操作方法。
  - 隔离了不同数据库实现：
    - 采用面向接口编程，如果底层数据库变化，如由 MySQL 变成 Oracle 只要增加 DAO 接口的新实现类即可，原有 MySQL 实现不用修改。这符合 "开-闭" 原则。该原则降低了代码的耦合性，提高了代码扩展性和系统的可移植性。

DAO – 组成部分

- 实现DAO的时候需要实现以下几个部分：
  - DAO接口：
    - 把对数据库的所有操作定义成抽象方法，可以提供多种实现
  - DAO实现类：
    - 针对不同数据库给出DAO接口定义方法的具体实现
  - 实体类：
    - 用于存放与传输对象数据
  - 数据库连接和关闭工具类：
    - 避免了数据库连接和关闭代码的重复使用，方便修改

DAO – 组成部分

```
//DAO 接口:
public interface PetDao {
    /**
     * 查询所有宠物
     */
    List<Pet> findAllPets() throws Exception;
}
```

DAO – 组成部分

```
//DAO 实现类:
public class PetDaoImpl extends BaseDao implements PetDao {
    /**
     * 查询所有宠物
     */
    public List<Pet> findAllPets() throws Exception {
        Connection conn=BaseDao.getConnection();
        String sql="select * from pet";
        PreparedStatement stmt= conn.prepareStatement(sql);
        ResultSet rs=    stmt.executeQuery();
        List<Pet> petList=new ArrayList<Pet>();
        while(rs.next()) {
            Pet pet=new Pet(
                    rs.getInt("id"),
                    rs.getInt("owner_id"),
                    rs.getInt("store_id"),
                    rs.getString("name"),
                    rs.getString("type_name"),
                    rs.getInt("health"),
                    rs.getInt("love"),
                    rs.getDate("birthday")
                    );
                petList.add(pet);
        }
        BaseDao.closeAll(conn, stmt, rs);
        return petList;
    }
}
```

DAO – 组成部分

```
//宠物实体类(省略了getter和setter方法)
public class Pet {
    private Integer id;
    private Integer ownerId;    //主人ID
    private Integer storeId;    //商店ID
    private String name;    //姓名
    private String typeName;    //类型
    private int health;    //健康值
    private int love;    //爱心值
    private Date birthday;    //生日
}
```

DAO – 组成部分

```
//通过JDBC连接数据库
public class BaseDao {
    private static String driver="com.mysql.jdbc.Driver";
    private static String url="jdbc:mysql://127.0.0.1:3306/epet";
    private static String user="root";
    private static String password="root";
        static {
            try {
                Class.forName(driver);
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url, user, password);
    }

    public static void closeAll(Connection conn,Statement stmt,ResultSet rs)
throws SQLException {
        if(rs!=null) {
            rs.close();
        }
        if(stmt!=null) {
            stmt.close();
        }
        if(conn!=null) {
            conn.close();
        }
    }
```

```
 public int executeSQL(String preparedSql, Object[] param)
throws ClassNotFoundException {
        Connection conn = null;
        PreparedStatement pstmt = null;
        /* 处理SQL,执行SQL */
        try {
            conn = getConnection(); // 得到数据库连接
            pstmt = conn.prepareStatement(preparedSql); // 得到
PreparedStatement对象
            if (param != null) {
                for (int i = 0; i < param.length; i++) {
                    pstmt.setObject(i + 1, param[i]); // 为预编译sql设置
参数
                }
            }
        ResultSet num = pstmt.executeQuery(); // 执行SQL语句
        } catch (SQLException e) {
            e.printStackTrace(); // 处理SQLException异常
        } finally {
            try {
                BaseDao.closeAll(conn, pstmt, null);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return 0;
}
```

MySql数据库

- MySQL 是现在最流行的关系型数据库管理系统
  - 具体可以参考官方网站的说明，安装并使用。https://dev.mysql.com/doc/refman/8.0/en/installing.html

MySql数据库

- 使用Mysql的时候，在POM文件中添加如下依赖

```
<!--mysql-->
      <dependency>
         <groupId>mysql</groupId>
         <artifactId>mysql-connector-java</artifactId>
         <scope>runtime</scope>
      </dependency>
```

MySql数据库

- 在Spring Boot项目中使用前需要在application.yml文件修改MySql的用户和密码

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mysql_test
    username: root
    password: root
    driver: com.mysql.jdbc.Driver
```

目录

本节概述和学习目标

- 本节从ORM出发，介绍Mybatis框架的机制

- 掌握ORM基本概念，理解Mybatis作用、特点，掌握Mybatis的基本使用方法

ORM

- 结合面向对象编程的思想，可以将关系型数据库中的关系用对象来表达，这其中就是要解决面向对象和关系型数据库之间存在的如下表的映射，也就是对象关系映射（Object Relation Mapping）

关系型数据库	面向对象
表（Table）	类（Class）
记录（Record）	对象（Object）
字段（Field）	属性

ORM

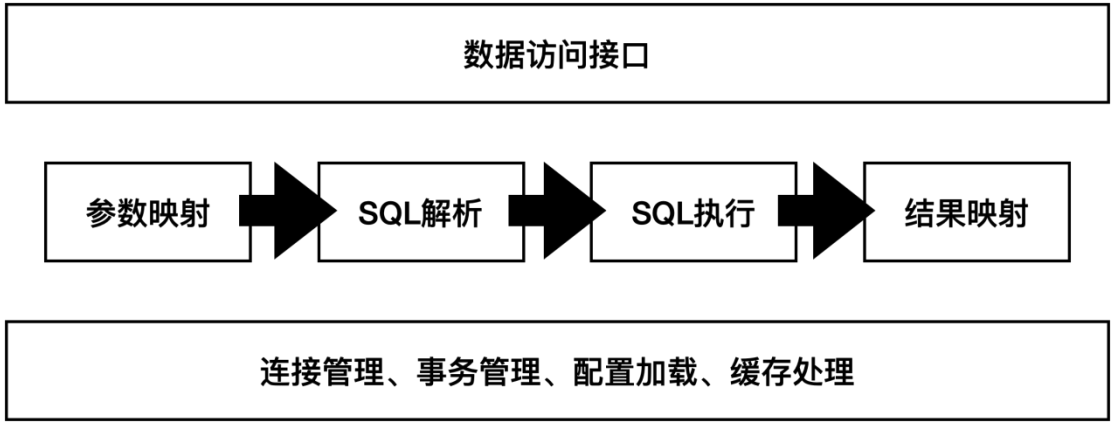
- 举例说明
  - SELECT id, first_name, last_name, phone, birth_date, sex FROM persons WHERE id = 10
  - 如果我们直接在逻辑代码直接通过SQL操作数据库，那么具体代码写法如下：
    - res = db.execSql(sql);
    - name = res[0]["FIRST_NAME"];
  - 如果改成 ORM 的写法，则具体代码写法如下：
    - p = Person.get(10);
    - name = p.first_name;

ORM

- 通过ORM，使逻辑代码直接操作对象，而不是使用具体的SQL进行数据操作。具体的SQL操作由ORM框架自身来实现。开发者可以将注意力更加关注于逻辑代码的编写。

ORM

- ORM具体框架
  - 最上层是接口层提供数据访问接口。
  - 中间是数据处理层，通过参数映射、SQL解析、SQL执行、结果映射完成对象和关系型数据库的映射。
  - 最底层是基础支撑层提供一个基础管理的服务。



ORM

- ORM是Object和Relation之间的映射
  - Object->Relation
  - Relation->Object
- Hibernate和MyBatis是常见的两种ORM框架：
  - Hibernate是个完整的ORM框架，完全可以通过对象关系模型实现对数据库的操作，拥有完整的JavaBean对象与数据库的映射结构来自动生成SQL。
  - MyBatis完成的是Relation->Object，也就是其所说的Data Mapper Framework，仅有基本的字段映射，对象数据以及对象实际关系仍然需要通过手写SQL来实现和管理。

Mybatis

- MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架：
  - MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。
  - MyBatis 可以对配置和原生Map使用简单的 XML 或注解，将接口和持久化对象（Persistent Object，PO）映射成数据库中的记录。
  - PO通常使用Java 的 普通的 Java对象（Plain Old Java Objects，POJOs）来实现。
  - MyBatis没有实现JPA，它和ORM框架的设计思路不完全一样。
    - MyBatis是拥抱SQL，而ORM则更靠近面向对象，不建议写SQL，则推荐用框架自带的类SQL代替。
    - 更准确的说法是，MyBatis是一种持久层框架，是一种SQL映射框架，而不是完整的ORM框架。

Mybatis的安装和使用

- 要使用 MyBatis， 只需将 mybatis-x.x.x.jar 文件置于类路径（classpath）中即可。
- 如果使用 Maven 来构建项目，则需将下面的依赖代码置于 pom.xml 文件中：
  - <dependency>
  - <groupId>org.mybatis</groupId>
  - <artifactId>mybatis</artifactId>
  - <version>x.x.x</version>
  - </dependency>

Mybatis的安装和使用

- Springboot中可以使用如下依赖

```
<!--spring boot mybatis-->
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>2.1.4</version>
        </dependency>
```

Mybatis的安装和使用

- 具体的MyBatis相关的文件包括
  - Mapper接口：
    - 业务逻辑使用的API。
  - PO对象：
    - 接口传递的持久化数据对象。
  - Mapper实现的XML文件：
    - 用SQL语句实现Mapper接口。
  - 项目POM文件：
    - 包括Mybatis和MySQL相关类库的依赖。
  - 项目application.yml文件：
    - 其中包括JDBC和Mybatis的配置。

Mybatis的安装和使用

- Mapper实现的XML文件

```
@Mapper
@Repository
public interface AdminMapper {

    int addManager(User user);

    List<User> getAllManagers();
}
```

Mybatis的安装和使用

- PO对象

```
public class User {  //省略getter、setter
    private Integer id;
    private String email;
    private String password;
    private String userName;
    private String phoneNumber;
    private double credit;
    private UserType userType;
}
```

Mybatis的安装和使用

- Mapper实现的XML文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.hotel.data.admin.AdminMapper">
    <insert id="addManager" parameterType="com.example.hotel.po.User" useGeneratedKeys="true" keyProperty="id">
        insert into User(email,password,usertype)
        values(#{email},#{password},#{userType})
    </insert>
    <select id="getAllManagers" resultMap="User">
        select * from User where usertype='HotelManager'
    </select>


    <resultMap id="User" type="com.example.hotel.po.User">
        <id column="id" property="id"></id>
        <result column="email" property="email"></result>
        <result column="password" property="password"></result>
        <result column="username" property="userNzame"></result>
        <result column="phonenumber" property="phoneNumber"></result>
        <result column="credit" property="credit"></result>
        <result column="usertype" property="userType"></result>
    </resultMap>
</mapper>
```

Mybatis的安装和使用

- 项目POM文件

```
<!--mybatis-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.0</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

Mybatis的安装和使用

- 项目application.yml文件

```
spring:
  datasource:
     url: jdbc:mysql://localhost:3306/Hotel?serverTimezone=CTT&characterEncoding=UTF-8
     username: root
     password: ******
     driver-class-name: com.mysql.cj.jdbc.Driver
     max-active: 200
     max-idle: 20
     min-idle: 10
  thymeleaf:
     cache: false
  jackson:
    time-zone: GMT+8

mybatis:
  mapper-locations: classpath:dataImpl/*/*Mapper.xml
```

目录

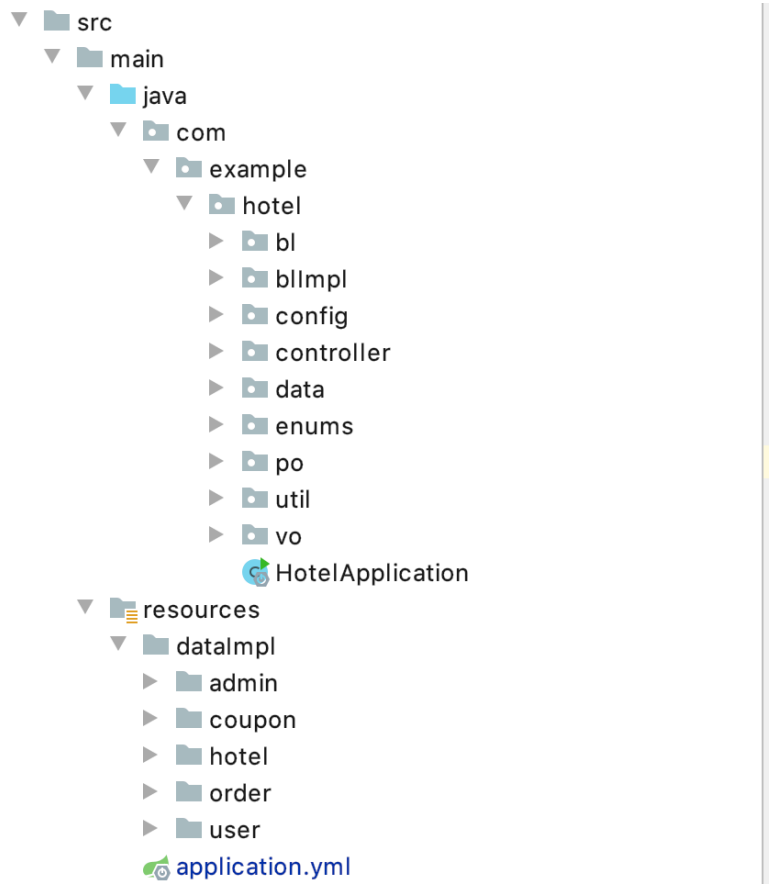
本节概述和学习目标

- 本节介绍了结合了Mybatis的Spring Boot的实战案例的代码级解析

- 掌握Mybatis+Spring Boot的项目搭建方法，理解其项目结构，掌握调用流程

项目结构

- Spring Boot+Mybatis项目的结构：
  - contoller:负责处理REST API请求。
  - bl：业务逻辑层接口。
  - blImpl:业务逻辑层实现。
  - data：数据层接口。
  - po：持久化对象，用于数据层和逻辑层之间数据传递。
  - vo：值对象，用于逻辑层和展示层（前端）数据传递。
  - dataImpl：数据层实现，Mybatis是XML文件。



后端调用流程

- 整个后端调用的流程是controller->bl->blImpl->data->dataImpl
  - controller是控制器。承担了接受REST API请求之后，根据URL找到对应的bl来处理。
  - bl是业务逻辑的接口。
  - blImpl才是业务逻辑真正的实现。blImpl会调用数据层的接口data。
  - dataImpl则是通过Mybatis给出的关于数据层接口的实现。将data接口与SQL语句相映射。

源代码解析 - controller

- controller.coupon.CouponController类
  - @RestController REST风格API的控制器
  - @RequestMapping("/api/coupon") 请求的URL
  - @PostMapping("/hotelTargetMoney") Post请求子路径
  - @RequestParam 请求参数
  - @Autowired 自动装配
  - couponVO 传给前端的数据

源代码解析 - controller

- controller.coupon.CouponController类

```
@RestController                                                                    }
@RequestMapping("/api/coupon")
public class CouponController {

    @Autowired
    private CouponService couponService;

    @PostMapping("/hotelTargetMoney")
    public ResponseVO addHotelTargetMoneyCoupon(@RequestBody
HotelTargetMoneyCouponVO hotelTargetMoneyCouponVO) {

        CouponVO couponVO =
couponService.addHotelTargetMoneyCoupon(hotelTargetMoneyCouponVO);

        return ResponseVO.buildSuccess(couponVO);
    }

    @GetMapping("/hotelAllCoupons")
    public ResponseVO getHotelAllCoupons(@RequestParam Integer hotelId) {
        return ResponseVO.buildSuccess(couponService.getHotelAllCoupon(hotelId));
    }
```

源代码解析 - bl

- bl.coupon.CouponService类
  - HotelTargetMoneyCouponVO、hotelId是前端输入的参数
  - OrderVO 是逻辑层返回的结果

源代码解析 - controller

- bl.coupon.CouponService类

```
public interface CouponService {
    /**
     * 返回某一订单可用的优惠策略列表
     * @param orderVO
     * @return
     */
    List<Coupon> getMatchOrderCoupon(OrderVO orderVO);  ## Use CouponVO instead of Coupon

    /**
     * 查看某个酒店提供的所有优惠策略（包括失效的）
     * @param hotelId
     * @return
     */
    List<Coupon> getHotelAllCoupon(Integer hotelId);

    /**
     * 添加酒店满减优惠策略
     * @param couponVO
     * @return
     */
    CouponVO addHotelTargetMoneyCoupon(HotelTargetMoneyCouponVO couponVO);
}
```

源代码解析 - blImpl

- blImpl.coupon.CouponServiceImpl类
  - @Service 是逻辑对象的注解
  - 逻辑层服务的实现调用了数据层Mapper的接口

源代码解析 - controller

- blImpl.coupon.CouponServiceImpl类

```
@Service
public class CouponServiceImpl implements CouponService {


    private final  TargetMoneyCouponStrategyImpl targetMoneyCouponStrategy;

    private final  TimeCouponStrategyImpl timeCouponStrategy;
    private final CouponMapper couponMapper;

    private static List<CouponMatchStrategy> strategyList = new ArrayList<>();

    @Autowired
    public CouponServiceImpl(TargetMoneyCouponStrategyImpl targetMoneyCouponStrategy,
                             TimeCouponStrategyImpl timeCouponStrategy,
                             CouponMapper couponMapper) {
        this.couponMapper = couponMapper;
        this.targetMoneyCouponStrategy = targetMoneyCouponStrategy;
        this.timeCouponStrategy = timeCouponStrategy;
        strategyList.add(targetMoneyCouponStrategy);
        strategyList.add(timeCouponStrategy);
    }
    @Override
    public List<Coupon> getMatchOrderCoupon(OrderVO orderVO) {

        List<Coupon> hotelCoupons = getHotelAllCoupon(orderVO.getHotelId());

        List<Coupon> availAbleCoupons = new ArrayList<>();

        for (int i = 0; i < hotelCoupons.size(); i++) {
            for (CouponMatchStrategy strategy : strategyList) {
                if (strategy.isMatch(orderVO, hotelCoupons.get(i))) {
                    availAbleCoupons.add(hotelCoupons.get(i));
                }
            }
        }

        return availAbleCoupons;
    }
}
```

```
    @Override
    public List<Coupon>
getHotelAllCoupon(Integer hotelId) {
        List<Coupon> hotelCoupons =
couponMapper.selectByHotelId(hotelId);
        return hotelCoupons;
    }

    @Override
    public CouponVO
addHotelTargetMoneyCoupon(HotelTarge
tMoneyCouponVO couponVO) {
        Coupon coupon = new Coupon();

coupon.setCouponName(couponVO.getN
ame());

coupon.setDescription(couponVO.getDes
cription());

coupon.setCouponType(couponVO.getTy
pe());

coupon.setTargetMoney(couponVO.getTa
rgetMoney());

coupon.setHotelId(couponVO.getHotelId()
);

coupon.setDiscountMoney(couponVO.get
DiscountMoney());
        coupon.setStatus(1);
        int result =
couponMapper.insertCoupon(coupon);
        couponVO.setId(result);
        return couponVO;
    }
}
```

源代码解析 - data

- data.coupon.CouponMapper类
  - @Mapper 注解是mybatis的注解，是用来说明这个是一个Mapper，对应的xxxMapper.xml就是来实现这个Mapper。
  - @Repository 注解是Spring的注解，使用该注解把当前类注册成一个Java Bean，并将其标记为数据持久层。

源代码解析 - controller

- data.coupon.CouponMapper类

```
package com.example.hotel.data.coupon;

import com.example.hotel.po.Coupon;
import org.apache.ibatis.annotations.Mapper;
import org.springframework.stereotype.Repository;

import java.util.List;

@Mapper
@Repository
public interface CouponMapper {
    int insertCoupon(Coupon coupon);

    List<Coupon> selectByHotelId(Integer hotelId);
}
```

源代码解析 – dataImpl

- dataImpl.coupon.CouponMapper.xml文件
  - namespace: 接口的ID
  - id: 方法的ID

源代码解析 - controller

- dataImpl.coupon.CouponMapper.xml文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://
mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.example.hotel.data.coupon.CouponMapper">

    <insert id="insertCoupon" parameterType="com.example.hotel.po.Coupon"
            useGeneratedKeys="true" keyProperty="id">
        insert into
coupon(description,couponName,target_money,discount_money,start_time,end_tim
e,hotelId,couponType,discount,status)

values(#{description},#{couponName},#{targetMoney},#{discountMoney},#{startTim
e},#{endTime},#{hotelId},#{couponType},#{discount},#{status})
    </insert>  <!--  插入一个优惠券   -->

    <select id="selectByHotelId" resultMap="Coupon">
        select * from Coupon where hotelId=#{hotelId}
    </select>


</mapper>
```

```
<resultMap id="Coupon" type="com.example.hotel.po.Coupon">
        <result column="description" property="description"></result>
        <result column="id" property="id"></result>
        <result column="couponName" property="couponName"></result>
        <result column="hotelId" property="hotelId"></result>
        <result column="couponType" property="couponType"></result>
        <result column="discount" property="discount"></result>
        <result column="status" property="status"></result>
        <result column="target_money" property="targetMoney"></result>
        <result column="discount_money" property="discountMoney"></result>
        <result column="start_time" property="startTime"></result>
        <result column="end_time" property="endTime"></result>
    </resultMap>
```

思考题

1. REST风格的优缺点是什么？未来Web架构会朝什么方向发展。

本章总结

- 本章对HTTP协议的相关知识进行了简单的介绍
- 介绍了Tomcat和Servlet的基本原理
- 介绍了REST风格的由来、REST风格API规范和JSON格式的相关内容
- 讲解了Spring Boot的基本概念和使用方法
- 讲解了数据持久化基本概念，介绍了SQL、JDBC及其他一些相关工具
- 介绍了Mybatis的概念和用法
- 用一个Spring Boot+Mybatis的实际编程案例进一步讲解这两个框架的用法