**组长：王子昂**

**组员：王迪**

**黎佩瑜**

**陈灯月**

**童路勤**

****

天气预报

系统实现文档

系统实现文档

1 服务器模块实现 3

1.1 与前端axios交互 4

1.2 数据库连接与操作 8

1.3 shiro安全管理 8

2 预测模型实现 11

2.1 数据清洗 11

2.2 模型建立 12

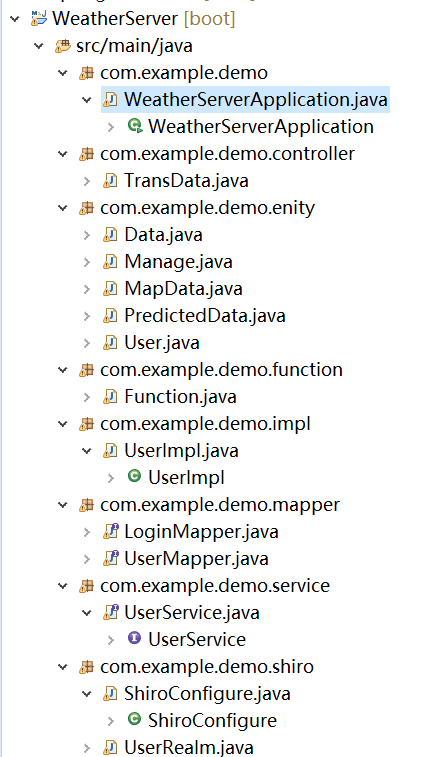
3 前端界面实现 14

3.1 界面与服务器交互 14

4 数据库实现 15

1 服务器模块实现

主要技术与框架：java、Spring boot、shiro

目录结构：

### 1.1 与前端axios交互

|  |  |
| --- | --- |
| 设计 | 从前端收到用户名username和密码password，通过shiro认证密码进行登陆。 |
| 实现 | @RequestMapping(value = "/login", method = RequestMethod.***POST***)  @ResponseBody  **public** **int** login(@RequestBody User user) {  String s1=user.getUsername();  String s2=user.getPassword();  // 从SecurityUtils里边创建一个 subject  Subject subject = SecurityUtils.*getSubject*();  // 在认证提交前准备 token（令牌）  UsernamePasswordToken token = **new** UsernamePasswordToken(s1, s2);  System.***out***.println("mapper "+mapper);  String role = mapper.getUserRoleByName(s1);  System.***out***.println("subject "+subject);  System.***out***.println("role "+role);  **try** {  System.***out***.println("token "+token);  subject.login(token);  }**catch**(UnknownAccountException e) {//用户名不存在  **return** 0;  }**catch**(IncorrectCredentialsException e) {//密码错误  **return** 2;  }  **if**(role.equals("user")) {  **return** 1;  }**else** **if**(role.equals("admin")){  **return** 3;  }  **else** {  **return** 4;  }  } |
| 设计 | 从前端收到用户名和密码，连接数据库实现注册 |
| 实现 | @RequestMapping(value = "/register", method = RequestMethod.***POST***)  @ResponseBody  **public** **int** register(@RequestBody User user) {  String s1=user.getUsername();  String s2=user.getPassword();  **if**(userMapper.existName(s1)!=**null**) {  **return** 0;  }**else** {  userMapper.insertUser(s1, s2,"user");  **return** 1;  }  } |
| 设计 | 从前端收到日期与指令，指令chart返回前端该日期往后七天（含该日期）北京的气温预测数据，其他指令返回全国各城市该日期的气温预测数据。 |
| 实现 | @RequestMapping(value = "/pickdate", method = RequestMethod.***POST***)  @ResponseBody  **public** String getJson(@RequestBody Data data) {  String[] pos= {"安徽","北京","福建","甘肃","广东","广西","贵州","黑龙江","海南","河北","河南",  "河北","湖北","湖南","吉林","江苏","江西","辽宁","内蒙古","宁夏","青海","山东","山西",  "陕西","上海","四川","天津","西藏","新疆","云南","浙江","重庆"};  **int** k=0;  String sql,r,d;  Connection conn = *getConn*();  ArrayList<PredictedData> preDataJsonList =**new** ArrayList<PredictedData>();  String[] temp;  String s1=data.getDate1();  String s2=data.getDate2();  String date=data.getDate();  String order=data.getOrder();  **if**(order.equals("chart")) {  sql = "select \* from beijing2";  **try** {  ResultSet rs = ((PreparedStatement) conn.prepareStatement(sql)).executeQuery();  **while**(rs.next()) {  d=rs.getString("date");  **if**(d.equals(s1)) {  r=rs.getString("value");  temp=r.split("#");  PredictedData preData=**new** PredictedData();  preData.setDate(rs.getString("date"));  preData.setAverage(Double.*parseDouble*(temp[0]));  preData.setMax(Double.*parseDouble*(temp[1]));  preData.setMin(Double.*parseDouble*(temp[2]));  preDataJsonList.add(preData);  **while**(rs.next()&&k<6) {  r=rs.getString("value");  temp=r.split("#");  PredictedData preData1=**new** PredictedData();  preData1.setDate(rs.getString("date"));  preData1.setAverage(Double.*parseDouble*(temp[0]));  preData1.setMax(Double.*parseDouble*(temp[1]));  preData1.setMin(Double.*parseDouble*(temp[2]));  preDataJsonList.add(preData1);  k++;  }    }  }  } **catch** (SQLException e) {  e.printStackTrace();  }  String re=*getOptionsJson*(preDataJsonList);  System.***out***.println(re);  **return** re;  }  **else** {  String re="[";  **for**(**int** i=0;i<31;i++) {  **try** {  MapData mapdata=**new** MapData();  sql = "select \* from "+pos[i];  ResultSet rs = ((PreparedStatement) conn.prepareStatement(sql)).executeQuery();  **while**(rs.next()) {  d=rs.getString("date");  **if**(d.equals(date)) {  re+="{";  r=rs.getString("value");  temp=r.split("#");  re+="\"pos\":";  re+="\""+pos[i]+"\",";  re+="\"min\":"+temp[2]+",";  re+="\"avg\":"+temp[0]+",";  re+="\"max\":"+temp[1]+"},";  **break**;  }  }  } **catch** (SQLException e) {  e.printStackTrace();  }    }  re=re.substring(0,re.length() - 1);  re+="]";  System.***out***.println(re);  **return** re;  }    } |
| 设计 | 从前端接收一个java Bean实体类，根据all/delete/modify/add相关指令，对数据库进行对应的增删查改，并返回前端相应的信息（操作成功/失败原因）。 |
| 实现 | @RequestMapping(value = "/admin", method = RequestMethod.***POST***)  @ResponseBody  **public** String admin(@RequestBody Manage manage) {  String order=manage.getOrder();  String name=manage.getUsername();  String password=manage.getPassword();  String newName=manage.getUsernameafter();  String nameBefore=manage.getUsernamebefore();  **if**(order.equals("all")) {  ArrayList<User> userlist=(ArrayList<User>) userMapper.getUserList();  String re=*getEvaluatedOptionsJson*(userlist);  System.***out***.println(re);  **return** re;  }  **if**(order.equals("delete")) {  userMapper.deleteUser(name);  **return** "1";  }  **if**(order.equals("modify")) {  **if**(newName.equals(nameBefore)) {//只改密码  userMapper.updatePassword(nameBefore, password);  **return** "1";  }  **else** {  String exist=userMapper.existName(newName);  **if**(exist==**null**) {  String role=userMapper.getRole(nameBefore);  userMapper.deleteUser(nameBefore);  userMapper.insertUser(newName, password,role);  **return** "1";  }  **else** {  **return** "2";  }  }  }  **if**(order.equals("add")){ //add  String exist=userMapper.existName(name);  **if**(exist==**null**) {  userMapper.insertUser(name, password, "user");  **return** "1";  }  **else** {  **return** "2";  }  }  **return** "3";  } |

### 1.2 数据库连接与操作

|  |  |
| --- | --- |
| 设计 | 连接数据库 |
| 实现 | **private** **static** Connection getConn() {  String driver = "com.mysql.jdbc.Driver";  String url = "jdbc:mysql://localhost:3306/predict?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";  String username = "root";  String password = "Mysql78089091";  Connection conn = **null**;  **try** {  Class.*forName*(driver); // classLoader,加载对应驱动  conn = (Connection) DriverManager.*getConnection*(url, username, password);  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** conn;  } |

### 1.3 shiro安全管理

**Shiro配置文件**

|  |  |
| --- | --- |
| 设计 | 设置安全管理器，添加Shiro的内置过滤器 |
| 实现 | @Bean  **public** ShiroFilterFactoryBean getShiroFilterFactoryBean(@Qualifier("getDefaultWebSecurityManager") DefaultWebSecurityManager defaultWebSecurityManager) {  ShiroFilterFactoryBean bean=**new** ShiroFilterFactoryBean();  //设置安全管理器  bean.setSecurityManager(defaultWebSecurityManager);  //添加Shiro的内置过滤器  Map<String,String>filterMap=**new** LinkedHashMap<>();  System.***out***.println("拦截了吗？");  filterMap.put("/login", "anon");  filterMap.put("/register", "anon");  filterMap.put("/pickdate", "roles[user]");  filterMap.put("/admin", "roles[admin]");  bean.setUnauthorizedUrl("/");  bean.setLoginUrl("/");  bean.setFilterChainDefinitionMap(filterMap);  **return** bean;  } |

**自定义Realm，实现权限和身份认证**

|  |  |
| --- | --- |
| 设计 | 获取用户身份做权限认证 |
| 实现 | **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  System.***out***.println("————权限认证————");  Subject subject=SecurityUtils.*getSubject*();  User currentUser=(User)subject.getPrincipal();  //String username = (String) SecurityUtils.getSubject().getPrincipal();  SimpleAuthorizationInfo info = **new** SimpleAuthorizationInfo();  //获得该用户角色  //String role = userService.getUserRoleByName(username);  String role=currentUser.getRole();  System.***out***.println("username "+currentUser.getUsername());  System.***out***.println("role "+role);  Set<String> set = **new** HashSet<>();  //需要将 role 封装到 Set 作为 info.setRoles() 的参数  set.add(role);  //设置该用户拥有的角色  info.setRoles(set);  **return** info;  } |
| 设计 | 获取输入的用户名和密码，交给shiro判断密码正确与否，做身份认证。 |
| 实现 | **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) **throws** AuthenticationException {  System.***out***.println("————身份认证方法————");  UsernamePasswordToken token = (UsernamePasswordToken) authenticationToken;  // 从数据库获取对应用户名密码的用户  String test = token.getUsername();  System.***out***.println(test);  String password = userService.getUserPasswordByName(token.getUsername());  User user=userService.getUser(test);  System.***out***.println(password);  **if** (**null** == password) {//用户名不存在  **return** **null**;  }  System.***out***.println("身份认证完毕");  //密码认证交给shiro  //return new SimpleAuthenticationInfo("", password, "");  **return** **new** SimpleAuthenticationInfo(user, password, getName());  } |

2 预测模型实现

### 2.1 数据清洗

采用spark本地单节点进行数据清洗

|  |  |
| --- | --- |
| 设计 | spark-merger.py负责将一个城市对应的1980-2020一共41个数据文件合并成一个 |
| 实现 | **from** pyspark.sql **import** SparkSession  **import** pandas **as** pd  cities = []  log\_file = open('E:/log\_file.log', 'w')  **with** open('E:/map2.txt', 'r') **as** maplist:  lines = maplist.readlines()  **for** line **in** lines:  cities.append((line[:11].strip(), line[12:].strip()))  spark = SparkSession.builder.appName("SimpleApp").getOrCreate()  **for** city\_num, city\_name **in** cities:  df = spark.read.csv('E:/data\_p3/{}\_{}\_{}.csv'.format(city\_name, city\_num, 1980), header=True)  **for** i **in** range(1981, 2021):  **print**('merge precess: {} {}'.format(i, city\_name))  df1 = spark.read.csv('E:/data\_p3/{}\_{}\_{}.csv'.format(city\_name, city\_num, i), header=True)  df = df.unionAll(df1)  df\_pd = df.toPandas()  **try**:  df\_pd['date'] = pd.to\_datetime(df\_pd['date'], format='%Y%m%d')  **except**:  log\_file.write('Error happens in {}\n'.format(city\_name))  df\_pd.to\_csv('E:/data\_p4/{}\_{}.csv'.format(city\_name, city\_num), index=**None**)  spark.stop() |
| 设计 | spark-selector.py 负责从一年多城市的数据中提取出指定城市 |
| 实现 | **rom** pyspark.sql **import** SparkSession  **import** pandas **as** pd  year = 1983  cities = []  **with** open('E:/map2.txt', 'r') **as** maplist:  lines = maplist.readlines()  **for** line **in** lines:  cities.append((line[:11].strip(), line[12:].strip()))  log\_file = open('E:/log\_file\_{}.log'.format(year), 'w')  csvFile = "E:/weather\_data/{}.csv".format(year)  spark = SparkSession.builder.appName("SimpleApp").getOrCreate()  df = spark.read.csv(csvFile).cache()  df = df.drop('\_c4', '\_c5', '\_c6', '\_c7')  num = 0  **for** city\_num, city\_name **in** cities:  **try**:  **print**('extract: {} {}'.format(num, city\_name))  df1 = df.filter("\_c0 == '"+city\_num+"'")  df2 = df1.select("\_c1", "\_c2", "\_c3")  df2 = df2.toDF("date", "key", "value")  df2 = df2.withColumn('value', df2.value / 10)  tavg = df2.filter("key == 'TAVG'").drop('key').withColumnRenamed('value', 'tavg').toPandas()  tmax = df2.filter("key == 'TMAX'").drop('key').withColumnRenamed('value', 'tmax').toPandas()  tmin = df2.filter("key == 'TMIN'").drop('key').withColumnRenamed('value', 'tmin').toPandas()  tbin1 = pd.merge(tavg, tmax, how='outer', on='date')  tbin1 = pd.merge(tbin1, tmin, how='outer', on='date')  tbin1.to\_csv('E:/data\_p3/{}\_{}\_{}.csv'.format(city\_name, city\_num, year), index=**None**)  **except**:  log\_file.write('{}-{} has error.\n'.format(city\_num, city\_name))  num = num + 1  log\_file.close()    spark.stop() |

### 2.2 模型建立

先后建立了ARIMA、LSTM两个模型，最后采用了LSTM模型进行预测

|  |  |
| --- | --- |
| 设计 | lstm\_trainer.py 进行LSTM模型训练 |
| 实现 | *#* 训练模型，训练结束后保存模型并输出在训练集和验证集上的预测效果  **def** train(self, epochs=40, batch\_size=64, log\_file=**None**):  dataset = self.load\_data(self.data\_file.format(self.city\_name), [self.label])  *#* 数据处理  scaler = MinMaxScaler(feature\_range=self.scaler)  trainX, trainY, testX, testY = self.process\_data(dataset, scaler)  **print**(trainX)  **print**(trainY)  model = self.get\_model(self.layers, compile=True)  model.summary()  reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.7, patience=3, verbose=1)  history = model.fit(trainX, trainY, epochs=epochs, batch\_size=batch\_size, verbose=1,  validation\_data=(testX, testY))  model.save\_weights(self.model\_path.format(self.city\_name, self.label))  *#* 预测  train\_predict = model.predict(trainX)  test\_predict = model.predict(testX)  *#* 逆正则化  reprocessed = reprocess\_data(scaler, [test\_predict, testY, train\_predict, trainY])  repro\_test\_predict, repro\_testY, repro\_train\_predict, repro\_trainY = reprocessed  *#* 计算 *MSE* 等  evaluate('test', repro\_testY, repro\_test\_predict, log\_file=log\_file)  evaluate('train', repro\_trainY, repro\_train\_predict, log\_file=log\_file) |
| 设计 | lstm\_predictor.py 用LSTM模型进行数据预测 |
| 实现 | *#* 输入长度为 *self.time\_step* 的序列，预测序列后长度为 *self.predict\_num* 的序列  **def** predict(self, x\_pred):  model = self.get\_model(self.layers)  model.load\_weights(self.model\_path.format(self.city\_name, self.label))  predict\_x = np.array(x\_pred).astype(float)  *# reshape*，用于归一化  predict\_x = np.reshape(predict\_x, (self.time\_step, self.feature\_num))  scaler\_predict = MinMaxScaler(feature\_range=self.scaler)  predict\_x = scaler\_predict.fit\_transform(predict\_x)  *# reshape*，用于输入 *LSTM*  predict\_x = np.reshape(predict\_x, (1, self.time\_step, self.feature\_num))  predict\_y = model.predict(predict\_x)  *#* 逆归一化得到结果  predict\_y = scaler\_predict.inverse\_transform(predict\_y) |

3 前端界面实现

主要技术：vue框架、Element-UI、axios、Echart

### 3.1 界面与服务器交互

|  |  |
| --- | --- |
| 设计 | 登陆页面发送请求并接受服务器信息 |
| 实现 | Macintosh HD:Users:li:Library:Containers:com.tencent.xinWeChat:Data:Library:Application Support:com.tencent.xinWeChat:2.0b4.0.9:c74edfdd30c36b195669b6a9e6eb3168:Message:MessageTemp:28474d761ca330245ed26a83435bc3ac:Image:161594630912_.pic_hd.jpg |
| 设计 | 注册页面发送请求并接受服务器信息 |
| 实现 | Macintosh HD:Users:li:Library:Containers:com.tencent.xinWeChat:Data:Library:Application Support:com.tencent.xinWeChat:2.0b4.0.9:c74edfdd30c36b195669b6a9e6eb3168:Message:MessageTemp:28474d761ca330245ed26a83435bc3ac:Image:171594630913_.pic_hd.jpg |
| 设计 | 气温查询页面发送请求并接受服务器信息 |
| 实现 | Macintosh HD:Users:li:Library:Containers:com.tencent.xinWeChat:Data:Library:Application Support:com.tencent.xinWeChat:2.0b4.0.9:c74edfdd30c36b195669b6a9e6eb3168:Message:MessageTemp:28474d761ca330245ed26a83435bc3ac:Image:151594630911_.pic_hd.jpg |

4 数据库实现

主要技术：mysql、华为云部署

|  |  |
| --- | --- |
| 设计 | 数据库操作 |
| LoginMapper.java | 从数据库中获取名字，密码和User对象。实现增加新的用户到数据库 |
| 增加用户 | @Insert("insert into member(username,password) values(#{user\_name},#{password})")  **void** addUser(String user\_name, String password); |
| 通过用户名获取角色 | @Select("select distinct roles from member where username = #{username}")  String getUserRoleByName(String username); |
| 通过用户名获取密码 | @Select("select distinct password from member where username = #{username}")  String getUserPasswordByName(String username); |
| 通过用户名获取User | @Select("select \* from member where username=#{username}")  User getUser(String username); |
| UserMapper.java | 获取全部用户List, 获取指定用户名，获取角色，更新密码，增加新的用户，删除用户 |
| 获取User列表 | @Select("Select \* from member")  List<User> getUserList(); |
| 返回指定用户名 | @Select("Select \* from member where username=#{username}")  String existName(String username); |
| 通过用户名获取角色 | @Select("Select roles from member where username=#{username}")  String getRole(String username); |
| 更新密码 | @Update("update member set password=#{password} where username=#{username}")  **void** updatePassword(String username,String password); |
| 插入新用户 | @Insert("insert into member(username,password,roles) values(#{username},#{password},#{roles})")  **void** insertUser(String username,String password,String roles); |
| 删除指定用户 | @Delete("delete from member where username = #{usename}")  **void** deleteUser(String username); |