服务端第五次作业说明文档

1. 姓名：张洪胤

> 1. 清空web.xml配置，改成java配置类

2. 编写MyWebAppInitializer类

```
package com.example.config;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import org.springframework.web.servlet.DispatcherServlet;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;

public class MyWebAppInitializer implements WebApplicationInitializer {

    /**
     * Servlet容器启动时会自动运行该方法
     */
    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {

        servletContext.setInitParameter("contextConfigLocation", "classpath:applicationContext.xml");

        ServletRegistration.Dynamic registration = servletContext.addServlet("viewspace", new DispatcherServlet());
        registration.setLoadOnStartup(3);
        registration.addMapping("*.html");

        servletContext.addListener(new ContextLoaderListener());
    }
}
```

2. 删除web.xml文件，运行项目，可以正常按路径访问

> 2. 优化控制层的loginCheck方法，并且添加数据校验功能（@Valid），用户名3-6个字符，口令6个字
> 符，并有错误提示

1. 在pom.xml文件中导入实现@Valid数据校验功能所需的依赖。

```xml
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.2.4.Final</version>
</dependency>
```

2. 在applicationContext.xml文件中添加相应的Bean配置

```xml
<bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
  <property name="providerClass" value="org.hibernate.validator.HibernateValidator"/>
</bean>
```

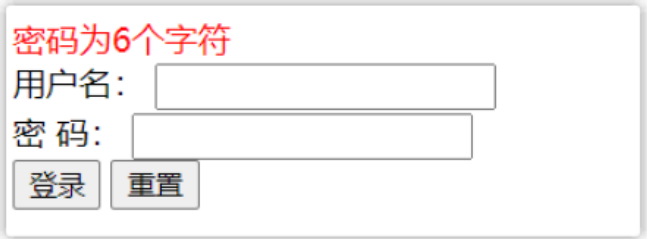
3. 按照题目要求，使用@Size在的LoginInfo相应字段上添加校验规则，并添加message

```java
public class LoginInfo {
    @NotNull
    @Size(min=3, max=6, message = "用户名3-6个字符")
    private String userName;

    @NotNull
    @Size(min=6, max=6, message = "口令为6个字符")
    private String password;

    public String getPassword() {
        return password;
    }
```

4. 修改控制层的loginCheck方法，在参数loginInfo 前加上@Valid 注解表明需要进行校验，新增参数BindingResult bindingResult 用于监视校验结果，并根据是否有错误信息，返回相应的ModelAndView

```java
@RequestMapping(value = "/loginCheck.html")
public ModelAndView loginCheck(HttpServletRequest request, @Valid LoginInfo loginInfo,
                               BindingResult bindingResult) {
    if(bindingResult.hasErrors()) {
        return new ModelAndView("login", "error", bindingResult.getAllErrors().get(0).getDefaultMessage());
    }

    boolean isValidUser =
            userService.hasMatchUser(loginInfo.getUserName(),
                    loginInfo.getPassword());
    if (!isValidUser) {
        return new ModelAndView("login", "error", "用户名或密码错误。");
    } else {
        User user = userService.findUserByUserName(loginInfo
                .getUserName());
        user.setLastIp(request.getLocalAddr());
        user.setLastVisit(new Date());
        userService.saveLog(user);
        request.getSession().setAttribute("user", user);
        return new ModelAndView("main");
    }
}
```

5. 错误输入结果



> 3. 数据库由原来的mysql改成H2内嵌数据库，不要有外部数据库访问依赖，exampledb.sql数据脚本同
> 步修改

1. 添加相应的pom依赖

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.182</version>
</dependency>
```

2. 在applicationContext.xml文件中添加配置，并在resource下放入schema.sql 和data.sql 两个数据脚本， schema.sql 用于建立数据表， data.sql 用于插入数据

3. 检查结果：通过检查，数据库可以正常的完成访问

> 4. DAO层实现由现在的jdbc改成JpaRepository自动实现，方法名可以改变

1. pom.xml中导入JPA的依赖

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>1.3.2.RELEASE</version>
</dependency>
```

2. 在applicationContext.xml文件中添加JPA配置

```
<bean id="emf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
    p:dataSource-ref="dataSource"
    p:persistenceUnitName="com"
    p:jpaVendorAdapter-ref="jpaVendorAdapter"
    p:packagesToScan="com.example.domain" />

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager"
    p:entityManagerFactory-ref="emf" />
```

3. 修改domain文件夹下的LoginLog 和User 两个实体类，与数据库的表进行一一映射

4. 重新编写dao层中的LoginLogDao 和UserDao ， 通过JpaRepository自动实现，UserDao中重写了getMatchCount方法

```
package com.example.dao;

import com.example.domain.LoginLog;
import org.springframework.data.jpa.repository.JpaRepository;

public interface LoginLogDao extends JpaRepository<LoginLog, Long> {
}
```

```
package com.example.dao;

import com.example.domain.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

/**
 * @Author stormbroken
 * Create by 2021/04/04
 * @Version 1.0
 **/

public interface UserDao extends JpaRepository<User, Long> {
    @Query(value = "select count(*) from t_user where user_name = ?1 and password = ?2", nativeQuery=true)
    Object getMatchCount(String userName, String password);

    @Query(value = "select * from t_user where user_name = ?1",nativeQuery=true)
    User findUserByUserName(String username);
}
```

5. 同步修改UserService 和UserDaoTest
6. 检查测试修改无误

> 5. DAO层的findUserByUserName添加缓存功能，缓存用EhCache实现

1. pom.xml中引入EhCache依赖

```
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>2.7.4</version>
</dependency>
```

2. 在resources中添加cache.xml配置cache，并创建对应的缓存目录(d:/ehcache)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd">

    <!-- 磁盘缓存位置 -->
    <diskStore path="d:/ehcache"/>

    <!-- 默认缓存 -->
    <defaultCache
            maxEntriesLocalHeap="10000"
            eternal="false"
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            maxEntriesLocalDisk="10000000"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU">
        <persistence strategy="localTempSwap"/>
    </defaultCache>

    <!-- helloworld缓存 -->
    <cache name="HelloWorldCache"
           maxElementsInMemory="1"
           eternal="true"
           timeToIdleSeconds="5"
           timeToLiveSeconds="5"
           overflowToDisk="false"
           diskPersistent="true"
           memoryStoreEvictionPolicy="LRU"/>
    <cache name="users"
           maxBytesLocalHeap="50m"
           timeToLiveSeconds="100">
    </cache>
</ehcache>
```

3. 添加EhCacheConfig.java 配置文件，对EhCache进行配置，开启缓存

```java
package com.example.config;

import net.sf.ehcache.CacheManager;
import org.springframework.cache.annotation.CachingConfigurerSupport;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cache.ehcache.EhCacheCacheManager;
import org.springframework.cache.ehcache.EhCacheManagerFactoryBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;

@Configuration
@EnableCaching
public class EhCacheConfig extends CachingConfigurerSupport {

    @Bean
    public EhCacheCacheManager cacheManager(CacheManager cm) {
        return new EhCacheCacheManager(cm);
    }

    @Bean
    public EhCacheManagerFactoryBean ehcache() {
        EhCacheManagerFactoryBean ehCacheFactoryBean =
                new EhCacheManagerFactoryBean();
        ehCacheFactoryBean.setConfigLocation(
                new ClassPathResource("cache.xml"));
        return ehCacheFactoryBean;
    }
}
```

4. 对findUserByUserName添加缓存

```java
@Cacheable(value = "users")
public User findUserByUserName(String userName) {
    return userDao.findUserByUserName(userName);
}
```

5. 测试发现对应的缓存目录出现data和index文件，并且缓存成功被使用到。

> 6. 测试改进：service层的测试将现在直连数据库改成使用mock取代dao层

1. 使用mock修改UserServiceTest 代码

```
@Test
public void hasMatchUser() {
    when(userService.hasMatchUser("admin", "123456")).thenReturn(true);
    boolean b1 = userService.hasMatchUser("admin", "123456");
    boolean b2 = userService.hasMatchUser("admin", "1111");
    assertTrue(b1);
    assertFalse(b2);
}


@Test
public void findUserByUserName() {
    User user = new User();
    user.setUserName("admin");
    when(userService.findUserByUserName("admin")).thenReturn(user);
    user = userService.findUserByUserName("admin");
    assertEquals(user.getUserName(), "admin");
}

@Test
public void loginSuccess() {
    User user = new User();
    user.setUserName("admin");
    when(userService.findUserByUserName("admin")).thenReturn(user);
    user = userService.findUserByUserName("admin");
    user.setUserId(1);
    user.setUserName("admin");
    user.setLastIp("192.168.12.7");
    user.setLastVisit(new Date());
    when(userService.saveLog(user)).thenReturn(true);
    assertEquals(true, userService.saveLog(user));
}
```

2. 执行测试用例，并全部通过