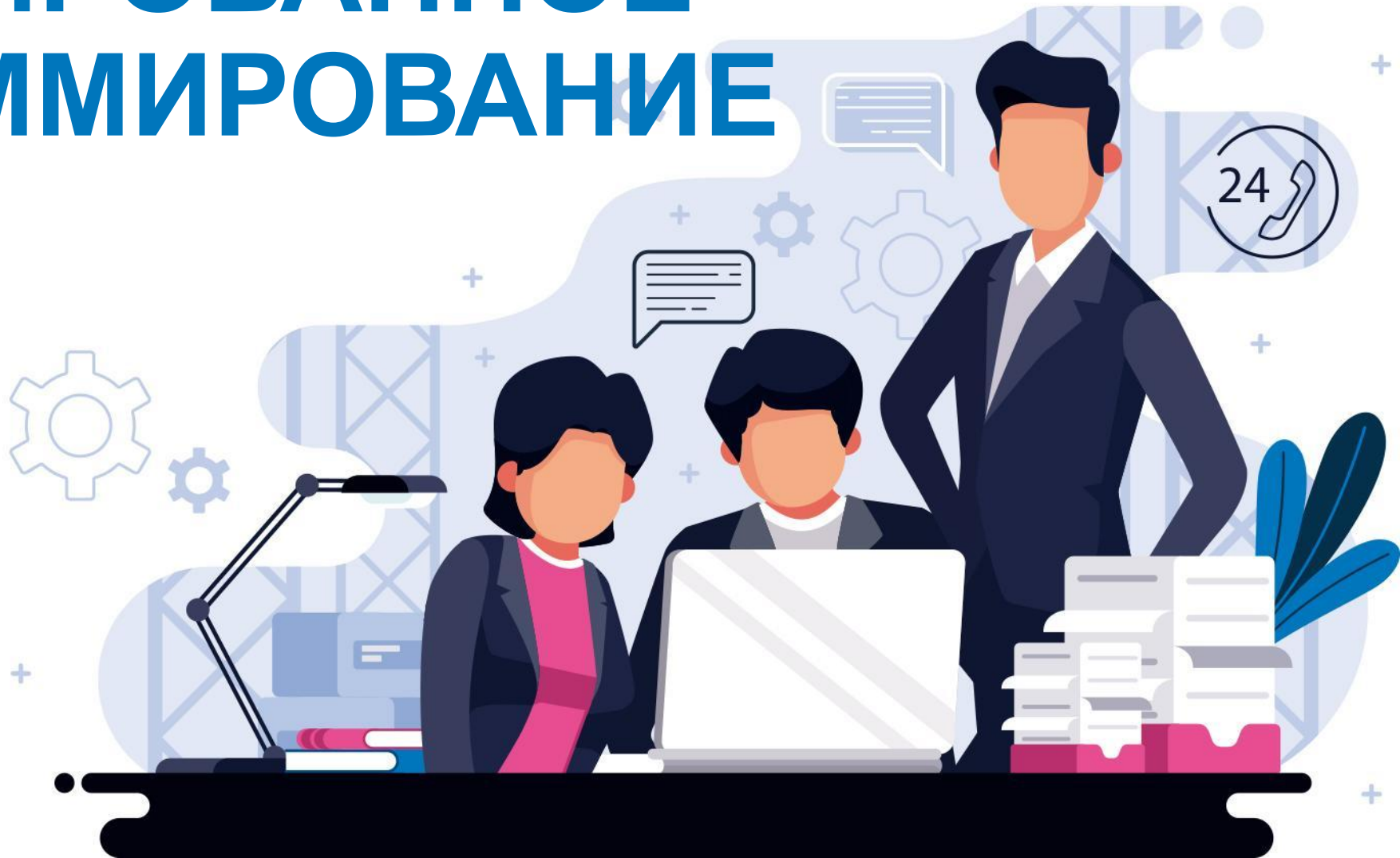
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лекция 10



План

- Конфигурация приложения Spring MVC + Hibernate
- Отображение списка работников
- Аннотация @Service
- Добавление работника

Конфигурация приложения Spring MVC + Hibernate

Прежде всего необходимо добавить зависимость от hibernate

https://mvnrepository.com/artifact/org.hibernate/hibernate-core

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.1.Final</version>
</dependency>
```

Конфигурация приложения Spring MVC + Hibernate

Далее необходимо добавить mysql connector

https://mvnrepository.com/artifact/mysql/mysql-connector-java

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.27</version>
</dependency>
```

Конфигурация приложения Spring MVC + Hibernate

Также нам потребудется c3p0

https://mvnrepository.com/artifact/com.mchange/c3p0

```
<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.5.2</version>
</dependency>
```

Конфигурация приложения Spring MVC + Hibernate

Spring ORM

https://mvnrepository.com/artifact/org.springframework/spring-orm

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.3.12</version>
</dependency>
```

Конфигурация приложения Spring MVC + Hibernate

Добавим некоторые настройки в файл applicationContext

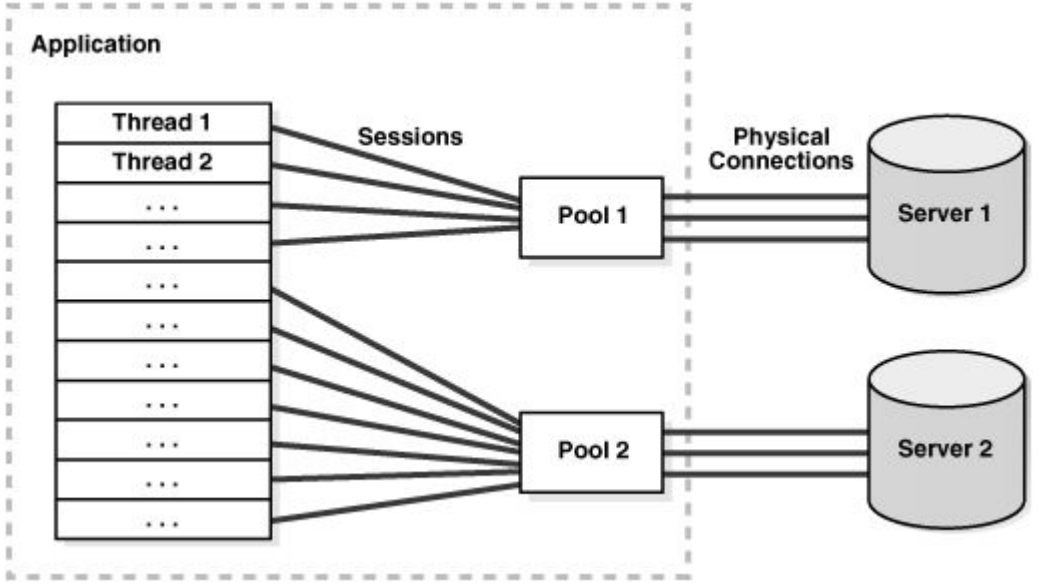
```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
      destroy-method="close">
    <property name="driverClass" value="com.mysql.cj.jdbc.Driver" />
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/my_db?useSSL=false&amp;serverTimezone=UTC" />
    <property name="user" value="root" />
    <property name="password" value="root" />
</bean>
```

Конфигурация приложения Spring MVC + Hibernate

Что такое c3p0?

c3p0 — ещё одна библиотека для создания пулов соединений. От HikariCP её отличает встроенная поддержка кэширования запросов, которая позволяет автоматически переиспользовать prepared statements, увеличивая тем самым производительность.

Пул соединений с базой данных это набор заранее открытых соединений с базой данных используемый для предоставления соединения в тот момент, когда оно требуется. Пулы соединений используются для повышения производительности при работе с базами данных.



Конфигурация приложения Spring MVC + Hibernate

Далее необходимо создать бин sessionFactory и указать, где будут располагаться entity

```
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="packagesToScan" value="com.donnu.spring.mvc.entity" />
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
</bean>
```

Конфигурация приложения Spring MVC + Hibernate

Также нам понадобится transactionManager

```
<bean id="transactionManager"
      class="org.springframework.orm.hibernate5.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>

<tx:annotation-driven transaction-manager="transactionManager" />
```

Конфигурация приложения Spring MVC + Hibernate

Полный текст:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
                        http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context.xsd
            http://www.springframework.org/schema/mvc
     http://www.springframework.org/schema/mvc/spring-mvc.xsd
                        http://www.springframework.org/schema/tx
                        http://www.springframework.org/schema/tx/spring-tx.xsd">
  <context:component-scan base-package="com.donnu.spring.mvc" />
  <mvc:annotation-driven/>
  <bean
       class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/view/" />
    <property name="suffix" value=".jsp" />
  </bean>
  <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
      destroy-method="close">
```

Отображение списка работников

Теперь необходимо создать класс, который будет отвечать за работу с таблицей в БД

```java
@Entity
@Table(name = "employees")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "name")
    private String name;

    @Column(name = "surname")
    private String surname;

    @Column(name = "department")
    private String department;

    @Column(name = "salary")
    private int salary;

    public Employee() {
    }
```

id	name	surname	department	salary
1	Ivan	Ivamov	IT	500
2	Oleg	Petrov	Sales	700
3	Nina	Sidorova	HR	850

Отображение списка работников

Создадим DAO.

DAO (**D**ata **A**ccess **O**bject) – это слой объектов которые обеспечивают доступ к данным.

Отображение списка работников

В соответствии с best practice необходимо создать интерфейс:

```java
package com.donnu.spring.mvc.dao;
import com.donnu.spring.mvc.entity.Employee;
import java.util.List;

public interface EmployeeDAO {
    public List<Employee> getAllEmployees();
}
```

И его реализацию:

```java
package com.donnu.spring.mvc.dao;
import com.donnu.spring.mvc.entity.Employee;
import java.util.List;

public class EmployeeDAOImplementation implements EmployeeDAO {
    @Override
    public List<Employee> getAllEmployees() {
        return null;
    }
}
```

Отображение списка работников

Теперь необходимо обеспечить доступ реализации к БД. Для этого ей понадобится доступ к sessionFactory.

```java
package com.donnu.spring.mvc.dao;
import com.donnu.spring.mvc.entity.Employee;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.List;

public class EmployeeDAOImplementation implements EmployeeDAO {
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public List<Employee> getAllEmployees() {
        return null;
    }
}
```

Отображение списка работников

Реализуем метод getAllEmployees. Укажем ему аннотацию **@Transactional**.

При использовании аннотации **@Transactional**, Spring берет на себя ответственность за открытие и закрытие транзакций.

Отображение списка работников

```java
package com.donnu.spring.mvc.dao;
import com.donnu.spring.mvc.entity.Employee;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

public class EmployeeDAOImplementation implements EmployeeDAO {
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    @Transactional
    public List<Employee> getAllEmployees() {

        Session session = sessionFactory.getCurrentSession();
        List<Employee> allEmployees = session.createQuery( s: "from Employee", Employee.class).getResultList();

        return allEmployees;
    }
}
```

Отображение списка работников

Способ получения 1:

```
Session session = sessionFactory.getCurrentSession();
List<Employee> allEmployees = session.createQuery( s: "from Employee", Employee.class).getResultList();
```

Способ получения 2:

```
Session session = sessionFactory.getCurrentSession();
Query<Employee> query = session.createQuery( s: "from Employee", Employee.class);
List<Employee> allEmployees = query.getResultList();
```

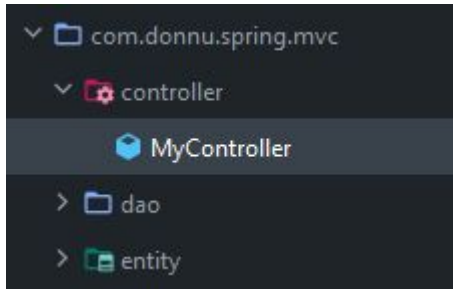
Отображение списка работников

@Repository – специализированный компонент (@Component). Данная аннотация используется для DAO. При поиске аннотаций Spring также будет регистрировать все DAO с аннотацией @Repository в Spring Container.

```
@Repository
public class EmployeeDAOImplementation implements EmployeeDAO {
```

Отображение списка работников

Теперь создадим контроллер

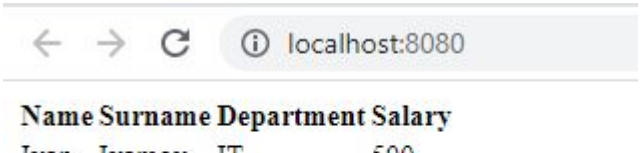



```java
@Controller
public class MyController {

    @Autowired
    private EmployeeDAO employeeDAO;

    @RequestMapping("/")
    public String showAllEmployees(Model model) {
        List<Employee> allEmployees = employeeDAO.getAllEmployees();

        model.addAttribute( attributeName: "allEmployees", allEmployees);

        return "all-employees";
    }
}
```

Отображение списка работников

Добавим страницу вывода

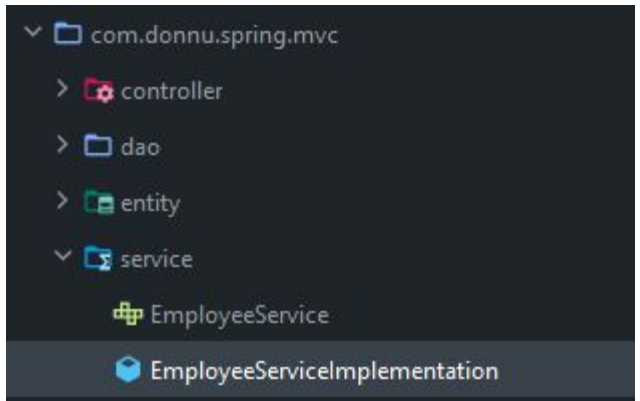
```jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <table>
        <tr>
            <th>Name</th>
            <th>Surname</th>
            <th>Department</th>
            <th>Salary</th>
        </tr>
        <c:forEach var="employee" items="${allEmployees}">
            <tr>
                <td>${employee.name}</td>
                <td>${employee.surname}</td>
                <td>${employee.department}</td>
                <td>${employee.salary}</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```



Аннотация @Service

В соответствии с best practice между контроллером и dao должен быть service

```java
package com.donnu.spring.mvc.service;

import com.donnu.spring.mvc.entity.Employee;

import java.util.List;

public interface EmployeeService {
    public List<Employee> getAllEmployees();
}
```

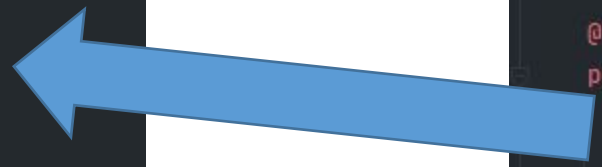



```java
package com.donnu.spring.mvc.service;
import com.donnu.spring.mvc.entity.Employee;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class EmployeeServiceImplementation  implements EmployeeService{
    @Override
    public List<Employee> getAllEmployees() {
        return null;
    }
}
```

Аннотация @Service

Аннотация **@Service** отмечает класс, содержащий бизнес-логику. В иерархии компонентов Service выступает связующим звеном между контроллером и DAO.

```java
@Service
public class EmployeeServiceImplementation  implements EmployeeService{

    @Autowired
    private EmployeeDAO employeeDAO;

    @Override
    public List<Employee> getAllEmployees() {
        return employeeDAO.getAllEmployees();
    }
}
```



```java
@Controller
public class MyController {

    @Autowired
    private EmployeeService employeeService;

    @RequestMapping("/")
    public String showAllEmployees(Model model) {
        List<Employee> allEmployees = employeeService.getAllEmployees();

        model.addAttribute( attributeName: "allEmployees", allEmployees);

        return "all-employees";
    }
}
```

Аннотация @Service

Также аннотацию @Transactional можно перенести из DAO в Service

```
@Service
public class EmployeeServiceImplementation  implements EmployeeService{

    @Autowired
    private EmployeeDAO employeeDAO;

    @Override
    @Transactional
    public List<Employee> getAllEmployees() {
        return employeeDAO.getAllEmployees();
    }
}
```

Добавление работника

Прежде всего добавим кнопку и метод перехода на необходимый view

```
<table>
    <tr>
        <th>Name</th>
        <th>Surname</th>
        <th>Department</th>
        <th>Salary</th>
    </tr>
    <c:forEach var="employee" items="${allEmployees}">
        <tr>
            <td>${employee.name}</td>
            <td>${employee.surname}</td>
            <td>${employee.department}</td>
            <td>${employee.salary}</td>
        </tr>
    </c:forEach>
</table>
<br>
<button onclick="window.location.href = 'addEmployee'">Add</button>
```

```
@RequestMapping("/addEmployee")
public String addEmployee(Model model) {

    Employee employee = new Employee();
    model.addAttribute( attributeName: "employee", employee);

    return "employee-info";
}
```

Добавление работника

Создадим форму:

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
<form:form action="saveEmployee" modelAttribute="employee">
    Name <form:input path="name"/>
    <br><br>
    Surname <form:input path="surname"/>
    <br><br>
    Salary <form:input path="salary"/>
    <br><br>
    Salary <form:input path="department"/>
    <br><br>
    <button>OK</button>
</form:form>
</body>
</html>
```

Добавление работника

Теперь сохраним работника

```java
public interface EmployeeDAO {
    public List<Employee> getAllEmployees();

    public void saveEmployee(Employee employee);
}
```

```java
public interface EmployeeService {
    public List<Employee> getAllEmployees();

    public void saveEmployee(Employee employee);
}
```

```java
@Repository
public class EmployeeDAOImplementation implements EmployeeDAO {
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public List<Employee> getAllEmployees() {...}

    @Override
    public void saveEmployee(Employee employee) {
        Session session = sessionFactory.getCurrentSession();
        session.save(employee);
    }
}
```

```java
@Service
public class EmployeeServiceImplementation  implements EmployeeService{

    @Autowired
    private EmployeeDAO employeeDAO;

    @Override
    @Transactional
    public List<Employee> getAllEmployees() { return employeeDAO.getAllEmployees(); }

    @Override
    @Transactional
    public void saveEmployee(Employee employee) { employeeDAO.saveEmployee(employee); }
}
```

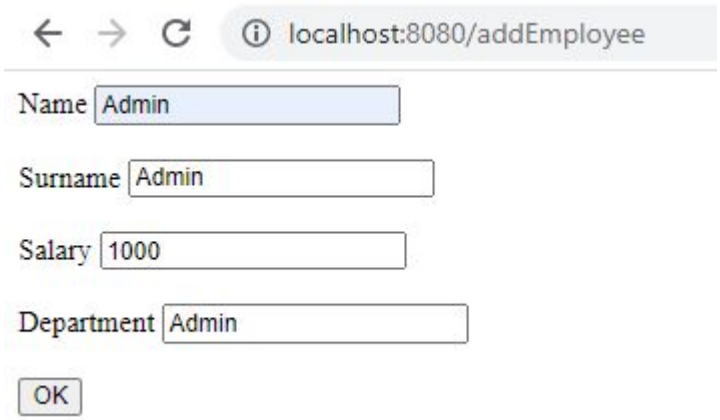
Добавление работника

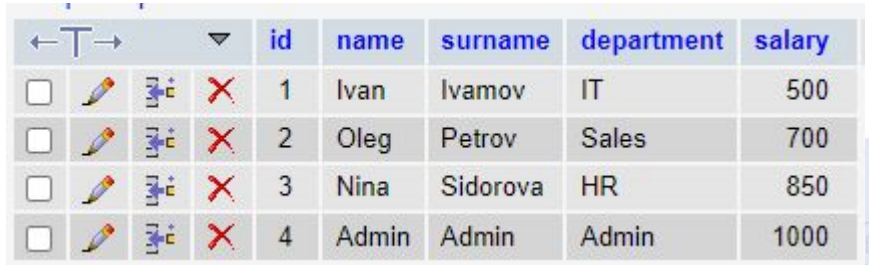
Теперь сохраним работника

```
@RequestMapping("/saveEmployee")
public String saveEmployee(@ModelAttribute("employee") Employee employee) {

    employeeService.saveEmployee(employee);

    return "redirect:/";
}
```

Добавление работника

Теперь сохраним работника





id	name	surname	department	salary
1	Ivan	Ivamov	IT	500
2	Oleg	Petrov	Sales	700
3	Nina	Sidorova	HR	850
4	Admin	Admin	Admin	1000