第 5 章 CHAPTER 5 物联网应用层

5.1 物联网服务器

5.1.1 基于 Java 的 Web 服务器搭建

本书中采用 Java Web 技术，开发物联网服务器端软件。为实训项目提供门户展示、信息管理功能，同时为硬件端和移动端提供接口。基于 Java 的 Web 服务器搭建需要几个步骤，下面依次介绍。

1. JDK 的安装

JDK 的安装文件可以从 Oracle 公司的网站 https://www.oracle.com/下载。JDK 的安装步骤如下。

(1) 双击运行安装文件 jdk-9.0.4_windows-x64_bin.exe，会出现 Java SE 开发工具包安装向导，如图 5-1 所示。

图 5-1 Java SE 开发工具包安装向导

（2）单击图 5-1 中的“下一步”按钮，会出现 JDK 安装目录的选择界面，指定 JDK 安装目录为 C:\Program Files\Java\jdk-9.0.4，然后单击“下一步”按钮，如图 5-2 所示。

图 5-2　选择 JDK 安装目录

（3）出现 JDK 安装进度条，如图 5-3 所示。

图 5-3　JDK 安装进度条

（4）进度条结束后，出现指定 JRE 安装目录的界面。此处指定 JRE 的安装目录为 C:\Program Files\Java\jre-9.0.4，然后单击“下一步”按钮，如图 5-4 所示。

（5）选择好 JRE 路径，会出现 Java 安装进度条，如图 5-5 所示。

图 5-4 选择 JRE 安装路径

图 5-5 Java 安装进度条界面

(6) 显示 JDK 安装成功界面,如图 5-6 所示。

(7) JDK 安装完成后,配置环境变量。新建 JAVA_HOME 环境变量 JAVA_HOME=C:\Program Files\Java\jdk-9.0.4,修改 path 环境变量,在 path 变量尾部添加%JAVA_HOME%\bin。新建 classpath 环境变量 classpath=.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\tools.jar。

JDK 需要配置以上 3 个环境变量 JAVA_HOME、path 和 classpath; JDK1.5 版本之后可以不再设置 classpath,但建议保留 classpath 设置。

图 5-6　JDK 安装成功界面

2. Tomcat 的安装和使用

Tomcat 的安装文件可以从 Apache 公司的官方网站 http://tomcat.apache.org/index.html 下载，采用绿色版本 apache-tomcat-9.0.0.M9，解压到指定目录下即可。启动和关闭 Tomcat 服务器的文件位于 Tomcat 主目录下的 bin 文件夹下，文件名分别为 startup.bat 和 shutdown.bat。

(1) 执行 startup.bat，启动 Tomcat，如图 5-7 所示。

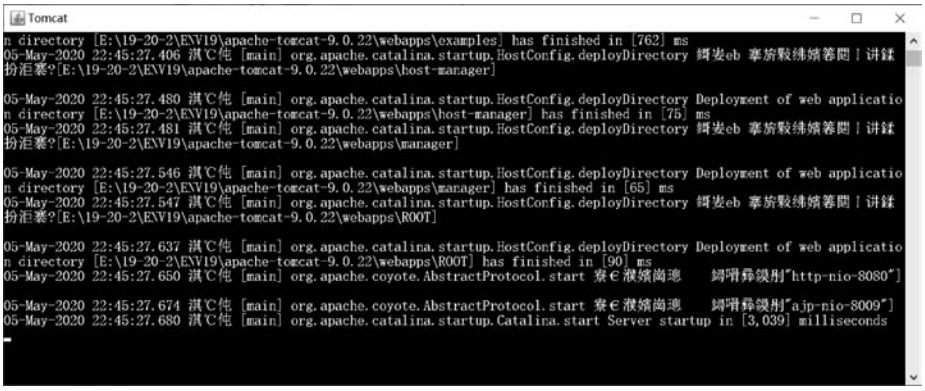

图 5-7　启动 Tomcat

(2) Tomcat 正常启动后，打开浏览器，在地址栏输入 URL：http://127.0.0.1:8080，将看到如图 5-8 所示界面。

(3) Tomcat 提供服务的默认端口号是 8080，当与其他应用程序的端口号发生冲

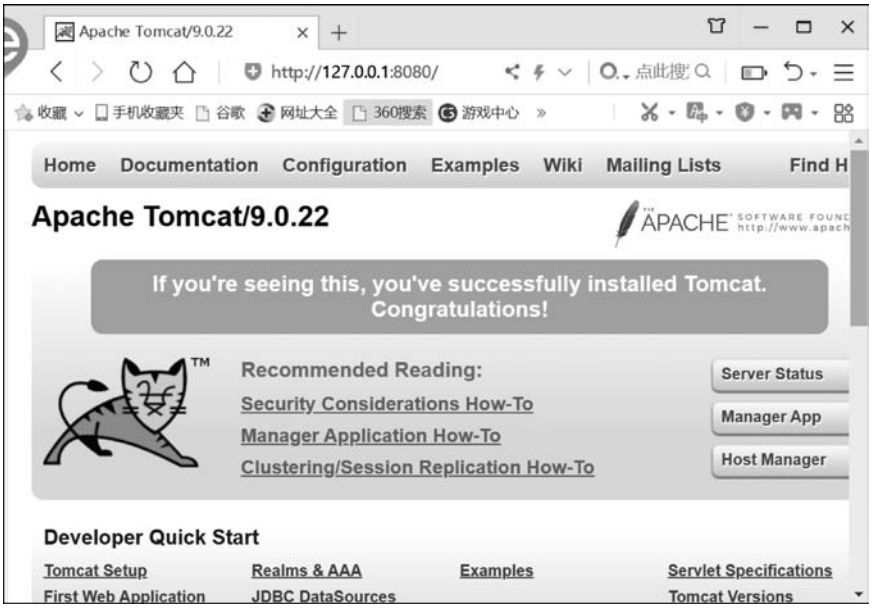

图 5-8 Tomcat 系统首页

突时，将无法正常启动。此时，可修改 Tomcat 的端口号为其他未被占用的端口号。修改 Tomcat 端口号的位置在 Tomcat 主目录下的 conf 文件夹下，文件名为 server.xml。可以用记事本等文本编辑工具将其打开，重新改写端口号，如图 5-9 所示。

图 5-9 改写 Tomcat 端口号

（4）Tomcat 的主目录下有若干目录，其用途如表 5-1 所示。

表 5-1　Tomcat 目录结构及用途

目　　录	用　　途
\bin	放置启动和关闭 Tomcat 的可执行文件和批处理文件
\lib	放置 Tomcat 运行所需要加载的 jar 包
\conf	放置 Tomcat 主要的配置文件
\logs	放置 Tomcat 日志文件
\temp	放置 Tomcat 运行时产生的临时文件
\webapps	放置 Web 应用的目录
\work	放置 JSP 页面转换成对应的 Servlet 的目录

3. Eclipse 的安装和使用

Eclipse 的安装文件 eclipse-inst-win64. exe 可以从 http://www. eclipse. org/downloads/下载，用户可以下载解压版使用。

（1）将 Eclipse 的解压版文件 eclipse-jee-oxygen-2-win32-x86_64. zip 解压之后，双击 eclipse. exe 图标，运行 Eclipse 集成开发环境，如图 5-10 所示。

图 5-10　解压并运行 Eclipse 集成开发环境

（2）打开 Eclipse Launcher 对话框，提示用户选择 Eclipse 的工作空间，如图 5-11 所示。

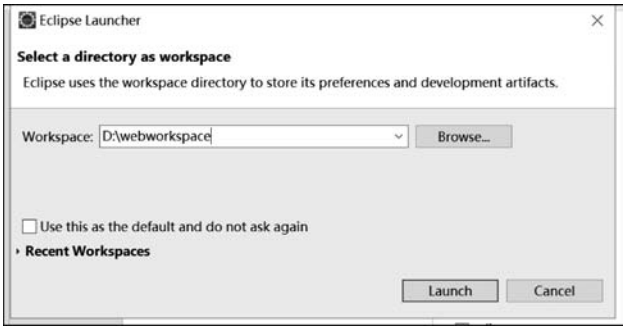

图 5-11　选择 Eclipse 的工作空间

（3）打开 Eclipse 后，可在 Servers 选项卡中，配置应用服务器 Tomcat，如图 5-12 所示。

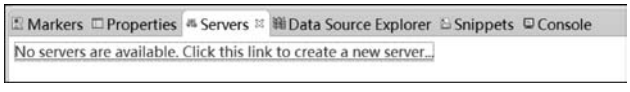

图 5-12　在 Eclipse 中配置应用服务器

（4）单击创建新服务器链接，会出现如图 5-13 所示的对话框，在对话框中选择解

压或安装过的 Tomcat 应用服务器。

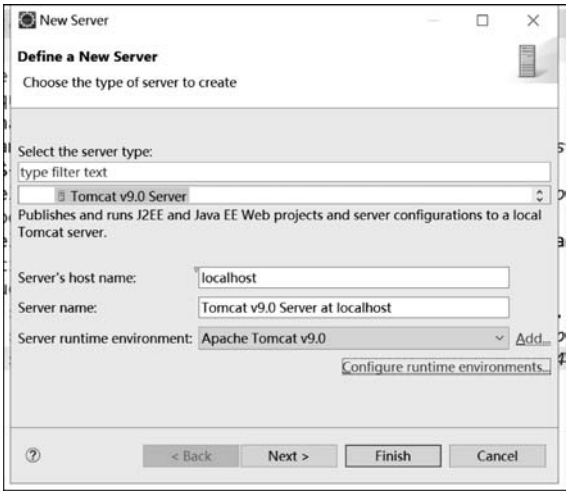

图 5-13 在 Eclipse 中关联 Tomcat 应用服务器

(5) 在 Eclipse 中关联 Tomcat 后,需要进行服务器名、主机名、运行时环境、服务器位置、部署路径等配置,如图 5-14 所示。

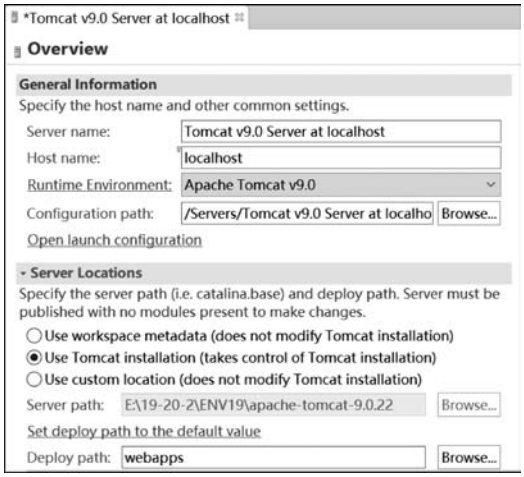

图 5-14 Tomcat 服务器配置

关于 Eclipse 中集成的 JDK 版本和安装位置,可以在如图 5-15 所示的对话框中进行查看和修改。

可在如图 5-16 所示的对话框中查看和编辑 Eclipse 中的运行时环境关联情况,即 Tomcat 应用服务器的版本选择。

单击图 5-16 中的 Edit 按钮,即可进入图 5-17 所示的对话框,编辑 Eclipse 的运行时环境配置,包括 Tomcat 的版本、位置和 JDK 版本。

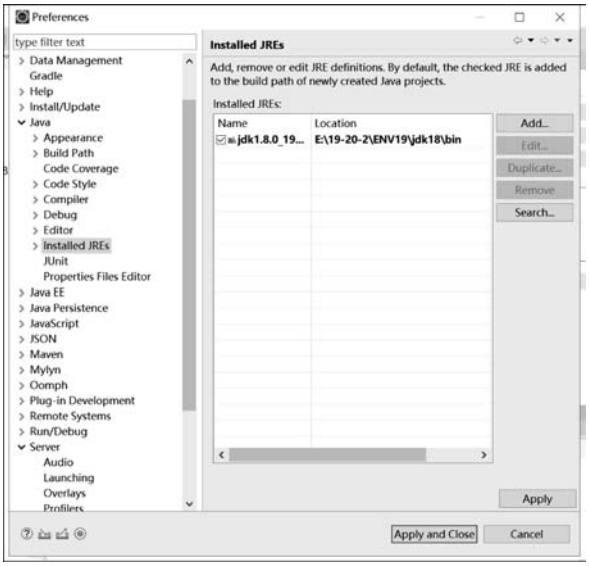

图 5-15　Eclipse 中的 JDK 版本和安装位置

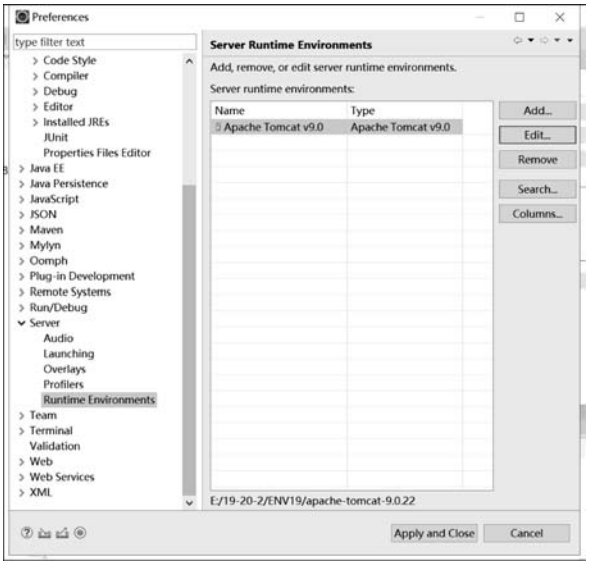

图 5-16　Eclipse 中的运行时环境设置

4. MySQL 的安装和配置

本书的实训项目“智能停车场”，将车位信息数据、用户数据等存储于 MySQL 数据库。

（1）从 MySQL 数据库的官网 http://www.mysql.com 下载与操作系统匹配的 MySQL 安装文件，单击后进入的首页如图 5-18 所示。

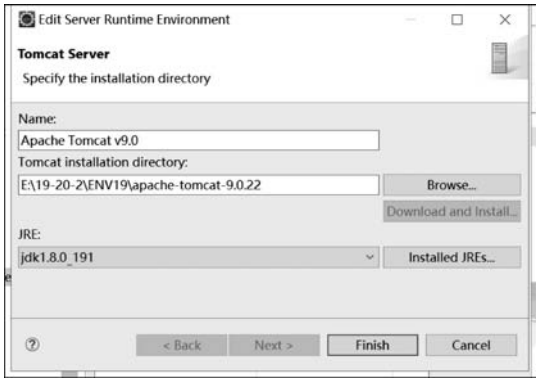

图 5-17　编辑服务器运行时环境

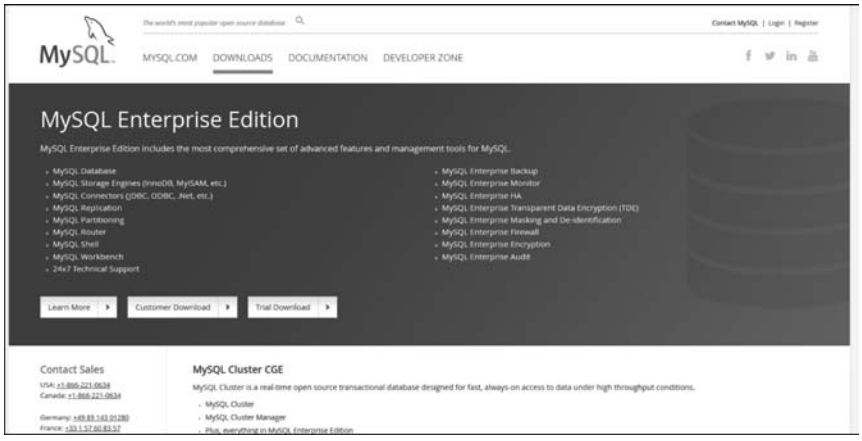

图 5-18　MySQL 官网首页

（2）单击 DOWNLOADS→Community，选择 MySQL Community Server，如图 5-19 所示。

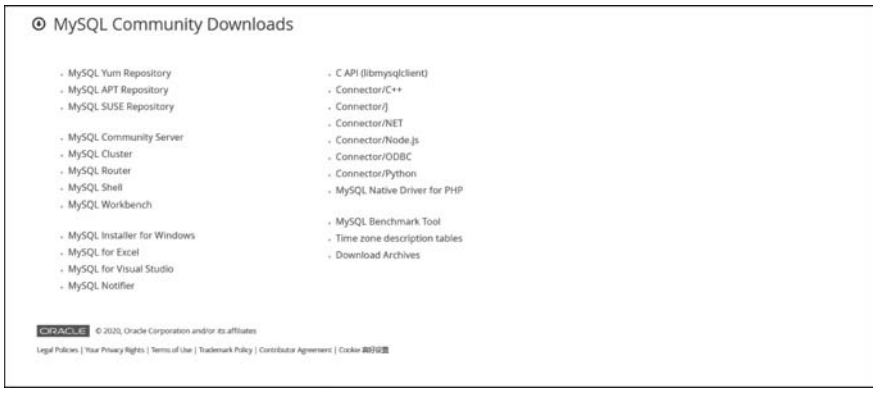

图 5-19　选择下载 MySQL Community Server

（3）找到 Recommended Download，单击 Go to Download Page，如图 5-20 所示。

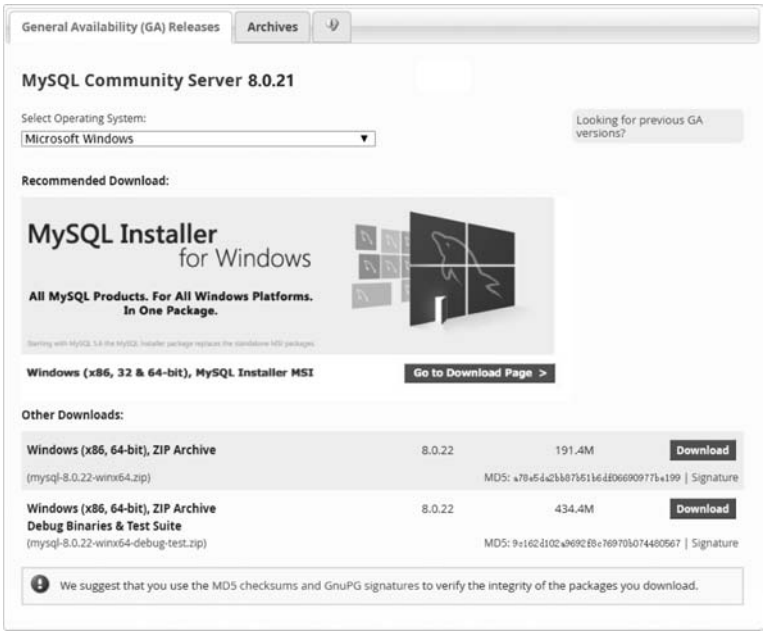

图 5-20　下载 Windows 版本 MySQL 安装文件

（4）进入下载界面，如图 5-21 所示；选择 No thanks，just start my download 开始下载。

图 5-21　MySQL 下载界面

（5）在图 5-22 所示的界面中，选择 MySQL 安装类型为 Developer Default，单击 Next 按钮，进入下一步。

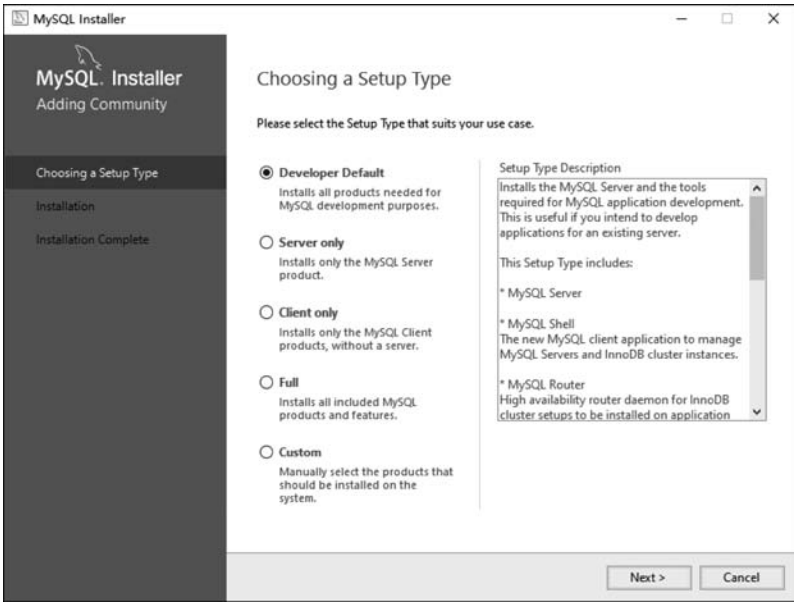

图 5-22　选择 MySQL 安装类型

（6）在如图 5-23 所示的界面中，检查 MySQL 的安装条件。如需要 Microsoft Visual C++包的支持，则选择补充安装。

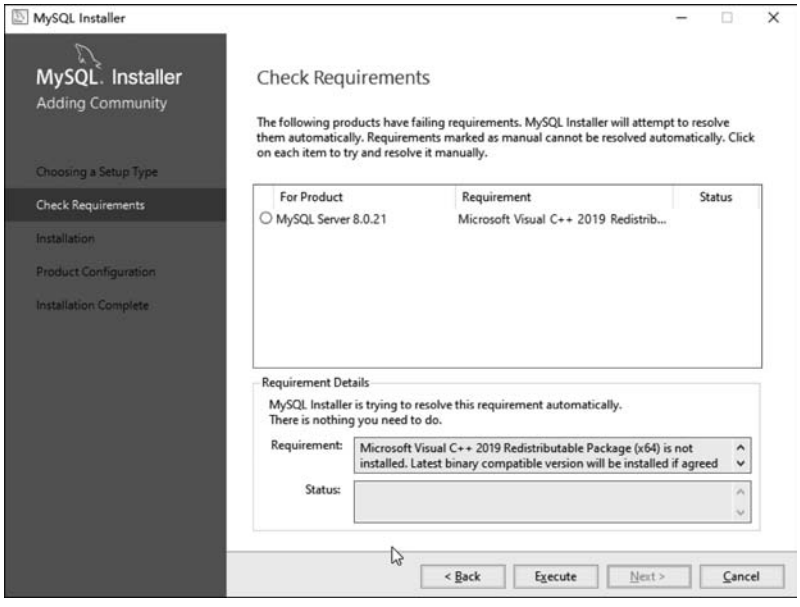

图 5-23　检查 MySQL 安装条件

（7）在图 5-24 所示的界面中，单击 Execute 按钮执行安装，在后续的界面中，均可默认单击 Next 按钮，如图 5-25 所示。

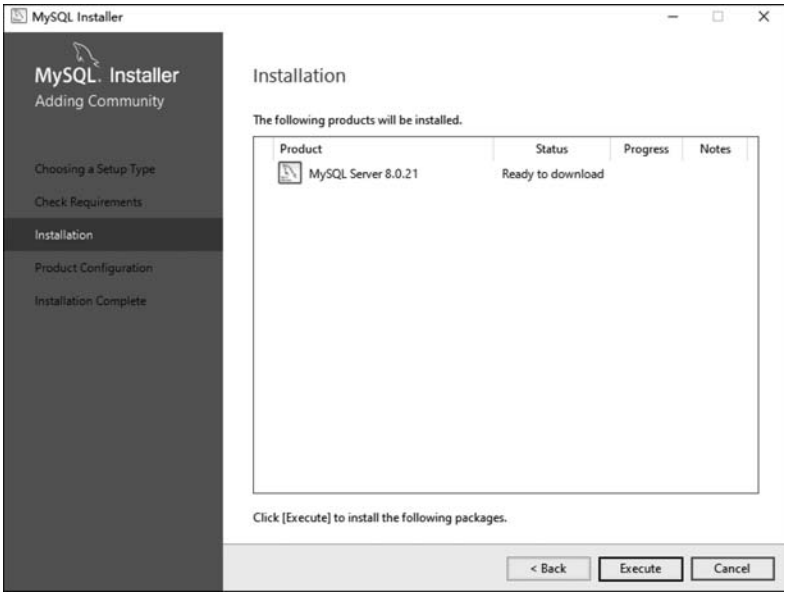

图 5-24　Installation 界面

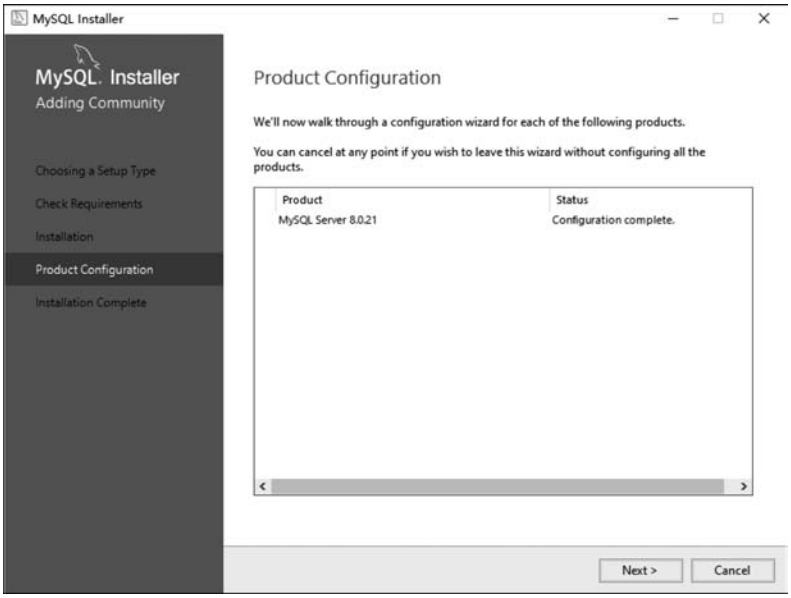

图 5-25　Product Configuration 界面

（8）在如图 5-26 所示的 High Availability 界面中，选中 Standalone MySQL Server/Classic MySQL Replication 单选按钮。

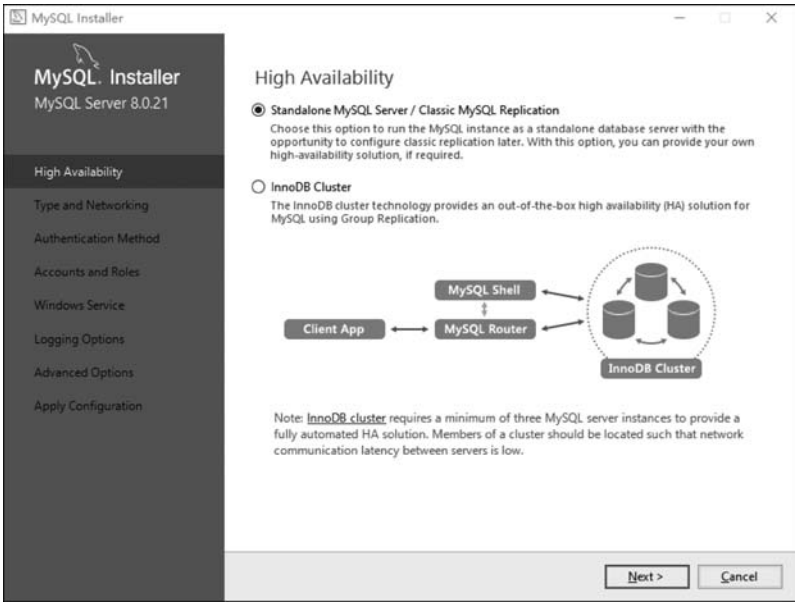

图 5-26 High Availability 界面

(9) 在如图 5-27 所示的界面中,设置 MySQL 的服务配置类型和网络设置。

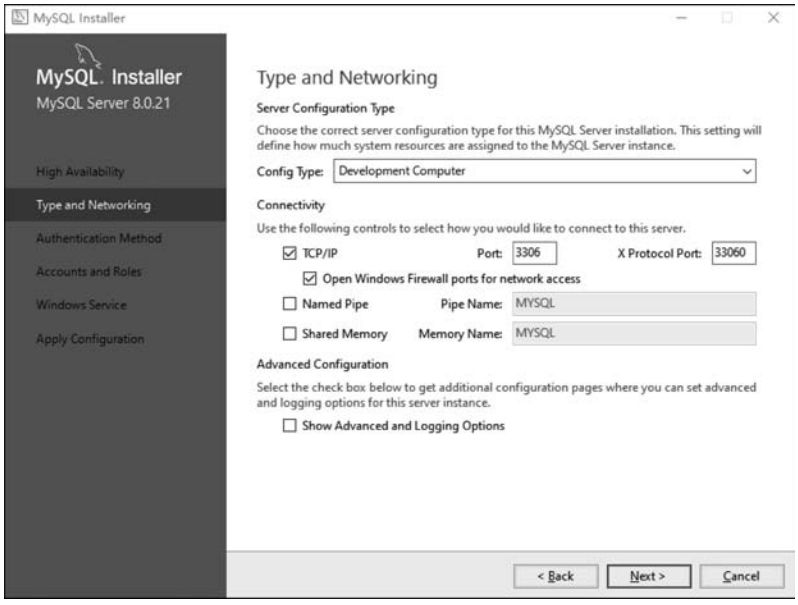

图 5-27 网络设置对话框

(10) 在如图 5-28 所示的界面中,设置 MySQL 的 root 用户密码,然后单击 Next 按钮,进入下一步。

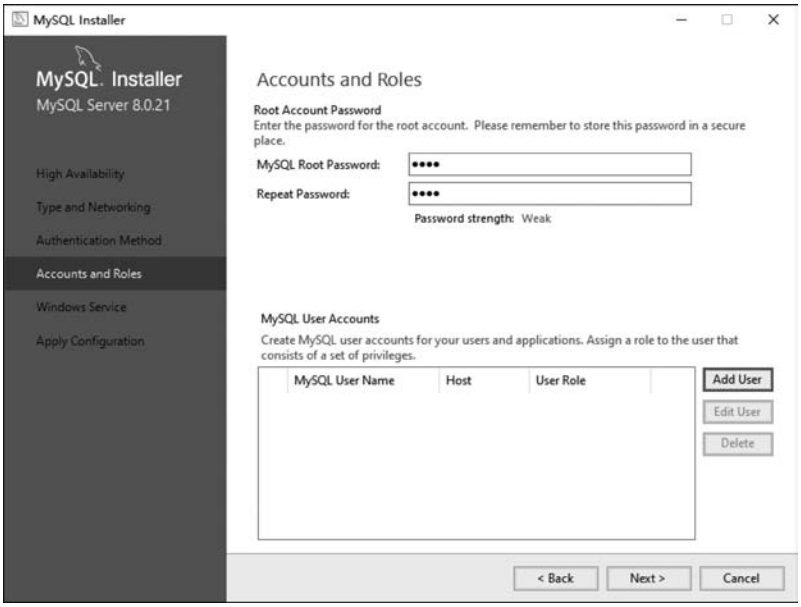

图 5-28　root 密码设置

（11）在图 5-29 所示的界面中，将 MySQL 服务配置为 Windows 服务，并配置在系统启动时自动启动服务，启动服务时使用标准系统账户等信息。

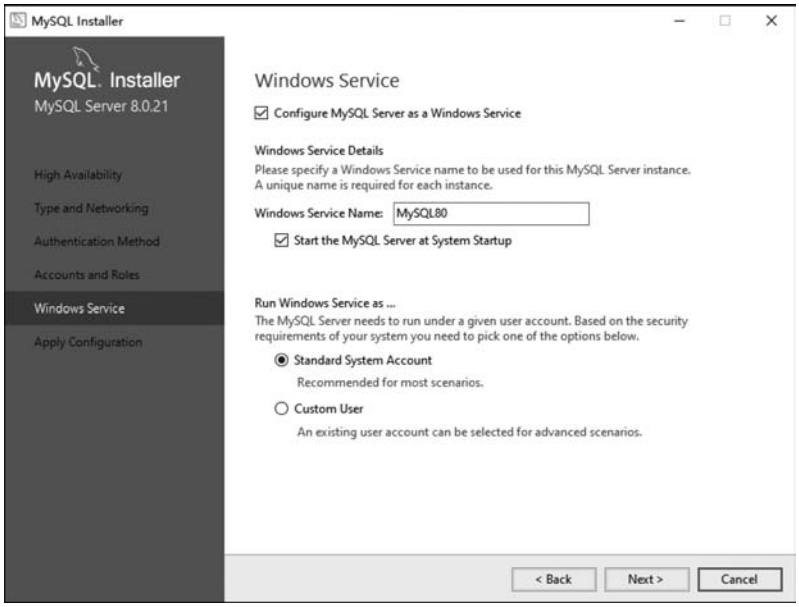

图 5-29　Windows 服务设置

（12）在图 5-30 所示的界面中，单击 Execute 按钮，即可由系统进行关闭现有服

务、写配置文件、更新防火墙、启动服务、应用安全设置等安装进程了。全部执行完毕后会出现图 5-31 所示的界面。

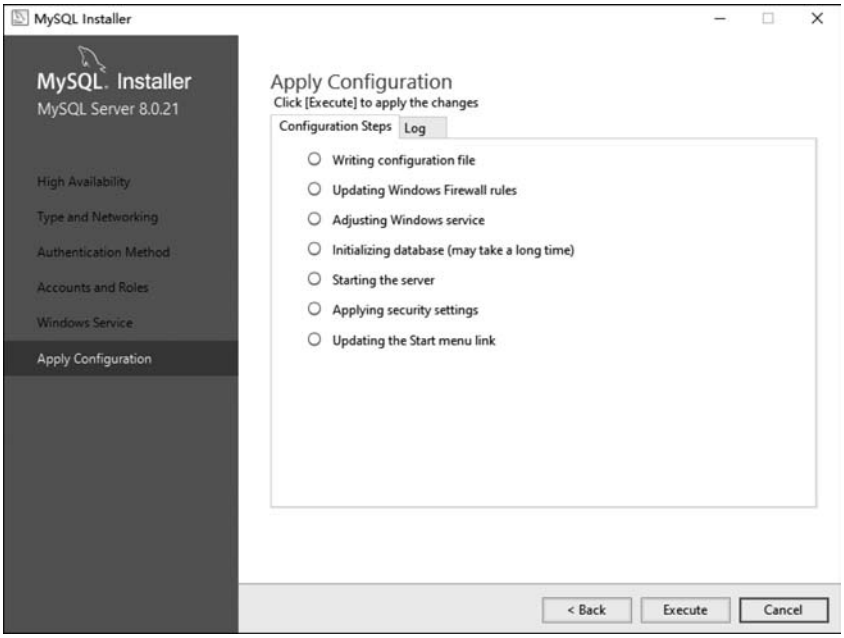

图 5-30 执行安装过程

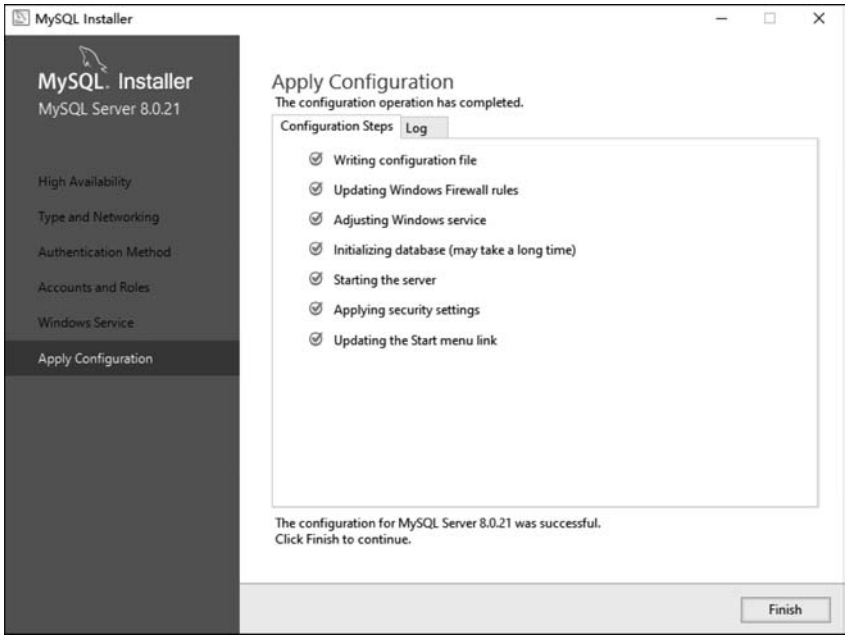

图 5-31 完成应用配置

(13) 进入产品配置界面,如图 5-32 所示,安装完成界面如图 5-33 所示。

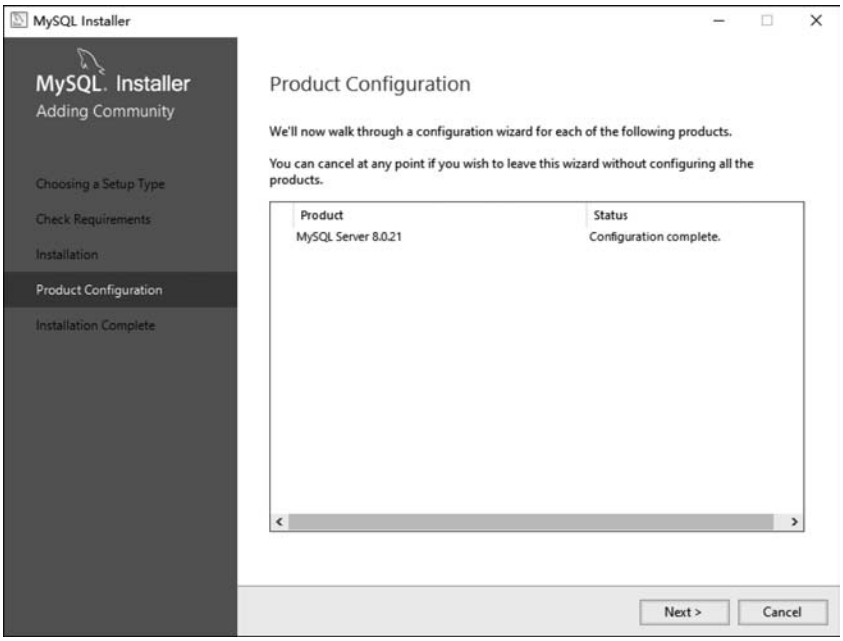

图 5-32　Product Configuration 界面

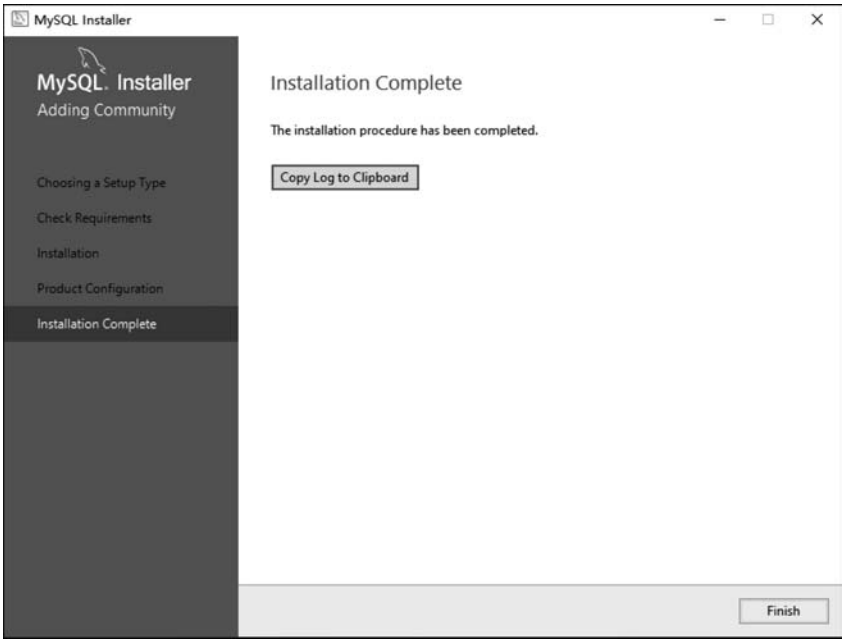

图 5-33　MySQL 安装完成界面

(14) 在图 5-34 所示的界面中,通过 Windows 的服务组件管理 MySQL 服务的启

动方式，可设置为自动或手动，可设置为本地系统账户登录。

图 5-34 测试连接 MySQL 服务

(15) 安装完成后进入 MySQL 的安装目录，进入 MySQL Sever，其目录下的文件如图 5-35 所示。

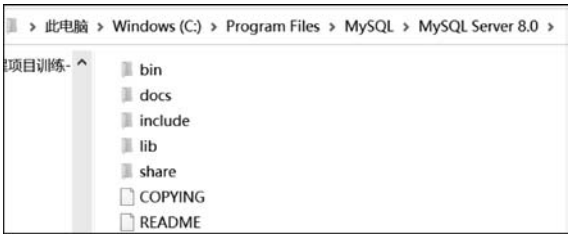

图 5-35 MySQL Server 8.0 目录结构

(16) bin 目录下保存了 MySQL 常用的命令工具以及管理工具、data 目录是 MySQL 默认用来保存数据文件以及日志文件的地方(我的因刚安装还没有 data 文件夹)、docs 目录下是 MySQL 的帮助文档、include 目录和 lib 目录是 MySQL 所依赖的头文件以及库文件、share 目录下保存目录文件以及日志文件。

进入 bin 目录，按住 Shift 键，然后单击鼠标右键可以选择在该目录下打开命令窗口，或者在地址栏中输入 cmd 进入命令窗口，输入 mysql-u root-p 后按 Enter 键，然后会提示输入密码，输入密码后就会进入 MySQL 的操作管理界面。

5. Navicat 的安装和使用

Navicat 是一套数据库开发工具，用户可以利用该工具同时连接 MySQL、MariaDB、MongoDB、SQL Server、Oracle、PostgreSQL 和 SQLite 数据库。它与 Amazon RDS、Amazon Aurora、Amazon Redshift、Microsoft Azure、Oracle Cloud、MongoDB Atlas、阿里云、腾讯云、华为云等云数据库兼容，通过 Navicat 可以在可视化界面下快速轻松地创建、管理和维护数据库，下载网址是 https://www.navicat.com.cn/，其安装过程比较简单，在此不再赘述。

5.1.2　数据库访问工具类的封装

服务器采用 JDBC 技术与数据库连接，并访问数据库内容。JDBC(Java Database Connectivity，Java 数据库连接)是一种用于数据库访问的 Java API(Application Programming Interface，应用程序设计接口)，由一组用 Java 语言编写的类和接口组成。有了 JDBC，就可以用纯 Java 语言和标准的 SQL 语句编写完整的数据库应用程序，并且真正实现软件的跨平台性。

简单地说，JDBC 能完成下列三件事。

(1) 为同一个数据库建立连接。

(2) 向数据库发送 SQL 语句。

(3) 处理数据库返回的结果。

为便于项目开发的层次化和模块化，将服务器访问数据库的流程封装为工具类 DBTool，将数据库的连接、关闭、查询、更新、分页显示方法封装于 DBTool 类，其类图如图 5-36 所示。

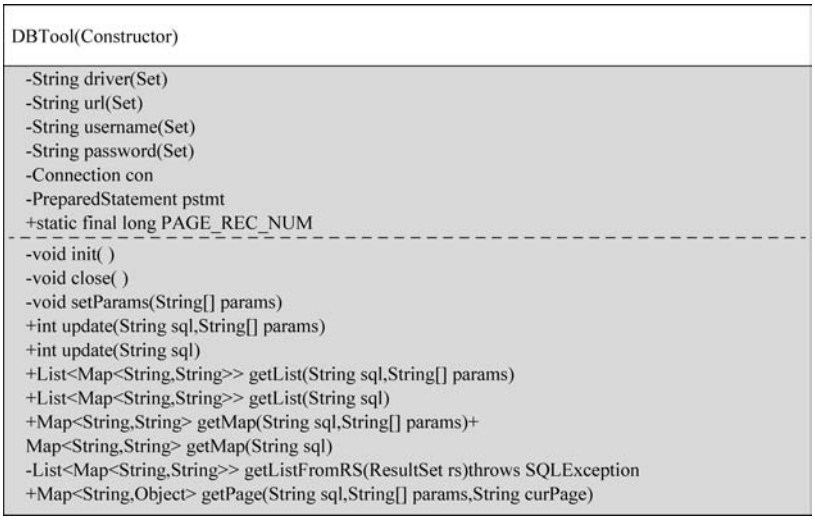

图 5-36　DBTool 类图

该工具类将访问数据库的连接关闭、参数设置，结果集到 Java 集合类对象的转换都定义为私有方法，仅将查询方法、更新方法、和分页显示方法设置为公有方法，供其他层次的开发者调用，从而提高了代码复用率，进而提高了项目开发效率。

DBTool 代码如下。

```
package util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


public class DBTool {
    private String driver;
    private String url;
    private String username;
    private String password;
    private Connection con;
    private PreparedStatement pstmt;
    public static final long PAGE_REC_NUM = 8;
    public void setDriver(String driver) {
        this.driver = driver;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public DBTool() {
        driver = "com.mysql.cj.jdbc.Driver";
        url = "jdbc:mysql://localhost:3306/meal?serverTimezone=UTC";
        username = "root";
        password = "root";
    }
```

```
    private void init() {

        try {
            Class.forName(driver);
            con = DriverManager.getConnection(url, username, password);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
    private void close() {
        if(pstmt!= null)
            try {
                pstmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        if(con!= null)
            try {
                con.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
    }
    private void setParams(String[] params) {
        if(params!= null) {
            for(int i = 0;i < params.length;i++) {
                try {
                    pstmt.setString(i + 1, params[i]);
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
    public int update(String sql,String[] params) {
        int result = 0;
        init();
        try {
            pstmt = con.prepareStatement(sql);
            setParams(params);
            result = pstmt.executeUpdate();
```

```
            } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            close();
        }
        return result;
    }
    public int update(String sql) {
        return update(sql, null);
    }
    public List<Map<String, String>> getList(String sql,String[] params){
        List<Map<String, String>> list = null;
        init();
        try {
            pstmt = con.prepareStatement(sql);
            setParams(params);
            ResultSet rs = pstmt.executeQuery();
            list = getListFromRS(rs);
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            close();
        }
        return list;
    }
    private List<Map<String, String>> getListFromRS(ResultSet rs) throws SQLException {
        List<Map<String, String>> list = new ArrayList<Map<String,String>>();
        ResultSetMetaData rsmd = rs.getMetaData();
        while(rs.next()) {
            Map<String, String> m = new HashMap<String, String>();
            for(int i = 1;i <= rsmd.getColumnCount();i++) {
                String colName = rsmd.getColumnLabel(i);
                String value = rs.getString(colName);
                if(value != null) {
                    m.put(colName, value);
                }
            }
            list.add(m);
        }
        return list ;
    }
    public List<Map<String, String>> getList(String sql){
        return getList(sql, null);
```

```
    }
    public Map<String, String> getMap(String sql,String[] params){
        Map<String, String> m = new HashMap<String, String>();
        List<Map<String, String>> list = getList(sql, params);
        if(list!= null&&list.size()!= 0) {
            m = list.get(0);
        }
        return m;
    }
    public Map<String,String> getMap(String sql){
        return getMap(sql, null);
    }
    public Map<String,Object> getPage(String sql,String[] params,String curPage){
        Map<String, Object> page = new HashMap<String, Object>();
        String newSql = sql + " limit " + (Long.parseLong(curPage) - 1) * PAGE_REC_NUM
+ "," + PAGE_REC_NUM;
        List<Map<String, String>> pageList = getList(newSql, params);
        sql = sql.toLowerCase();
        String countSql = "";
        if(sql.indexOf("group")>=0) {
            countSql = "select count(*) as tempNum from(" + sql + ") as temp";
        }
        else {
             countSql = "select count(*) as tempNum " + sql.substring(sql.indexOf
("from"));
        }
        String count_s = (String)getMap(countSql, params).get("tempNum");
        long count = Long.parseLong(count_s);
        long totalPage = 0;
        if(count % PAGE_REC_NUM == 0)
            totalPage = count/PAGE_REC_NUM;
        else
        page.put("list", pageList);
        page.put("totalPage", totalPage);
        page.put("recNum", PAGE_REC_NUM);
        return page;
    }
}
```

5.1.3　服务器端数据管理功能

服务器端的数据管理功能涉及项目的具体业务逻辑。例如，智能停车场系统，其主要业务逻辑包括车辆进出场的 RFID 刷卡识别、进出场时长统计、计费管理等。如果是环境监测系统，则主要的业务逻辑包括环境数据的显示、实时更新和报表统计。

无论项目的具体业务逻辑如何，在服务器端对项目数据的管理，基本上涵盖增加、删除、修改、查询这些典型的业务逻辑。下面以用户管理为例，阐述服务器端的数据管理功能。

（1）在 Eclipse 中新建动态 Web 工程，如图 5-37 所示。

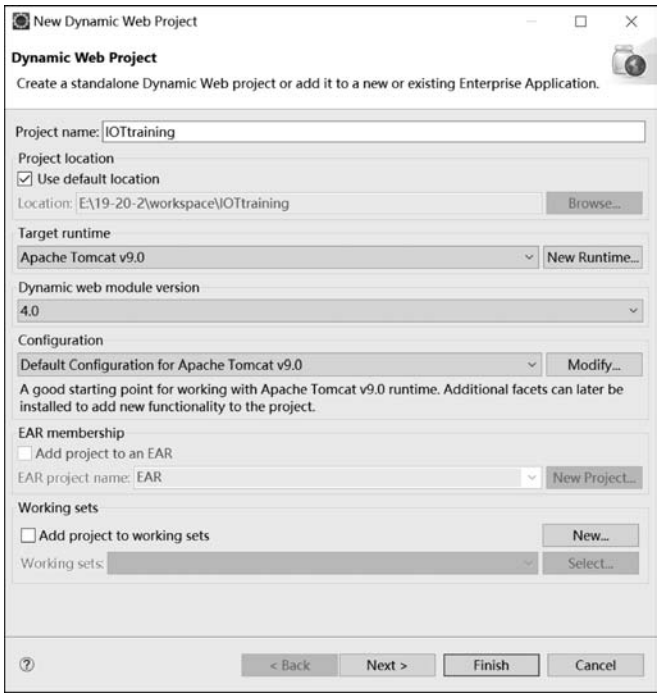

图 5-37　在 Eclipse 中新建动态 Web 工程

（2）单击 Next 按钮，将输出文件夹改为 WebContent\WEB-INF\classes，如图 5-38 所示。

（3）单击 Finish 按钮，即可新建 Web 工程。在 WEB-INF 的 lib 文件夹下复制 MySQL 驱动 jar 包，如图 5-39 所示。

（4）在工程名上右击，在弹出的快捷菜单中选择 build path→cofigure build path 选项，进入如图 5-40 所示的界面。

（5）单击 Add JARs 按钮，进入 jar 包选择和添加界面，如图 5-41 所示。

（6）导入 jar 包之后，单击 OK 按钮，完成数据库驱动包的导入。

下面即可新建工具类 DBTool，在工程目录树的 src 处右击，新建 class，指定包名和类名，如图 5-42 所示。

编写 DBTool 的代码，如 5.1.2 小节数据库访问工具类的封装。下面就可以开始用户管理功能的实现了。用户管理功能基于数据库中的 user 表。在 MySQL 中的数据库名为 parking。

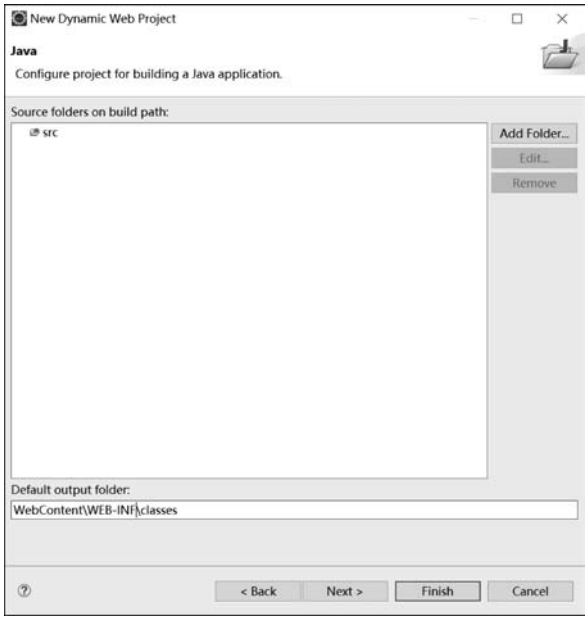

图 5-38　修改 Web 工程输出文件夹

图 5-39　复制 MySQL 驱动包

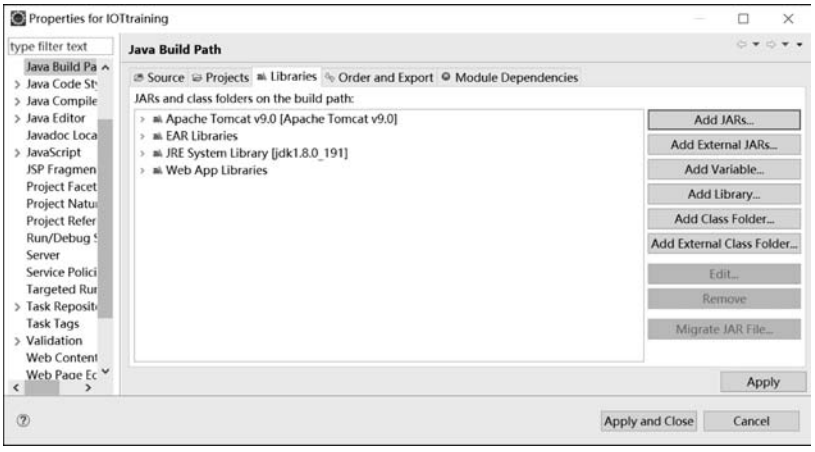

图 5-40　configure build path 界面

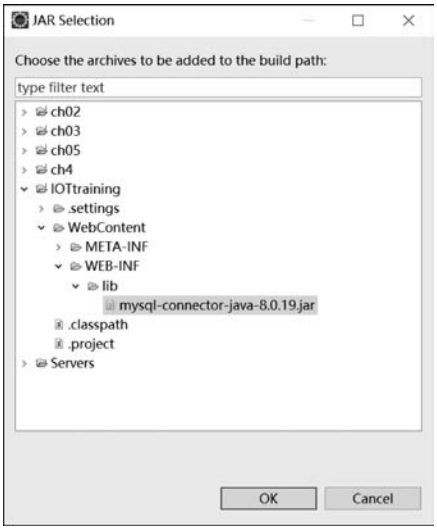

图 5-41 选择并添加 jar 包

图 5-42 创建 DBTool 类

在 parking 数据库中的 user 表，定义了智能停车场的用户信息，用户在停车场中凭 RFID 刷卡出入，如图 5-43 所示。

cardid	name	count	license	phone	opendate
2B2F	raindy	5	冀A1234	987654321	2017-10-25 15:28:03
3F5N	123	5	京A0000	1234567890	2017-10-24 16:19:08

图 5-43 parking 数据库中的 user 表内容

首先创建 UserService 类。封装停车场用户管理的业务逻辑，包括用户开卡、注销、修改停车卡信息和查询停车卡信息 4 个业务逻辑。UserService 位于 model 包下，其类图如图 5-44 所示。

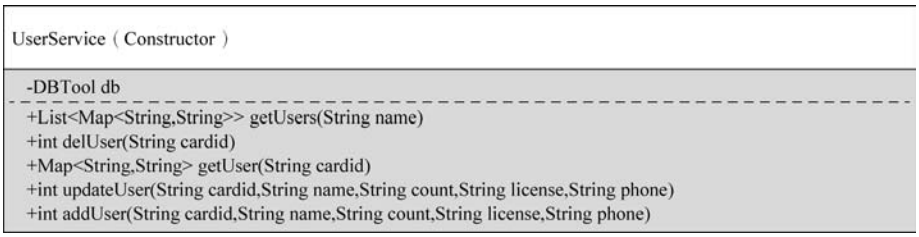

图 5-44　UserService 类图

UserService 代码如下所示。

```
package model;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import tools.DBTool;

public class UserService {
    private DBTool db;

    public UserService() {
        db = new DBTool();
    }
    public List<Map<String, String>> getUsers(String name){
        List<Map<String, String>> list = new ArrayList<Map<String,String>>();
        String sql = "select * from user";
        String[] params = null;
        if(name!= null) {
            sql = sql + " where name like ?";
            params = new String[] {"%" + name + "%"};
        }
        list = db.getList(sql, params);
        return list;
    }
    public int delUser(String cardid) {
        String[] params = {cardid};
        String sql = "delete from user where cardid = ?";
        return db.update(sql, params);
    }
```

```
    public int addUser(String cardid,String name,String count,String license,String
phone) {
        String sql = "select * from user where cardid=?";
        List<Map<String, String>> list = db.getList(sql,new String[] {cardid});
        if(list.size()!=0)
            return 0;
        else {
            sql = "insert into user values(?,?,?,?,?,now())";
            return db.update(sql, new String[] {cardid,name,count,license,phone});
        }
    }
    public Map<String, String> getUser(String cardid){
        Map<String, String> user = new HashMap<String, String>();
        String sql = "select * from user where cardid = ?";
        user = db.getMap(sql, new String[] {cardid});
        return user;
    }
    public int updateUser(String cardid,String name,String count,String license,String
phone) {
        String sql = "update user set name=?,count =?,license=?,phone=?,opendate
=now() where cardid=?";
        return db.update(sql, new String[] {name,count,license,phone,cardid});
    }
}
```

查询用户信息的界面为 list_user.jsp,代码如下。

```
<%@page import="java.util.Map"%>
<%@page import="java.util.List"%>
<%@page import="model.UserService"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<% String s_un = request.getParameter("name");
UserService us = new UserService();
List<Map<String,String>> users = us.getUsers(s_un);
%>
<div align="center">
<form action="list_users.jsp" method="post">
<input type="text" name="s_un" placeholder="请输入用户名查询">
```

```
<input type = "submit" value = "搜索">
</form>
<a href = "add_user.html">添加停车用户卡</a>
<table border = "1">
<tr>
<th>序号</th>
<th>卡号</th>
<th>用户名</th>
<th>停车次数</th>
<th>车牌号</th>
<th>电话</th>
<th>办卡时间</th>
<th>操作</th>
</tr>
<%
int num = 0;
for(Map<String,String> user:users){
    num++;
    %>
    <tr>
    <td><% = num %></td>
    <td><% = user.get("cardid") %></td>
    <td><% = user.get("name") %></td>
    <td><% = user.get("count") %></td>
    <td><% = user.get("license") %></td>
    <td><% = user.get("phone") %></td>
    <td><% = user.get("opendate") %></td>
    <td><a href = "del_user.jsp?cardid = <% = user.get("cardid") %>">删除</a>
    <a href = "edit_user.jsp?cardid = <% = user.get("cardid") %>">修改</a></td>
    </tr>
    <%
}
%>
</table>
</div>
</body>
</html>
```

删除用户信息的界面为 del_user.jsp，代码如下。

```
<%@page import = "java.util.Map" %>
<%@page import = "java.util.List" %>
<%@page import = "model.UserService" %>
<%@ page language = "java" contentType = "text/html; charset = UTF - 8"
    pageEncoding = "UTF - 8" %>
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="UTF-8">
<title>删除用户</title>
</head>
<body>
<%
String cardid = request.getParameter("cardid");
UserService us = new UserService();
int r = us.delUser(cardid);
if(r==1)
    out.println("删除用户成功!");
else
    out.println("删除用户失败");
%>
<a href = "list_users.jsp">返回用户列表</a>
</body>
</html>
```

用户的增加和修改功能,用户可以参照以上的用户列表和用户删除功能,自行实现,在此不再赘述。

5.1.4 为硬件端和移动端提供服务

作为物联网服务器,除了门户展示和信息管理功能之外,另一个重要功能是为物联网系统中的硬件端和移动端提供服务,使得硬件端实时采集的数据可以上传至服务器持久化存储,并由服务器进行后续的数据分析处理。同时使得移动端可以通过服务器实时查看数据变化,或发送指令给服务器,硬件端查询后,执行器件做出相应的改变。

下面以智能停车场的车位泊车信息为例,实现硬件端接口和移动端接口。智能停车场的车位信息对应数据库中的 psd 表,其表结构如图 5-45 所示。

Name	Type	Length	Decimals	Not null	Virtual	Key	Comment
node	int	1	0	☑	☐	🔑1	
status	varchar	6	0	☑	☐		

图 5-45 psd 表结构

其中,node 表示停车位节点,status 表示停车位状态。每个停车位上方部署有超声测距传感器,测量传感器与地面距离。没有车辆泊车的情况下,距离约为 3.5m,当有车辆泊车的情况下,测距结果小于 2.5m。依据这个原理进行车位空闲和占用的判断。硬件端会将测距结果发送给服务器,服务器端则修改 psd 表中的 status 状态,从而存储车位的泊车状态。

移动端通过网络通信框架访问服务器端，查询 psd 表中对应的车位节点的 status 状态，从而实现用户通过 Android APP 远程查看车位占用情况。

因此，服务器提供两套接口，分别为硬件端和移动端。硬件端组成 ZigBee 网络后，终端节点部署于每个车位上方，I/O 端口连接超声测距传感器，检测车位状态，发送给协调器，协调器将信息发送至服务器。协调器与服务器端的接口采用 Servlet 编写，代码如下。

Coord.java

```
package parkinglot;

import java.io.IOException;
import java.sql.Date;
import java.sql.Timestamp;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Iterator;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import utils.DBUtil;

/**
 * Servlet implementation class Coord
 */
@WebServlet("/Coord")
public class Coord extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Coord() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
```

```
     * /
    protected void doGet ( HttpServletRequest request, HttpServletResponse response )
throws ServletException, IOException {
        DBUtil db;
        db = new DBUtil();
        String sql;

        String cardId = null;
        cardId = request.getParameter("card");

        String role = null;
        role = request.getParameter("role");

        String dis = null;
        dis = request.getParameter("dis");

        String temp = null;
        String hu = null;
        temp = request.getParameter("temp");
        hu = request.getParameter("hu");
        System.out.println("yes");

        if(cardId != null) {
            if(role != null) {
                if(role.equals("enter")) {
                  sql = "insert into card(cardid,starttime,status) VALUES('" + cardId
+ "',now(),'泊车');";
                    db.update(sql);
                    sql = "select * from card where cardid = '" + cardId + "';";
                    List < Map < String, String >> list = db.getList(sql);
                    Iterator < Map < String, String >> iter = list.iterator();

                    //inverse control relay to control machine
                    if (list.isEmpty()){
                        response.getWriter().append("0");
                    } else {
                        response.getWriter().append("1");
                    }
                } else if (role.equals("exit")) {
                    //update endtime
                    sql = "update card set endtime = now() where cardid = '" + cardId + "';";
                    db.update(sql);

                    //acquire charge rule -- fee
                    sql = "select fee from chargerule;";
                    Map < String,String > res = db.getMap(sql);
```

```
                    String feeString = res.get("fee");
                    int fee = Integer.parseInt(feeString);

                    int charge = 0;

                    //acquire timestamp through cardid
                    sql = "select * from card where cardid = '" + cardId + "';";
                    res = db.getMap(sql);
                    //calculate min
                    try {
                        Timestamp times = string2Time(res.get("starttime"));
                        Timestamp timee = string2Time(res.get("endtime"));

                        long nd = 1000 * 24 * 60 * 60;
                        long nh = 1000 * 60 * 60;
                        long nm = 1000 * 60;
                        // ms
                        long diff = timee.getTime() - times.getTime();
                        // min
                        long min = diff % nd % nh / nm;
                                charge = (int) ((min/15) * fee);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }

                    //update status and charge
                     sql = "update card set status = '离开',charge = '" + charge + "'
where cardid = '" + cardId + "';";
                    db.update(sql);

                     request.getRequestDispatcher("UpdateUser?cardid = " + cardId).
forward(request, response);
                }
            }
        }

        if(dis != null && role != null) {
            int disint = Integer.parseInt(dis);
            //no car
            if(disint >= 100) {
                sql = "update psd set status = '空闲' where node = '" + role + "';";
            } else {
                sql = "update psd set status = '泊车' where node = '" + role + "';";
            }
            db.update(sql);
        }
```

```
        if(temp != null && hu != null && role != null) {
            System.out.println("温湿度");
            sql = "insert into dht11(temp,hu,time,node) VALUES('" + temp + "','" + hu
+ "',now(),'" + role + "');";
            db.update(sql);
        }

    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }


    /**
    * method 将字符串类型的日期转换为一个 Date(java.sql.Date)
    * @param dateString 需要转换为 Date 的字符串
    * @return dataTime Date
    */

    public final static java.sql.Date string2Date(String dateString)
    throws java.lang.Exception {
    DateFormat dateFormat;
    dateFormat = new SimpleDateFormat("yyyy-MM-dd", Locale.ENGLISH);
    dateFormat.setLenient(false);
    java.util.Date timeDate = dateFormat.parse(dateString);           //util 类型
    java.sql.Date dateTime = new java.sql.Date(timeDate.getTime());   //sql 类型
    return dateTime;
    }

    /**
     * method 将字符串类型的日期转换为一个 timestamp(时间戳记 java.sql.Timestamp)
     * @param dateString 需要转换为 timestamp 的字符串
     * @return dataTime timestamp
     */

    public final static java.sql.Timestamp string2Time(String dateString)
            throws java.text.ParseException {
            DateFormat dateFormat;
            dateFormat = new SimpleDateFormat("yyyy-MM-dd kk:mm:ss.SSS", Locale.
ENGLISH);       //设定格式
```

```
                //dateFormat = new SimpleDateFormat("yyyy-MM-dd kk:mm:ss", Locale.
ENGLISH);
            dateFormat.setLenient(false);
            java.util.Date timeDate = dateFormat.parse(dateString);     //util 类型
            java.sql.Timestamp dateTime = new java.sql.Timestamp(timeDate.getTime());
                                    //Timestamp 类型,timeDate.getTime()返回一个 long 型
            return dateTime;
            }

}
```

移动端采用 Android 技术实现,用户通过移动端远程查看车位状态。移动端通过网络请求,访问服务器。服务器端为移动端开发提供的接口代码如下。

QueryAllPSDData.java

```
package parkinglot;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import bean.PSD;
import utils.DBUtil;

/**
 * Servlet implementation class QueryAllUsers
 */
@WebServlet("/QueryPSDData")
public class QueryPSDData extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public QueryPSDData() {
        super();
        // TODO Auto-generated constructor stub
    }
```

```
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doPost(request, response);
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");
        response.setCharacterEncoding("utf-8");

        List<PSD> psdList = new ArrayList<PSD>();

        DBUtil db = new DBUtil();
        String sql;

        sql = "select * from psd;";
        List<Map<String, String>> list = db.getList(sql);
        Iterator<Map<String, String>> iter = list.iterator();

        if (list.isEmpty()){
            System.out.println("query nothing!");
        } else {
            while (iter.hasNext()) {
                Map<String, String> tempmap = (Map<String, String>) iter.next();
                PSD psd = new PSD();
                psd.setNode(tempmap.get("node"));
                psd.setStatus(tempmap.get("status"));
                psdList.add(psd);
            }
        }


        request.setAttribute("psdList",psdList);
        request.getRequestDispatcher("psdlist.jsp").forward(request, response);
    }

}
```

5.2　物联网移动端

5.2.1　Android 平台

Android 平台是我们生活中接触较多的平台，许多手机厂商是基于 Android 定制的系统，很多开发商开发基于 Android 平台的 App，在物联网中使用 Android 开发嵌入式控制程序。可以说，Android 是人们工作和生活中不可缺少的平台。因此，本书所研究的项目对于 Android 平台专门开发了一款 App。图 5-46 是一个简化的 Android 软件层次结构。

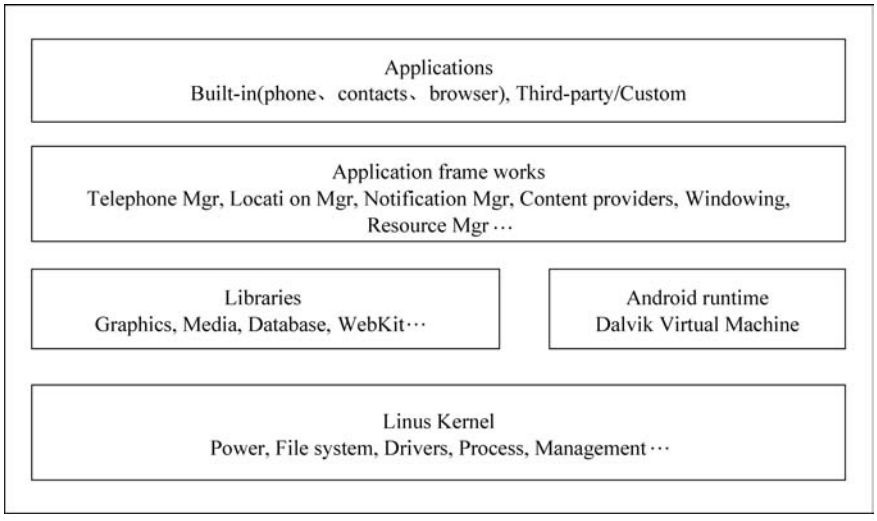

图 5-46　Android 软件层次结构

5.2.2　Android Studio 环境的安装

IDE 是 Intelligent Development Environment 的简称，即智能开发环境。Android IDE 是为 Android 应用开发提供支持的开发软件，有关 Android 的项目和代码将在 Android IDE 中管理。Android IDE 是一个集成开发环境，常用的 Android IDE 有 Eclipse＋ADT、ADT-Bundle 和 Android Studio。本书采用 Android Studio 作为移动端程序的集成开发环境。下面介绍 Android Studio 的安装配置过程。

（1）从 Android Studio 官方网址 http://www.android-studio.org/下载 Android Studio，界面如图 5-47 所示。

（2）下载好安装包之后，单击 Next 按钮进行安装，如图 5-48 所示。

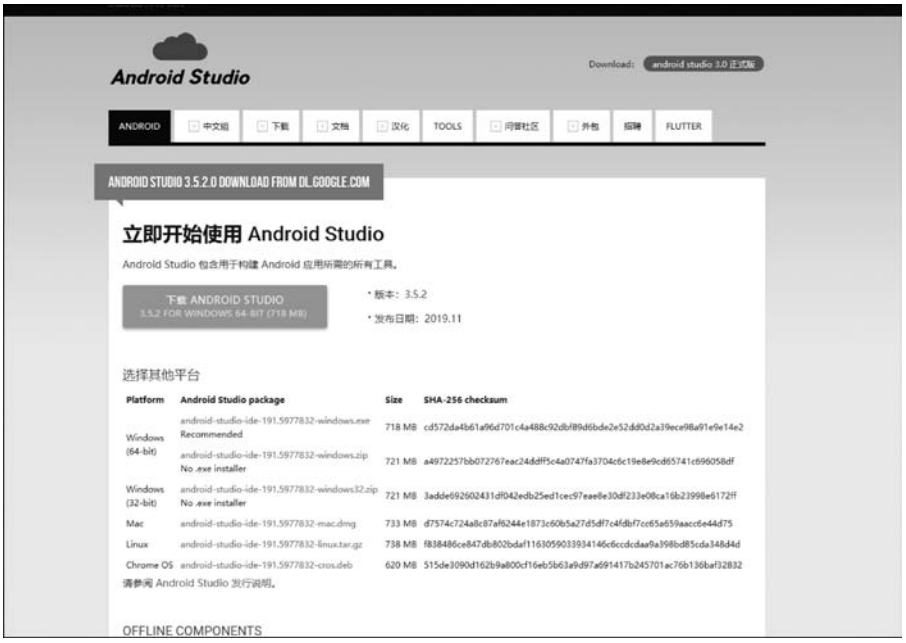

图 5-47　Android Studio 官方网站

图 5-48　Android 安装开始界面

（3）在选择组件对话框中，选择安装 Android Studio 虚拟机，如图 5-49 所示。

（4）在如图 5-50 所示的界面中，选择 Android Studio 的安装路径。

（5）在如图 5-51 所示的界面中，为 Android Studio 选择一个开始菜单文件夹，以便创建编程中使用到的快捷键。

（6）进入安装界面及安装完成界面如图 5-52～图 5-54 所示。

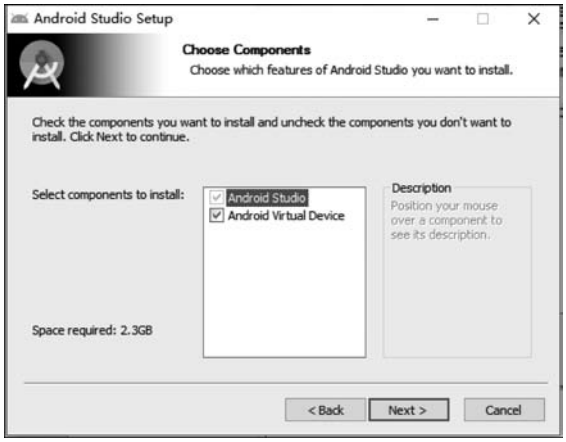

图 5-49　选择组件界面

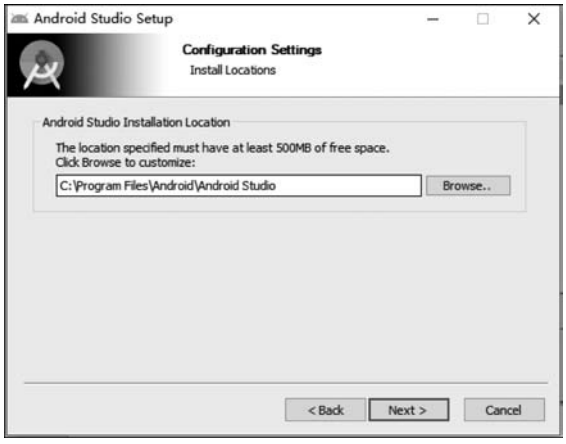

图 5-50　选择 Android Studio 安装路径

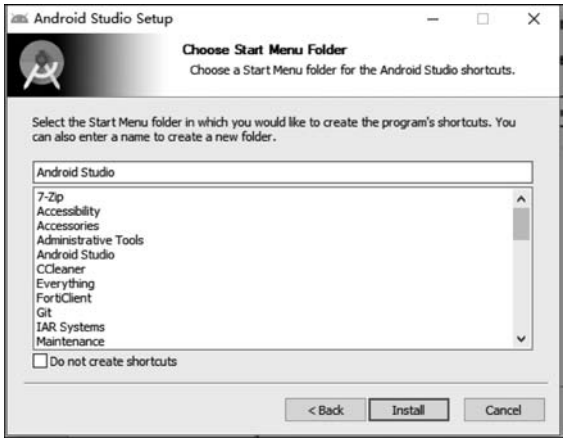

图 5-51　选择 Android Studio 开始菜单文件夹

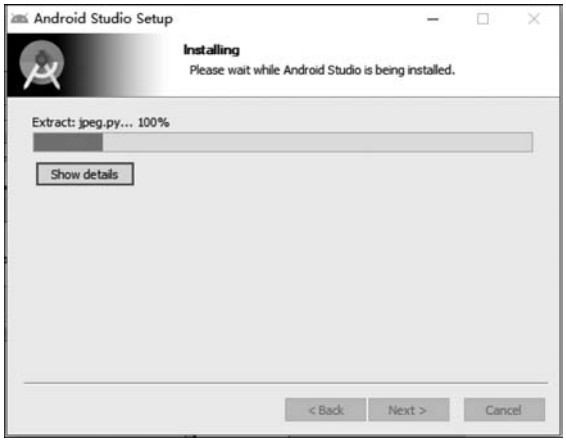

图 5-52 Android Studio 安装进度

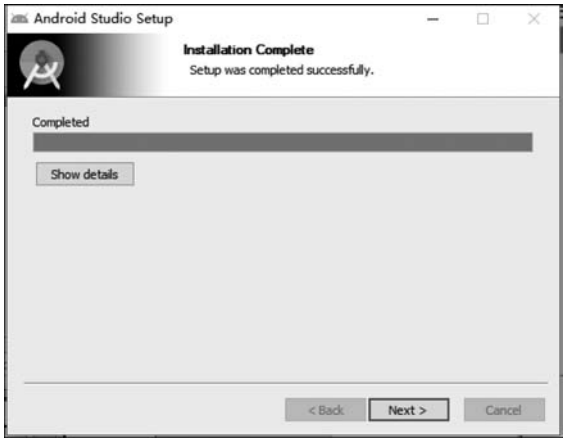

图 5-53 Android Studio 安装完成界面 1

图 5-54 Android Studio 安装完成界面 2

(7) 在导入 Android Studio 设置选项中,选择 Do not import settings,如图 5-55 所示。

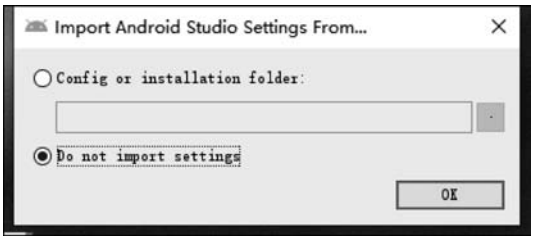

图 5-55　暂不导入 Android Studio 设置

(8) 在如图 5-56 所示的界面中,提示无法找到 Android SDK,此时单击 Cancel 按钮。

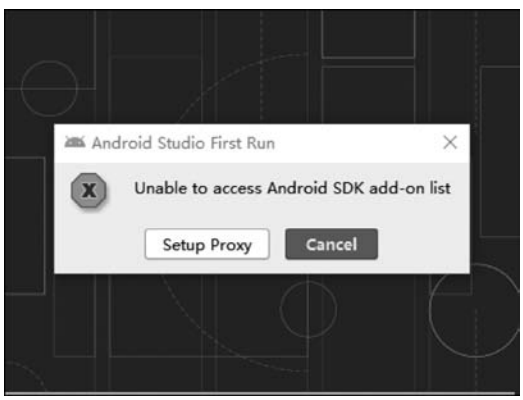

图 5-56　取消 Android SDK 获取

(9) 进入 Android Studio 的安装设置欢迎界面,如图 5-57 所示。

图 5-57　Android Studio 设置欢迎界面

（10）在安装类型中，选择标准安装，如图 5-58 所示。

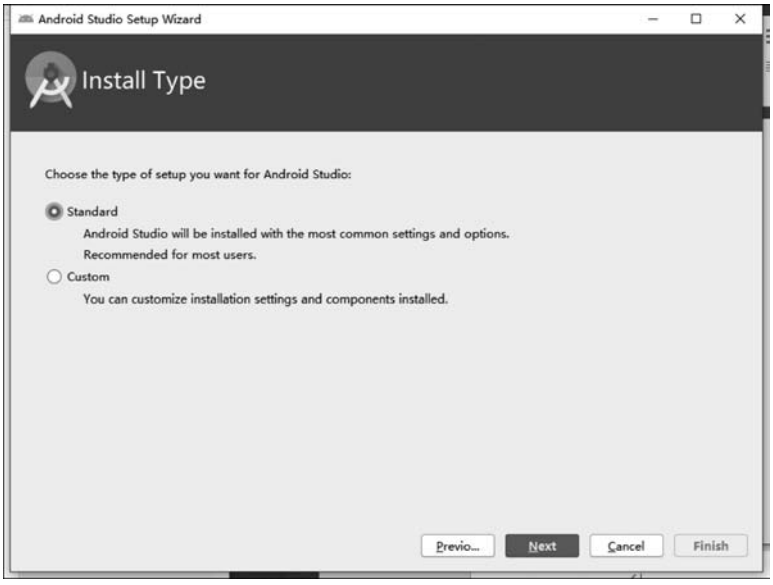

图 5-58　选择安装类型

（11）选择 Android Studio 的界面风格界面，可以选择自己熟悉的编程界面风格，如图 5-59 所示。

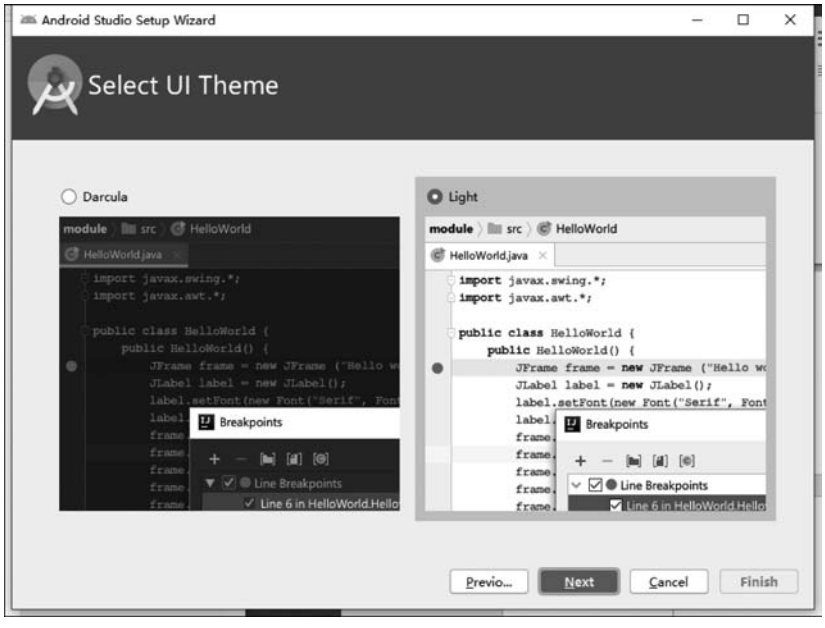

图 5-59　选择 Android Studio 界面风格

SDK 是 Software Development Kit 的简称，即软件开发工具包，一般是被软件工程师用于为特定的软件包、软件框架、硬件平台、操作系统等建立应用软件的开发工具的集合。

在 Android 中，Android SDK 为开发者提供了库文件以及其他开发所用到的工具。简单理解为 Android 开发工具包集合，是整体开发中所用到的工具包。这里需要指定 SDK 的本地路径，如果之前计算机中已经存在 SDK，可以指定该路径，后续就可以不用下载 SDK；这里暂时可以指定一个后续将保存 SDK 的路径，如图 5-60 所示。随后进入下载组件过程，如图 5-61 和图 5-62 所示。

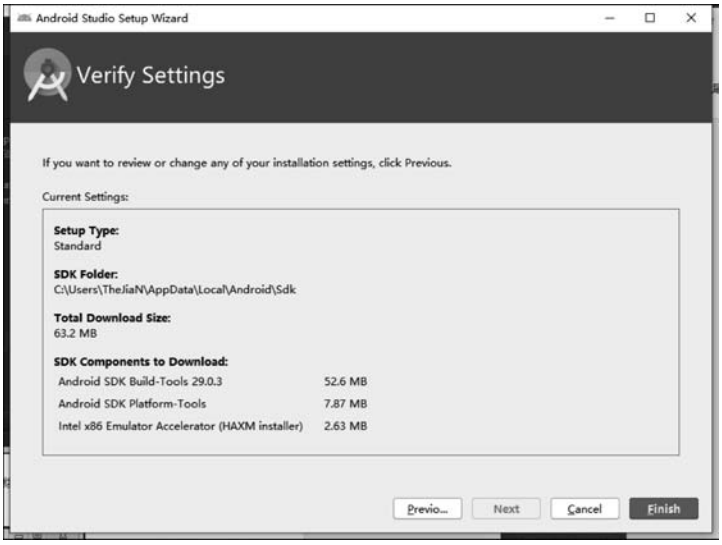

图 5-60　指定 SDK 路径

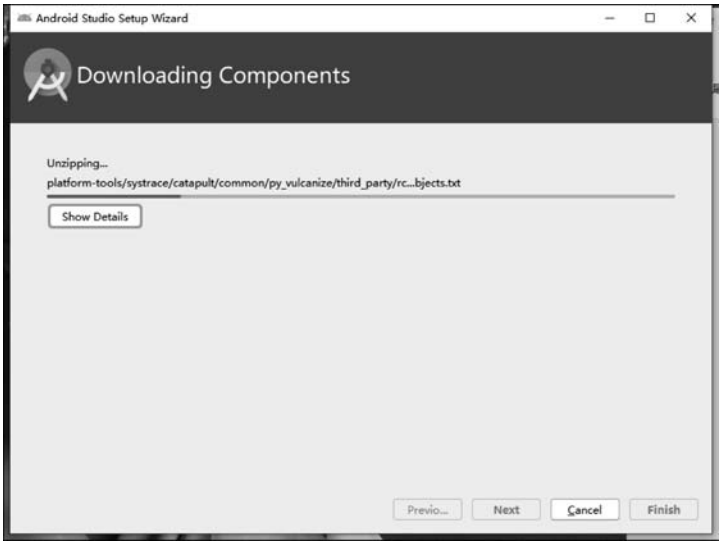

图 5-61　下载组件过程 1

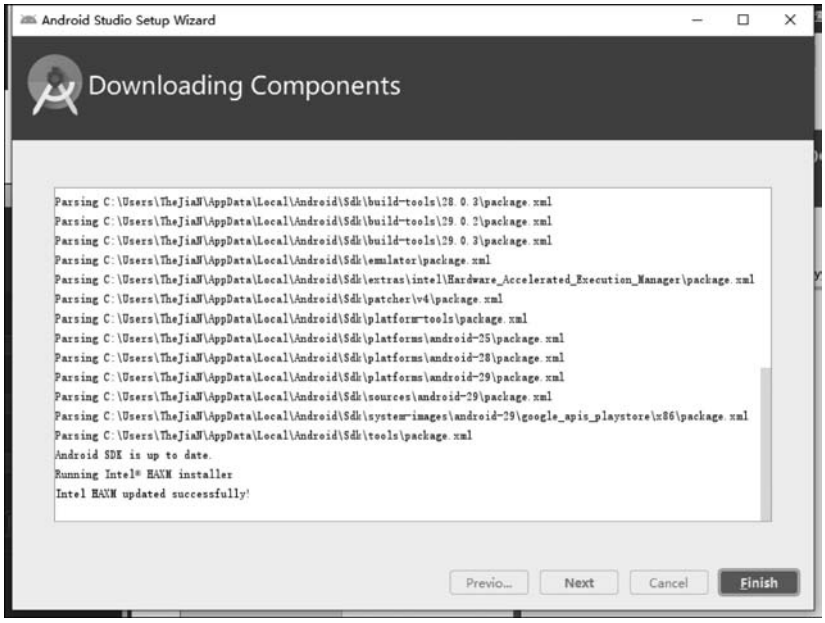

图 5-62　下载组件过程 2

组件下载完毕后，Android Studio 即安装成功，如图 5-63 所示。

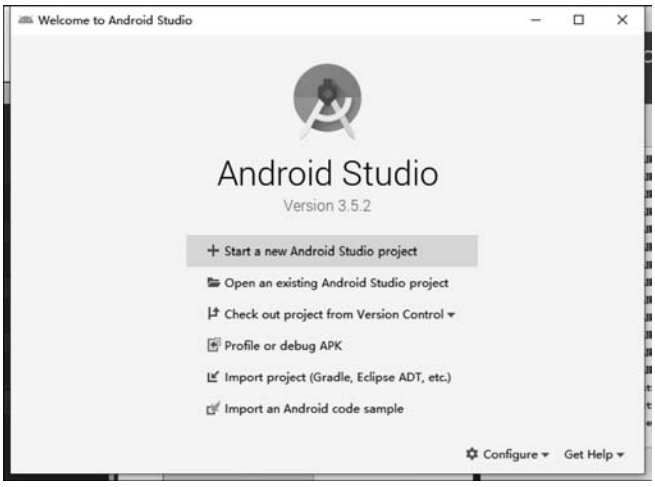

图 5-63　Android Studio 安装成功

单击图 5-63 中的 Start a new Android Studio project 新建一个工程，进入如图 5-64 所示界面，选择 Empty Activity 作为一个新的界面。

在如图 5-65 所示的界面中，新建一个新的工程，并加以配置。建立新工程后会出现工程编辑界面，如图 5-66 所示。

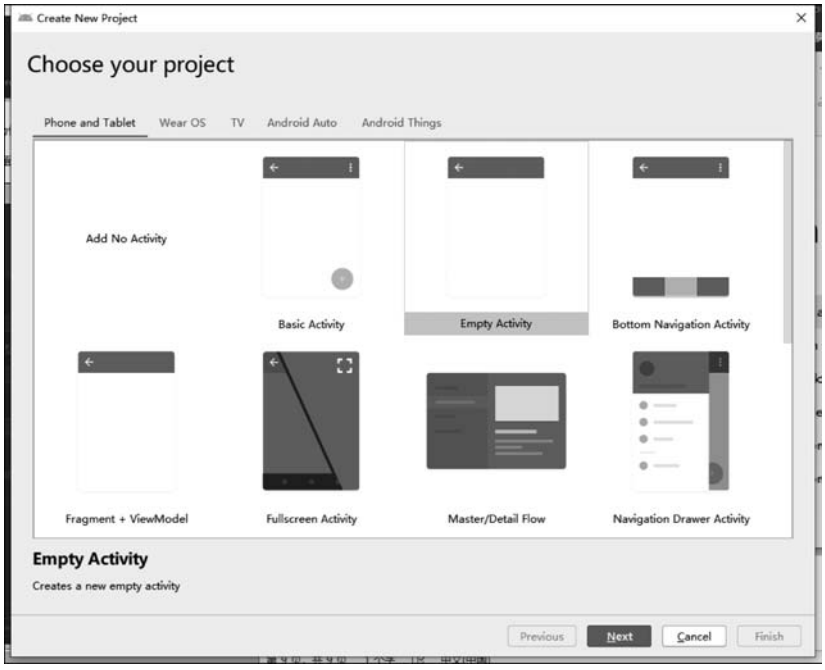

图 5-64　新建一个 Activity

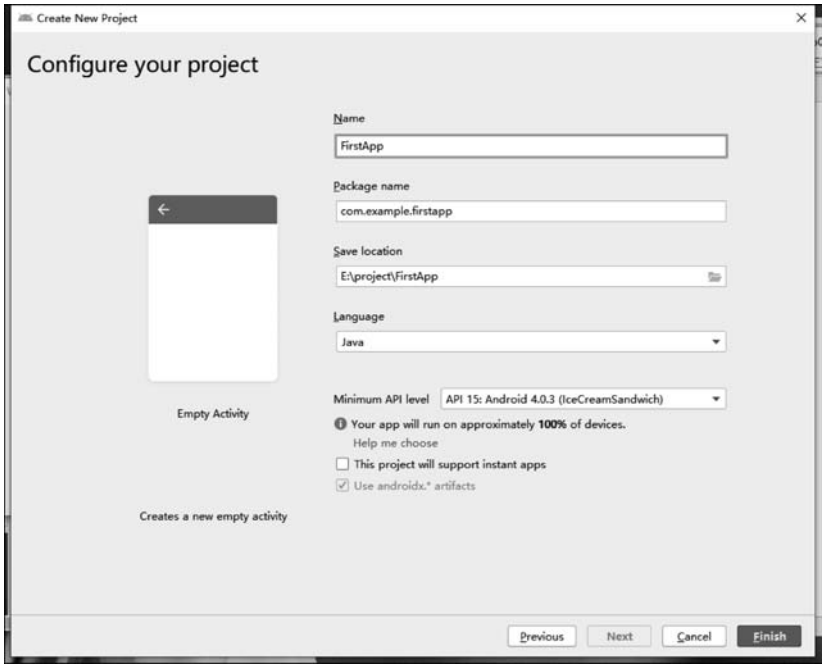

图 5-65　新建一个新的 Android 工程

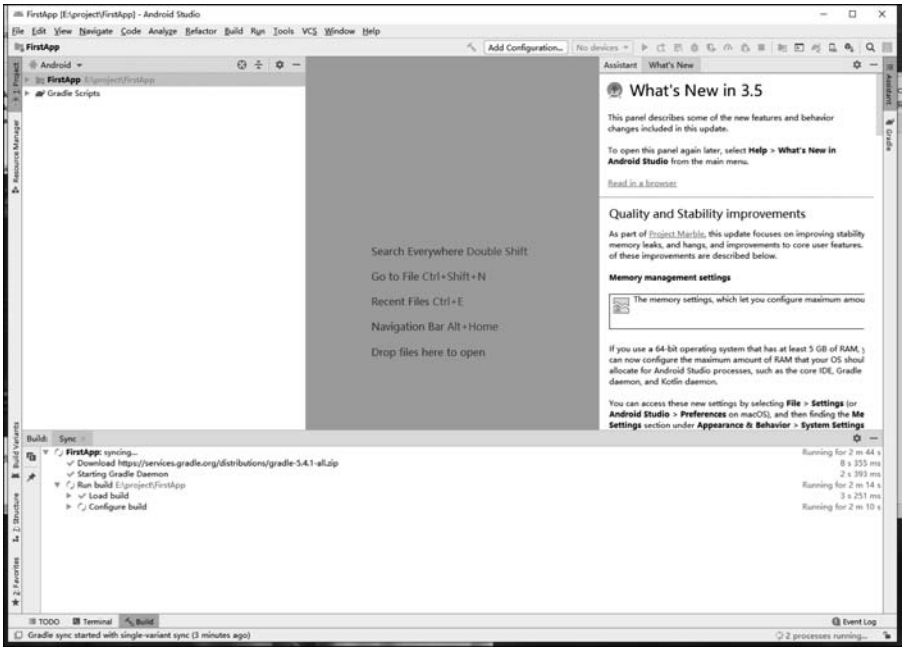

图 5-66　工程编辑界面

工程建立完成后，系统开始构建 gradle，如图 5-67 所示。

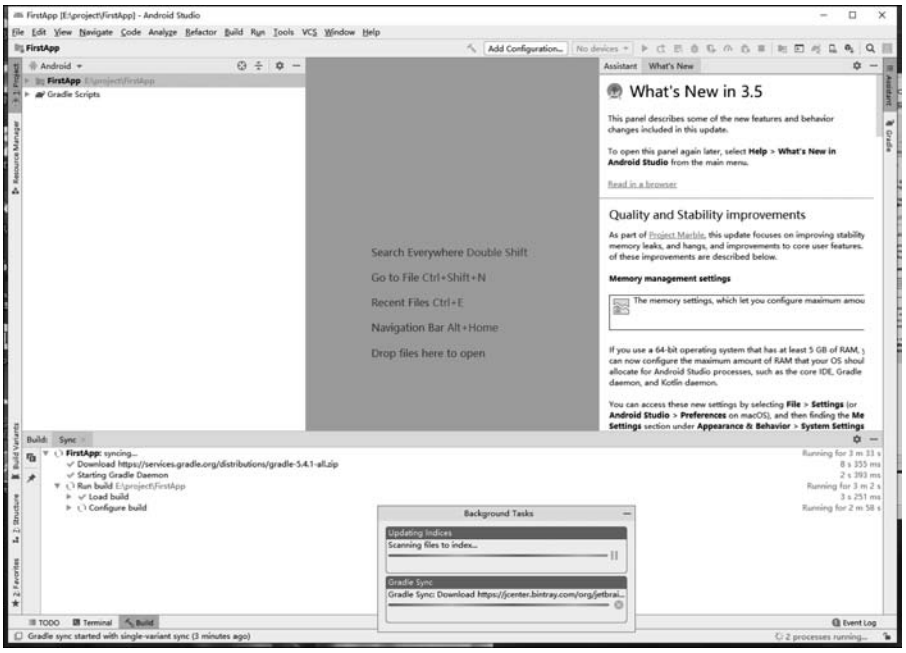

图 5-67　构建 gradle

gradle 构建成功之后，出现如图 5-68 所示的界面，代表一个 Android Studio 工程已经建立成功。即可在编码区域开始工程的编码实现，如图 5-69 所示。

图 5-68　Android Studio 工程建立成功

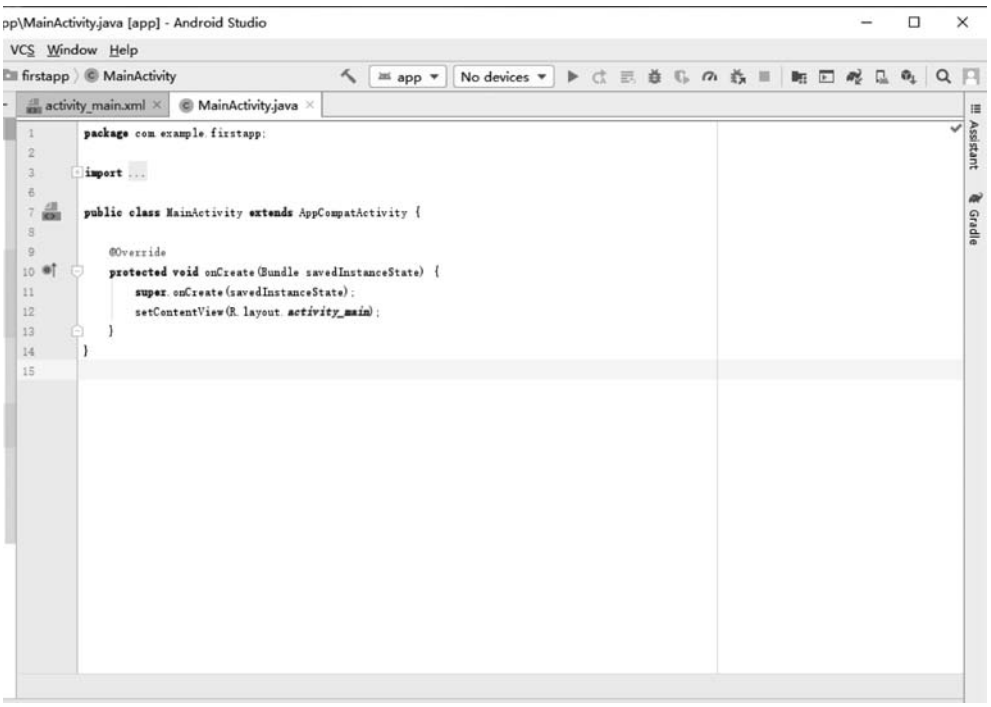

图 5-69　Android Studio 工程代码编辑区域

Android 工程编码完成后，可通过单击 Build→Build Bundle(s)/APK(S)选项，生成 Android APK 文件，用于在 Android 模拟器或 Android 系统的真机上运行测试，如图 5-70 所示。

5.2.3　Android 网络通信

OkHttp 框架是一个处理网络请求的开源项目，是安卓端的轻量级框架，由 Square 公司开发，用于替代 HttpUrlConnection 和 Apache HttpClient。OkHttp 框架可以支

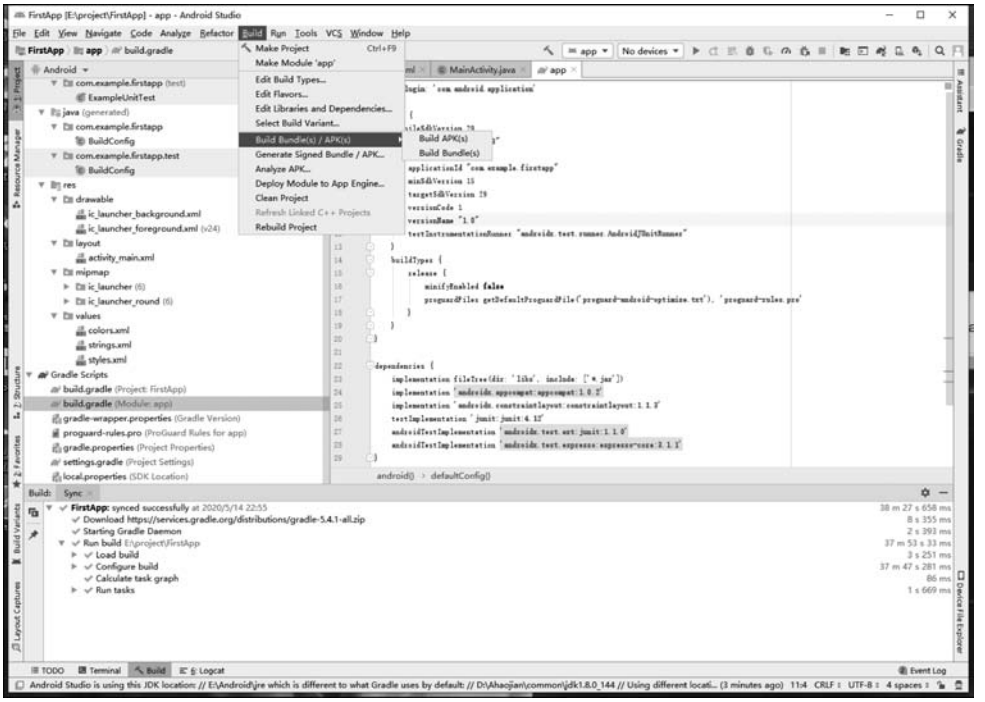

图 5-70 生成 Android APK

持 Android 2.3 及以上版本，需要 JDK 1.7 及以上版本。本书介绍利用 OkHttp 框架实现安卓端与服务器网络通信的方法。

OkHttp 框架优点如下。

（1）允许连接到同一个主机地址的所有请求，提高请求效率。

（2）共享 Socket，减少对服务器的请求次数。

（3）通过连接池，减少了请求延迟。

（4）通过缓存响应数据来减少重复的网络请求。

（5）减少了对数据流量的消耗。

（6）自动处理 GZip 压缩。

使用 OkHttp 框架之前，需要在工程中引入 OkHttp 包，OkHttp 框架还依赖另一个 okio 包，同样需要引入，代码如下所示；也可以在相应的 model 中的 build.gradle 配置文件中填入，然后将工程同步，环境会自动下载需要的包。

```
compile 'com.squareup.okhttp3:okhttp:3.2.0'
compile 'com.squareup.okio:okio:1.7.0'
//可以修改版本号
```

需要在 Android 工程中设置网络权限：

```
<user-permission andriod:name="andriod.premission.INTERNET"/> <!-- 用户连接网络权限 -->
```

OkHttp 框架的使用涉及 OkHttpClient、RequestBody、Request、Call、Response 等基本类。

(1) OkHttpClient：对于该类创建对象实例化的方式有默认的标准形式和自定义形式两种。

- 标准形式采用如下代码进行实例化。

```
OkHttpClient mOkHttpClient = new OkHttpClient();
```

- 以自定义形式实例化 OkHttpClient 对象时，可以设置网络连接的超时时长、读取超时时长和写入超时时长，可以调用 build()方法进行实例化，代码如下。

```
OkHttpClient mOkHttpClient = new OkHttpClient.Builder()
        .connectTimeout(10,TimeUtil.SECONDS)      //为新连接设置默认连接超时时长，第一个参数是时长，第二个参数是单位
        .readTimeout(10,TimeUtil.SECONDS)         //设置新连接的默认读取超时时长
        .writeTimeout(10,TimeUtil.SECONDS)        //设置新连接的默认写入超时时长
        .cache(setCache)                          //设置用于读取和写入缓存响应的响应缓存[1]
        .build();
```

在如上代码的[1]处，需要一个参数为 Cache 的对象，如下代码定义了 Cache 对象，大小是 10×1024×1024，在访问 File 对象指定的路径 filePath 时，可以扩展缓存区域大小。

```
//[1]:参数为 Cache 对象
File filePath = new File(getExternalCacheDir(),"netCache");
int cacheSize = 10 * 1024 * 1024;
Cache setCache = new Cache(filePath,cacheSize);
```

在 OkHttp2.x 版本中，设置以上超时时长的代码有所不同，如下所示。

```
mOkHttpClient.setConnectTimeout(10,TimeUtil.SECONDS);
mOkHttpCLient.setConnectTimeout(10,TimeUtil.SECONDS);
mOkHttpClient.setConnectTimeout(10,TimeUtil.SECONDS);
mOkHttpClient.setCache(setCache);
```

(2) RequestBody：该类是用于封装 OkHttp 框架进行网络通信时的请求体，适用于 post 请求方式，用于上传数据到服务器。其核心方法有如下 4 个。

```
public abstract MediaType contentType()
public long MediaType contentLength()
public abstract void writeTo(BufferedSink sink)
public static RequestBody create(MediaType contentType,String content) [2]
```

其中，MediaType 可以是多种类型，该类的对象实例化方式如下。

```
  //[2]:第二个参数可以是多种类型;该类存在 create 的多个重载方法
        //该类的对象实例化方式:
MediaType mMediaType = MediaType.parse("application/octet-stream");
//http://tool.oschina.net/commons
File putFile = new File(Environment.getExternaStorageDirectory(),"文件名.扩展名");
//父路径和子路径两个参数拼接起来为文件地址绝对地址
RequestBody mRequestBody = RequestBody.create(mMediaType,putFile);
```

RequestBody 的 create()方法需要两个参数：指定要上传文件的类型和文件对象本身。如果用 create()方法上传键值对类型的数据，可使用 FormBody，代码如下。

```
//使用 create()方法一般用于上传文件或字符串类型数据,对于上传键值对类型数据将会使
用 FormBody
RequestBody mRequestBody = FormBody.Builder()
                .add("key","value1")
                .add("key","value2")
                .build();
```

(3) Request：该类用于生成网络连接请求对象，包括请求参数、请求头、请求方式等多种信息，常用方法有 url()、post()、method()、headers()等。

```
Request mRequest = new Request.Builder()
            .url("http://www.baidu.com")
            .post(mRequestBody) [4]
            .build();
```

请求体可以通过 post()方法提交，如果不明确调用 post()方法，则可使用 get()方法请求网络。使用 post()方法上传数据比 get()方法更安全，数据大小不受限制，多个参数可以封装成请求体上传，对于长表单数据、上传文件，首选 post()方法请求网络。

(4) Call：该类是网络请求执行的最后一个需要实例的对象，该类提供了网络请求的同步与异步连接方式，网络连接属于耗时操作，因此不能编写在 UI 线程中。如果在网络请求后，需要更新 UI 布局就需要调用 Handler，实现 Handler 的 runOnUiThread()方法如下。

```
Call mCall = mOkHttpClient.newCall(mRequest);
//使用异步网络连接
mCall.enqueue(new CallBack(){
    @override
    public void onFailure(Call call,IOException e){
        //网络连接失败
    }
    @override
    public void onResponse(Call call,Response response){
        if(response.isSuccessful()){
            //请求数据成功,返回 code 为 200～300 [5]
        }else{
            //请求数据失败
        }
    }
});
```

其中,请求数据成功,服务器将返回响应状态码。响应状态码是在程序中经常需要判断的一个值,响应状态码大致分为:

- 1××:信息,表示请求收到,继续处理;
- 2××:成功,表示请求成功;
- 3××:重定向,为完成请求客户需进一步细化请求;
- 4××:由客户端引发的错误;
- 5××:由服务器引发的错误。

因此,2××(以 2 开头的状态码)表示请求成功。判断请求是否成功,及判断是否得到了请求成功的响应状态码,response.isSuccessful()的源码如下。

```
public boolean isSuccessful(){
    return code >= 200 && code < 300;
}
```

使用同步方式请求网络,在 Android 编程中被归类为耗时操作,耗时操作不允许在 UI 主线程中运行,因此需要开启新的子线程,代码如下。

```
new Thread(new Runnable(){
    Call mCall = mOkHttpClient.newCall(mRequest);
    Response mResponse = mCall.execute();        //涉及的 Response 对象知识点
    if(mResponse.isSuccessful()){
        //网络同步请求成功,更新 UI 布局需要使用 runOnUiThread()方法
    }else{

    }
```

```
}).start();                    //创建一个子线程后需要使用 start()方法启动该线程

注:runOnUiThread(new Runnable(){
        @override
        public void run(){
            //更新 UI 的逻辑代码
        }
    });

//同步方法会阻塞当前线程的执行,异步方法不会阻塞当前线程的执行
```

(5) Response：该类是网络请求后的响应信息对象,对于服务器返回的数据均存放在该示例对象中,而且对于 Response 实例一次请求中只能有一次有效调用,如果调用两次将会出现程序错误,这就使得在需要多次使用数据前要将 Response 实例中的数据保存下来,Response 类提供了多种方法,如 body、code、protocol、request、isSuccessful、headers(响应头对象)、toString 等。

```
Response mResponse = mCall.execute();
if(mResponse.isSuccessful()){
    String str = mResponse.body().string();   [6]
    int code = mResponse.code();
    Protocol protocol = mResponse.protocol();
    Request request = mResponse.request();
    String head = mResponse.toString();
    Log.i("Response 响应信息集:","" + str + code + protocol + request + head);
}
```

基于以上五个类,就可以写出基本的 get()、post()请求,进行同步或异步的网络连接了。对于 OkHttpClient 表示 HTTP 请求的客户端类,大多数情况下推荐只使用创建一个该对象的实例,全局使用。

下面是一个完整的关于 post()请求的异步方法示例。

```
public class PostAsyn(){
    private TextView tvShowNetInfo;
    public static OkHttpClient okHttpClient;
    static(
        okHttpClient = new OkHttpClient.Builder()
                .connectTimeout(15,TimeUnit.SECONDS)
                .readTimeout(15,TimeUnit.SECONDS)
                .writeTimeout(15,TimeUnit.SECOND)
                .build();
    );
    @Overvide
    public void onCreate(Bundle saveInstanceState){
```

```
        super.onCreate(saveInstanceState);
        setContentView(R.layout.activity_main);
        tvShowNetInfo = (TextView) findViewId(R.id.tv_show_net_info);
        postAsynDate();
    }
    public void postAsynData(){
        RequestBody requestBody = new FormBody.Builder()
                        .add("key1","value1")
                        .add("key2","value2")
                        .build();
        Request requset = new Request.Builder()
                    .url("http://www.baidu.com")
                    .post(requestBody)
                    .build();
        Call call = okHttpClient.newCall(request);
        call.enqueue(new Callback(){
                @Override
                public void onFailure(Call call,IOException e){
                    goUiThread("网络连接失败");
                }
                @Override
                public void onResponse(Call call,Response response){
                    if(response.isSuccessful()){
                        goUiThread("请求数据成功,响应码:" + response.code());
                    }else{
                         goUiThread("网络连接成功,但是没有响应数据,响应码:" +
response.code());
                    }
                }
            }

        );
    }
    private void goUiThread(final String str){
        runOnUiThread(new Runnable(){
            @Override
            public void run(){
                tvShowNetInfo.setText(str);
            }
        });
    }
}

//对应的布局文件略
```

5.3 习题

1. 下列(　　)选项不是 JSP 运行所必需的软件环境。

A. 操作系统　　B. JavaJDK

C. 支持 JSP 的 Web 服务器　　D. 数据库

2. 在 JDBC 中,用来描述结果集的接口是(　　)。

A. Statement　　B. Connection

C. ResultSet　　D. DriverManager

3. 在 JDBC API 中,下列(　　)接口或类可以用来保存从数据库返回的查询结果。

A. ResultSet　　B. Connection

C. Statement　　D. DriverManager

4. 在 JDBC API 中,下列(　　)接口或类可以用来执行 SQL 语句。

A. ResultSet　　B. Connection

C. Statement　　D. DriverManager

5. 下述选项中不属于 JDBC 基本功能的是(　　)。

A. 与数据库建立连接　　B. 提交 SQL 语句

C. 处理查询结果　　D. 数据库维护管理

6. 假设已创建了语句对象名为 sta,下列语句中错误的是(　　)。

A. sta.executeUpdate("delete from food where price < 0");

B. sta.executeQuery("delete from food where price < 0");

C. sta.execute("delete from food where price < 0");

D. sta.executeQuery("select * from food where price <> 0");

7. 创建 JDBC 的数据库连接对象,下列语句中正确的是(　　)。

A. Connection conn = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/mealsystem", "root", "root");

B. Connection conn = Class.forName("jdbc:mysql://127.0.0.1:3306/mealsystem", "root", "root");

C. Connection conn = Driver.getConnection("jdbc:mysql://127.0.0.1:3306/mealsystem", "root", "root");

D. Connection conn = DriverManager.getConnection("com.mysql.jdbc.Driver", "root", "root");

8. 下列关于 JDBC 说法中错误的是(　　)。

A. JDBC 使得编程人员从复杂的驱动器调用命令和函数中解脱出来,可以致力于应用程序中的关键地方

B. JDBC 支持非关系数据库，如 NoSQL 等

C. 用户可以使用 JDBC-ODBC 桥驱动器将 JDBC 函数调用转换为 ODBC

D. JDBC API 是面向对象的，可以让用户把常用的方法封装为一个类以备后用

9. 请阐述 MVC 设计模式的含义，并说明 M、V、C 各自代表什么层次。

10. 下列关于 MVC 的说法中错误的是(　　)。

A. 在 MVC 模式中，如果哪一层的需求发生了变化，只需要更改相应层的代码而不会影响其他层中的代码

B. 在 MVC 模式中，所有的核心业务逻辑都应该放在控制层实现

C. 在 MVC 模式中，由于按层把系统分开，那么就能更好地实现开发中的分工

D. 使用 MVC 模式，有利于组件的重用

11. Android 应用程序需要打包成(　　)文件格式在手机上安装运行。

A. class　　B. .xml　　C. .apk　　D. dex

12. 在 Activity 的生命周期中，当 Activity 被某个 AlertDialog 覆盖一部分后，会处于(　　)状态。

A. 暂停　　B. 活动　　C. 停止　　D. 销毁

13. Android 项目启动时最先加载的是 AndroidManifest.xml 文件，如果有多个 Activity，以下(　　)属性决定了该 Activity 最先被加载。

A. android.intent.action.LAUNCH

B. android:intent.action.ACTIVITY

C. android:intent.action.MAIN

D. android:intent.action.VIEW

14. 下列关于 Handler 的说法中不正确的是(　　)。

A. 它是实现不同进程间通信的一种机制

B. 它采用队列的方式来存储 Message

C. Handler 既是消息的发送者也是消息的处理者

D. 它是实现不同线程间通信的一种机制

15. 下列(　　)不是 Android 的存储方式。

A. File　　B. SharedPreferences

C. SQLite　　D. ContentProvider

16. Android 支持的 4 大重要组件，分别是 Activity、_____、Service 和 Content Provider。

17. Android 的事件处理机制有两种：一种是基于回调的处理机制；另一种是_____。