1. 创建项目文件结构

(1) 在src/main路径下创建java文件夹

(2) src/main/java文件夹下创建：

1. com.itsofttech.controller　控制层的包

2. com.itsofttech.dao　数据层的包

3. com.itsofttech.service 服务层的

4. com.itsofttech.pojo 封装实体类的包

(3) 在src/main路径下创建resources资源文件夹

(4)在src/路径下创建test文件夹，test文件夹下创建java文件夹，用于我们中间测试用

2. pom.xml添加springMVC的依赖

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.5.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.5.RELEASE</version>
</dependency>
```

3. web.xml头文件

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
          http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0"
         metadata-complete="true">

</web-app>
```

4. web.xml的springMVC配置

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
          http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0"
         metadata-complete="true">

    <!-- DispatchServleet -->
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- 引入xml文件 -->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:applicationContext.xml</param-value>
        </init-param>
        <!--启动级别-1-->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <!--/ 匹配所有的请求；（不包括.jsp）--><!--/* 匹配所有的请求；（包括.jsp）-->
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

5. resources资源文件夹下创建applicationContext.xml

6. 配置applicationContext.xml

```
<?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:p="http://www.springframework.org/schema/p"
xmlns:aop="http://www.springframework.org/schema/aop"
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/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/aop/spring-tx.xsd
http://www.springframework.org/schema/mvc
```

```
http://www.springframework.org/schema/aop/spring-mvc-4.0.xsd">

    <import resource="classpath:spring-mvc.xml"/>
    <import resource="classpath:spring-dao.xml"/>
    <import resource="classpath:spring-service.xml"/>
</beans>
```

7. resources资源文件夹下创建spring-mvc.xml

8. 配置spring-mvc.xml文件

```
<?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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/mvc
       https://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!-- 注解驱动 -->
    <mvc:annotation-driven />
    <!-- SpringMVC静态资源过滤 .css .js .html .mp3 .mp4 -->
    <mvc:default-servlet-handler/>
    <!-- 自动扫描包，让指定包下注解生效，由IOC容器统一管理 -->
    <context:component-scan base-package="com.itsofttech.controller"/>

    <!-- 视图解析器:模板引擎 ThymeLeaf FreeMarker -->
     <bean id="InternalResourceViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
         <!-- 前缀 -->
        <property name="prefix" value="/WEB-INF/admin/"/>
        <!-- 后缀 -->
        <property name="suffix" value=""/>
     </bean>
    <bean id="validCode" class="com.itsofttech.captcha.ValidCode"/>

    <!-- 拦截器配置 -->
    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/**"/>
            <mvc:exclude-mapping path="/**/fonts/*"/>
            <mvc:exclude-mapping path="/**/*.css"/>
            <mvc:exclude-mapping path="/**/*.xls"/>
            <mvc:exclude-mapping path="/**/*.xlsx"/>
            <mvc:exclude-mapping path="/**/*.js"/>
            <mvc:exclude-mapping path="/**/*.png"/>
            <mvc:exclude-mapping path="/**/*.gif"/>
            <mvc:exclude-mapping path="/**/*.jpg"/>
```

```
            <mvc:exclude-mapping path="/**/*.ico"/>
            <mvc:exclude-mapping path="/**/*.jpeg"/>
            <mvc:exclude-mapping path="/**/*login*"/>
            <mvc:exclude-mapping path="/**/admin/"/>
            <bean class="com.itsofttech.interceptor.EmsInterceptor"></bean>
        </mvc:interceptor>
    </mvc:interceptors>

    <!-- JSON乱码 -->
    <mvc:annotation-driven>
        <mvc:message-converters register-defaults="true">
            <bean
class="org.springframework.http.converter.StringHttpMessageConverter">
                <constructor-arg value="UTF-8"/>
            </bean>
            <bean
class="org.springframework.http.converter.json.MappingJackson2HttpMessageConvert
er">
                <property name="objectMapper">
                    <bean
class="org.springframework.http.converter.json.Jackson2ObjectMapperFactoryBean">
                        <property name="failOnEmptyBeans" value="false"/>
                    </bean>
                </property>
            </bean>
        </mvc:message-converters>
    </mvc:annotation-driven>

</beans>
```

9. pom.xml添加数据库驱动依赖

mysql官方提供的操作mysql数据库的类包

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.18</version>
</dependency>
```

10. pom.xml添加c3p0数据源

负责控制与数据库的连接

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

11. 添加mybatis依赖

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.4</version>
</dependency>
```

12. 添加mybatis-spring依赖

管理sql语句

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>2.0.5</version>
</dependency>
```

13. 添加log4j2日志输出依赖

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.10.0</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.10.0</version>
</dependency>

<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-web</artifactId>
    <version>2.9.1</version>
</dependency>
```

14. log4j2的web.xml配置

```
<!-- log4j2-begin -->
     <listener>
         <listener-
class>org.apache.logging.log4j.web.Log4jServletContextListener</listener-class>
    </listener>
     <filter>
         <filter-name>log4jServletFilter</filter-name>
```

```
        <filter-class>org.apache.logging.log4j.web.Log4jServletFilter</filter-
class>
    </filter>
    <filter-mapping>
        <filter-name>log4jServletFilter</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
        <dispatcher>ERROR</dispatcher>
    </filter-mapping>
<!-- log4j2-end -->
```

15. log4j2.xml配置

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="error">
    <!--先定义所有的appender -->
    <appenders>
        <!--这个输出控制台的配置 -->
        <Console name="Console" target="SYSTEM_OUT">
            <!--             控制台只输出level及以上级别的信息（onMatch），其他的直接拒
绝（onMismatch） -->
            <ThresholdFilter level="debug" onMatch="ACCEPT" onMismatch="DENY"/>
            <!--             这个都知道是输出日志的格式 -->
            <PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %class{36} %L %M -
%msg%xEx%n"/>
        </Console>

        <!--文件会打印出所有信息，这个log每次运行程序会自动清空，由append属性决定，这个也挺
有用的，适合临时测试用 -->
        <!--append为TRUE表示消息增加到指定文件中，false表示消息覆盖指定的文件内容，默认值是
true -->
        <File name="log" fileName="D:/logs/log4j2.log" append="false">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %class{36} %L %M -
%msg%xEx%n"/>
        </File>

        <!--添加过滤器ThresholdFilter,可以有选择的输出某个级别以上的类别
onMatch="ACCEPT" onMismatch="DENY"意思是匹配就接受,否则直接拒绝  -->
        <File name="ERROR" fileName="D:/logs/error.log">
            <ThresholdFilter level="error" onMatch="ACCEPT" onMismatch="DENY"/>
            <PatternLayout pattern="%d{yyyy.MM.dd 'at' HH:mm:ss z} %-5level
%class{36} %L %M - %msg%xEx%n"/>
        </File>

        <!--这个会打印出所有的信息，每次大小超过size，则这size大小的日志会自动存入按年份-月
份建立的文件夹下面并进行压缩，作为存档 -->
        <RollingFile name="RollingFile" fileName="D:/logs/web.log"
                     filePattern="logs/$${date:yyyy-MM}/web-%d{MM-dd-yyyy}-
%i.log.gz">
            <PatternLayout pattern="%d{yyyy-MM-dd 'at' HH:mm:ss z} %-5level
%class{36} %L %M - %msg%xEx%n"/>
            <SizeBasedTriggeringPolicy size="2MB"/>
```

```
        </RollingFile>
    </appenders>


    <!--然后定义logger，只有定义了logger并引入的appender，appender才会生效 -->
    <loggers>
        <root level="debug">
            <appender-ref ref="RollingFile"/>
            <appender-ref ref="Console"/>
            <appender-ref ref="ERROR" />
            <appender-ref ref="log"/>
        </root>
    </loggers>
</configuration>
```

16. 添加slf4j依赖

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.30</version>
</dependency>
```

17. 添加Mybatis缓存依赖

```
<dependency>
    <groupId>org.mybatis.caches</groupId>
    <artifactId>mybatis-ehcache</artifactId>
    <version>1.1.0</version>
</dependency>
```

18. resources资源文件夹下创建database.properties属性文件

```
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/cms?
useSSL=true&useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
user=root
password=itsofttech
```

19. resources资源文件夹下创建mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
```

```
                PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
                "http://mybatis.org/dtd/mybatis-3-config.dtd">
<!--configuration-->
<configuration>
    <settings>
        <!-- 显示开启全局缓存，默认就是开启的 -->
        <setting name="cacheEnabled" value="true"/>
    </settings>
    <!-- 设置EmployeeMapper.xml 内封装类的路径 -->
    <typeAliases>
        <package name="com.itsofttech.pojo"/>
    </typeAliases>

   <mappers>
        <mapper class="com.itsofttech.dao.ClaimMapper"/>
        <mapper class="com.itsofttech.dao.EmployeeMapper"/>
    </mappers>
</configuration>
```

20. resources资源文件夹下创建spring-dao.xml

21. 配置spring-dao.xml文件

```
<?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"
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
    ">

<context:property-placeholder location="classpath:database.properties"/>
    <!-- 连接池
    dbcp 半自动化，不能自动连接
    c3p0 自动化操作，自动加载配置文件，并且自动设置到对象里。
    druid
    hirika -> springboot
    用于控制和数据的连接-->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${driver}"/>
        <property name="jdbcUrl" value="${url}"/>
        <property name="user" value="${user}"/>
        <property name="password" value="${password}"/>
        <!-- c3p0连接池的私有属性 -->
        <property name="maxPoolSize" value="1"/>
        <property name="minPoolSize" value="1"/>
        <!-- 关闭连接后不自动commit -->
        <property name="autoCommitOnClose" value="false"/>
        <!-- 获取连接超时时间 -->
        <property name="checkoutTimeout" value="10000"/>
```

```
        <!-- 当获取连接失败重试次数 -->
        <property name="acquireRetryAttempts" value="2"/>
    </bean>

    <!-- sqlSessionFactory 负责把数据源和mybatis关联-->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <!-- 绑定Mybatis配置文件-->
        <property name="configLocation" value="classpath:mybatis-config.xml"/>
    </bean>

    <!-- 配置dao接口扫描 包，动态实现Dao接口可以注入到Spring容器中-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!-- 注入sqlSessionFactory -->
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
        <!-- 扫描要扫描的包 -->
        <property name="basePackage" value="com.itsofttech.dao"/>
    </bean>
</beans>
```

22. 在com.itsofttech.dao 数据层的包下创建接口 EmployeeMapper

```
public interface EmployeeMapper
{
    Employee selectEmployee(@Param("email")String email);
    int updatePassword(Map<String,String> map);
}
```

23. 创建Employee封装实体类

```
package com.itsofttech.pojo;

import java.io.Serializable;

public class Employee implements Serializable
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private String employeeID;
    private String employeeName;
    private String password;
    private String status;
    private String sex;
    private String birthday;
    private String age;
    private String joinedDate;
```

```
    private String joinedTime;
    private String postCode;
    private String address;
    private String phoneNumber;
    private String authority;
    private String mailAdress;
    private String insertDate;
    private String updateDate;

    public Employee()
    {
    }
    public Employee(String employeeID, String employeeName, String password,
String status, String sex, String birthday,
            String age, String joinedDate, String joinedTime, String postCode,
String address, String phoneNumber,
            String authority, String mailAdress, String insertDate, String
updateDate) {
        super();
        this.employeeID = employeeID;
        this.employeeName = employeeName;
        this.password = password;
        this.status = status;
        this.sex = sex;
        this.birthday = birthday;
        this.age = age;
        this.joinedDate = joinedDate;
        this.joinedTime = joinedTime;
        this.postCode = postCode;
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.authority = authority;
        this.mailAdress = mailAdress;
        this.insertDate = insertDate;
        this.updateDate = updateDate;
    }

    public String getEmployeeID() {
        return employeeID;
    }
    public void setEmployeeID(String employeeID) {
        this.employeeID = employeeID;
    }
    public String getEmployeeName() {
        return employeeName;
    }
    public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getStatus() {
        return status;
    }
```

```
    public void setStatus(String status) {
        this.status = status;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public String getBirthday() {
        return birthday;
    }
    public void setBirthday(String birthday) {
        this.birthday = birthday;
    }
    public String getAge() {
        return age;
    }
    public void setAge(String age) {
        this.age = age;
    }
    public String getJoinedDate() {
        return joinedDate;
    }
    public void setJoinedDate(String joinedDate) {
        this.joinedDate = joinedDate;
    }
    public String getJoinedTime() {
        return joinedTime;
    }
    public void setJoinedTime(String joinedTime) {
        this.joinedTime = joinedTime;
    }
    public String getPostCode() {
        return postCode;
    }
    public void setPostCode(String postCode) {
        this.postCode = postCode;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getPhoneNumber() {
        return phoneNumber;
    }
    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
    public String getAuthority() {
        return authority;
    }
    public void setAuthority(String authority) {
        this.authority = authority;
    }
    public String getMailAdress() {
```

```
        return mailAdress;
    }
    public void setMailAdress(String mailAdress) {
        this.mailAdress = mailAdress;
    }
    public String getInsertDate() {
        return insertDate;
    }
    public void setInsertDate(String insertDate) {
        this.insertDate = insertDate;
    }
    public String getUpdateDate() {
        return updateDate;
    }
    public void setUpdateDate(String updateDate) {
        this.updateDate = updateDate;
    }

    @Override
    public String toString() {
        return "Employee [employeeID=" + employeeID + ", employeeName=" +
employeeName + ", password=" + password
                + ", status=" + status + ", sex=" + sex + ", birthday=" +
birthday + ", age=" + age + ", joinedDate="
                + joinedDate + ", joinedTime=" + joinedTime + ", postCode=" +
postCode + ", address=" + address
                + ", phoneNumber=" + phoneNumber + ", authority=" + authority +
", mailAdress=" + mailAdress
                + ", insertDate=" + insertDate + ", updateDate=" + updateDate +
"]";
    }
}
```

24. 在com.itsofttech.dao包里建和接口名字相同的EmployeeMapper.xml

负责sql语句

```
<?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">

<!--namespace=Dao/Mapper 连接接口-->
<mapper namespace="com.itsofttech.dao.EmployeeMapper">

    <!-- 使用二级缓存 -->
    <cache type="org.mybatis.caches.ehcache.EhcacheCache"/>

    <!--id是接口里的方法名  parameterType是参数类型 resultType是返回类型#{取方法参数}-->
    <select id="selectEmployee" parameterType="string" resultType="Employee">
        select * from cms.employee where mailAdress=#{email}
    </select>
```

```
    <update id="updatePassword" parameterType="map">
        update cms.employee set password = #{password} where mailAdress=#{email}
    </update>
</mapper>
```

25. com.itsofttech.service 服务层下创建EmployeeService接口

EmployeeService接口要与EmployeeMapper接口的方法完全相同

```
public interface EmployeeService
{
    Employee queryEmployee(String email);
    int updatePassword(Map<String,String> map);
}
```

26. 创建EmployeeService接口的实现类

我们就使用这个实现类，通过EmployeeMapper 类型的变量调用对应方法，也可以在需要的时候，在这个类及方法里添加一些订制功能

```
package com.itsofttech.service;

import java.util.Map;
import com.itsofttech.dao.EmployeeMapper;
import com.itsofttech.pojo.Employee;

public class EmployeeServiceImpl implements EmployeeService
{
    EmployeeMapper employeeMapper;

    @Override
    public Employee selectEmployee(String email)
    {

        return employeeMapper.selectEmployee(email);
    }

    @Override
    public int updatePassword(Map<String, String> map)
    {

        return employeeMapper.updatePassword(map);
    }

    public EmployeeMapper getEmployeeMapper() {
        return employeeMapper;
    }

    public void setEmployeeMapper(EmployeeMapper employeeMapper) {
        this.employeeMapper = employeeMapper;
```

```
    }

}
```

27. 创建spring-service.xml

主要是针对service层的服务器配置和bean管理

```
<?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"

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
    ">

    <!-- 将我们的所有业务类，注入到spring,可以通过配置，或者注解实现 -->
    <bean id="EmployeeServiceImpl"
class="com.itsofttech.service.EmployeeServiceImpl">
        <property name="employeeMapper" ref="employeeMapper"/>
    </bean>

    <!-- 声明式事务配置 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <!-- 为事务管理也注入数据源 -->
        <property name="dataSource" ref="dataSource"/>
    </bean>

</beans>
```

28. 添加jackson依赖

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.10.3</version>
</dependency>
```

29. 创建json数据的封装类LoginInformation

```
package com.itsofttech.pojo;

public class LoginInformation
```

```
{
    private String user;
    private String password;

    public String getUser()
    {
        return user;
    }
    public void setUser(String user)
    {
        this.user = user;
    }
    public String getPassword()
    {
        return password;
    }
    public void setPassword(String password)
    {
        this.password = password;
    }
}
```

30. com.itsofttech.controller控制层的包下创建IndexController类

```
package com.itsofttech.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.itsofttech.pojo.Employee;
import com.itsofttech.pojo.LoginInformation;
import com.itsofttech.service.EmployeeService;


//@RestController 标注类不会被视图解析器解析,可方便内部执行
//@RequestMapping("/hello")
@Controller
public class IndexController
{
    //@Autowired 注入的意思，向下边的变量自动注入，匹配类型给的bean。
    //@Qualifier("EmployeeServiceImpl") 和@Autowired 配合，指定匹配的bean的id是
EmployeeServiceImpl的对象
    @Autowired
    @Qualifier("EmployeeServiceImpl")
    private EmployeeService employeeService;

    //响应多个url请求
```

```
    @RequestMapping({"/","/index"})
    public String index()
    {
        return "index/index";
    }
    @RequestMapping("/cms")
    public String cms()
    {
        return "cms/cms";
    }
    //@RequestParam("user")把页面传递来的user参数绑定到参数data
    //@ResponseBody 把方法的返回值用字符串的形式返回，如果没有，就是作为地址的方式返回
    //@RequestMapping("/login") 映射浏览器访问的地址
    @RequestMapping("/login")
    @ResponseBody
    public String loginVerify(@RequestParam("data") String data) throws
JsonMappingException, JsonProcessingException
    {
        //通过jackson依赖创建ObjectMapper类对象
        ObjectMapper jsonMapper=new ObjectMapper();
        //解析
        LoginInformation loginInformation = jsonMapper.readValue(data,
LoginInformation.class);
        Employee
employee=employeeService.selectEmployee(loginInformation.getUser());
        if(employee==null)
        {
            return "null";
        }
        else if(employee.getPassword().equals(loginInformation.getPassword()))
        {
            System.out.println(employee);
            return "true";
        }
        else
        {
            return "false";
        }
    }
}
```

*配置根路径为“/”

服务器servers.xml文件里

```
      <Context docBase="cms" path="/" reloadable="true"
source="org.eclipse.jst.j2ee.server:cms"/></Host>
```