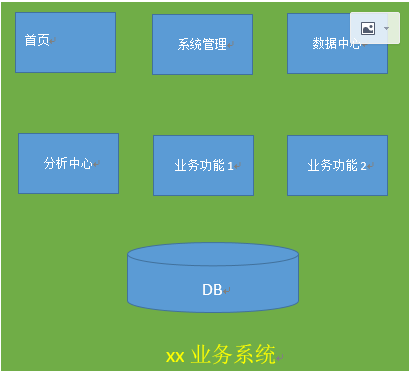
分布式系统架构设计和实现

课程类型：系统架构类

# 一、整体介绍

普通的项目架构



有什么问题？

1、    模块之间耦合度太高，其中一个升级其他都得升级

2、    开发困难，各个团队开发最后都要整合一起

3、    系统的扩展性差

4、    不能灵活的进行分布式部署。

解决方法：

优点：

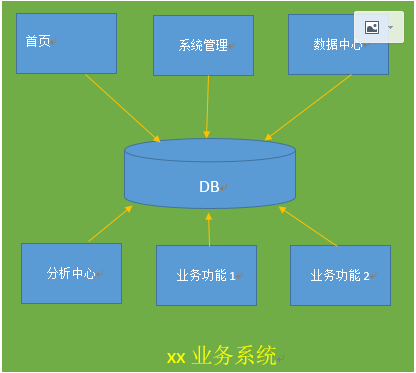
把模块拆分成独立的工程，单点运行。如果某一个点压力大可以对这一个点单独增加配置。其他的点不受影响。

缺点：

系统之间交互需要额外的工作量来进行接口的开发。

把系统拆分成多个工程，要完成系统的工程需要多个工程协作完成。这种形式叫做分布式。

分布式系统架构：



分布式架构：

把系统按照模块拆分成多个子系统。

优点：

1、把模块拆分，使用接口通信，降低模块之间的耦合度。

2、把项目拆分成若干个子项目，不同的团队负责不同的子项目。

3、增加功能时只需要再增加一个子项目，调用其他系统的接口就可以。

4、可以灵活的进行分布式部署。

缺点：

系统之间交互需要使用远程通信，接口开发增加工作量。

课程规划：

一、整体介绍

二、规划工程结构、使用maven进行构建

三、分布式框架搭建-SSM工程搭建

四、分布式框架搭建-SSM的测试

五、Restful原理分析和服务工程搭建

六、Restful服务发布

七、Restful服务测试

八、使用HttpClient实现系统之间服务调用

九、业务功能缓存的实现-redis单机版安装

十、业务功能缓存的实现-redis集群环境搭建

十一、redis单机和集群环境测试

十二、spring和redis的集成

十三、使用redis实现业务缓存

十四、单点登录SSO原理分析

十五、单点登录SSO工程搭建

十六、单点登录SSO服务规划

十七、单点登录SSO的服务发布

十八、单点登录SSO的注册功能实现

十九、单点登录SSO的登录功能实现

二十、业务系统与单点登录系统的整合

技术选型（主要技术）

    Spring、SpringMVC、Mybatis

    JSP、jQuery

    httpclient（调用系统服务）

Mysql

redis

开发工具和环境

STS（Eclipse）

Maven 3.5

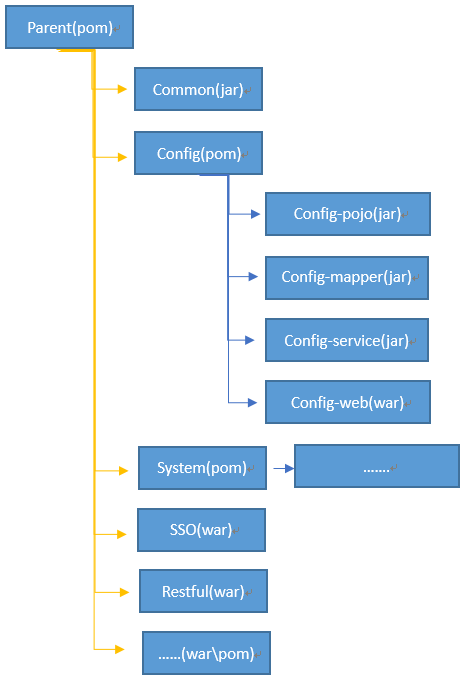
Tomcat 7.0.53（Maven Tomcat Plugin）

JDK 1.8

Mysql 5.6

# 二、规划工程结构、使用maven进行构建

工程结构图：



在搭建之前，需要大家熟悉maven这个工具。

工程类型有三种：

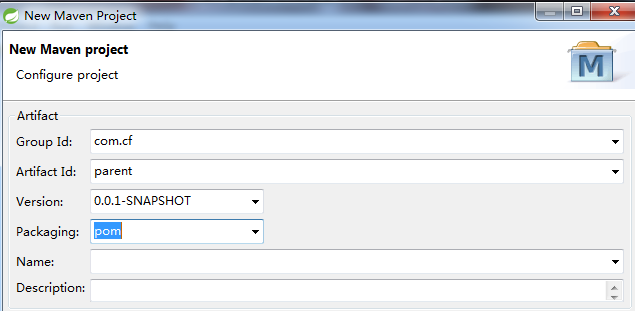
War包工程

Jar包工程

Pom工程

## 1、创建所有工程的父工程parent，用于定义版本管理定义

工程类型：pom



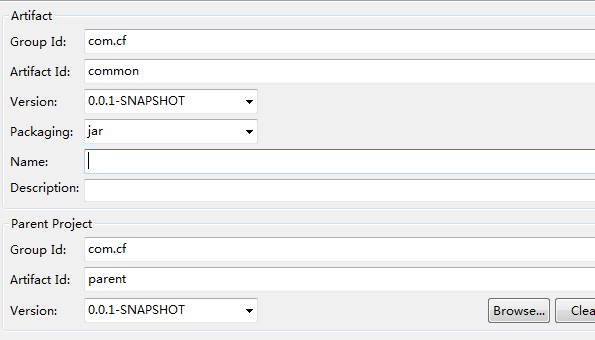
Pom.xml定义

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.cf</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging>  <!-- 集中定义依赖版本号 -->  <properties>  <junit.version>4.12</junit.version>  <spring.version>4.1.3.RELEASE</spring.version>  <mybatis.version>3.2.8</mybatis.version>  <mybatis.spring.version>1.2.2</mybatis.spring.version>  <mybatis.paginator.version>1.2.15</mybatis.paginator.version>  <mysql.version>5.1.32</mysql.version>  <slf4j.version>1.6.4</slf4j.version>  <jackson.version>2.4.2</jackson.version>  <druid.version>1.0.9</druid.version>  <httpclient.version>4.3.5</httpclient.version>  <jstl.version>1.2</jstl.version>  <servlet-api.version>2.5</servlet-api.version>  <jsp-api.version>2.0</jsp-api.version>  <commons-lang3.version>3.3.2</commons-lang3.version>  <commons-io.version>1.3.2</commons-io.version>  <commons-net.version>3.3</commons-net.version>  <pagehelper.version>4.2.1</pagehelper.version>  <jsqlparser.version>0.9.1</jsqlparser.version>  <commons-fileupload.version>1.3.1</commons-fileupload.version>  <jedis.version>2.7.2</jedis.version>  <json.version>1.2.23</json.version>  </properties>  <!-- 只定义依赖的版本，并不实际依赖 -->  <dependencyManagement>  <dependencies>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>${spring.version}</version>  </dependency>  <!-- Apache工具组件 -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>${commons-lang3.version}</version>  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-io</artifactId>  <version>${commons-io.version}</version>  </dependency>  <dependency>  <groupId>commons-net</groupId>  <artifactId>commons-net</artifactId>  <version>${commons-net.version}</version>  </dependency>  <!-- Jackson Json处理工具包 -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>${jackson.version}</version>  </dependency>  <!-- httpclient -->  <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpclient</artifactId>  <version>${httpclient.version}</version>  </dependency>  <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  <!-- 日志处理 -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <!-- Mybatis -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${mybatis.version}</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${mybatis.spring.version}</version>  </dependency>  <dependency>  <groupId>com.github.miemiedev</groupId>  <artifactId>mybatis-paginator</artifactId>  <version>${mybatis.paginator.version}</version>  </dependency>  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>${pagehelper.version}</version>  </dependency>  <!-- MySql -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <!-- 连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>${druid.version}</version>  </dependency>  <!-- JSP相关 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>${jstl.version}</version>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>${servlet-api.version}</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <version>${jsp-api.version}</version>  <scope>provided</scope>  </dependency>  <!-- 文件上传组件 -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>${commons-fileupload.version}</version>  </dependency>  <!-- Redis客户端 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>${jedis.version}</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>${json.version}</version>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <finalName>${project.artifactId}</finalName>  <plugins>  <!-- 资源文件拷贝插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-resources-plugin</artifactId>  <version>2.7</version>  <configuration>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  <!-- java编译插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.2</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  <pluginManagement>  <plugins>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  </plugin>  </plugins>  </pluginManagement>  </build>  </project> |

## 2、创建common共同处理工程，用于存放整个系统的共同方法类、转换类、处理类。

工程类型；jar

需要继承parent父工程



|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.cf</groupId>  <artifactId>common</artifactId>  <version>0.0.1-SNAPSHOT</version>  <!-- jar包的依赖 -->  <dependencies>  <!-- Apache工具组件 -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-io</artifactId>  </dependency>  <dependency>  <groupId>commons-net</groupId>  <artifactId>commons-net</artifactId>  </dependency>  <!-- Jackson Json处理工具包 -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  </dependency>  <!-- httpclient -->  <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpclient</artifactId>  </dependency>  <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  <!-- 日志处理 -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  </dependency>  <!-- json -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <scope>provided</scope>  </dependency>  </dependencies>  </project> |

## 3、创建第一个业务工程，实现各个技术的解耦（也可以放在一个工程中，根据业务的复杂程度来划分）

配置中心结构：

聚合工程config（pom）

Pojo（jar）

Mapper（jar）

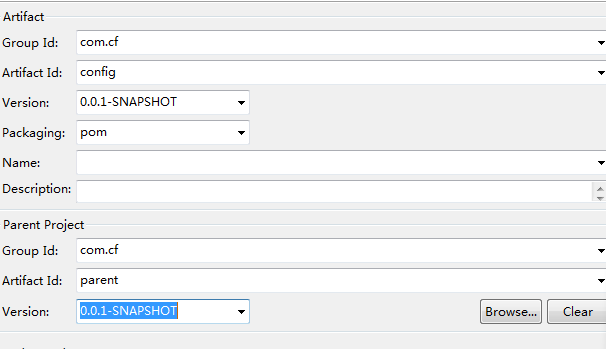
Service（jar）

Web（war）

### 3.1、创建配置中心的聚合工程config

工程类型：pom

继承parent

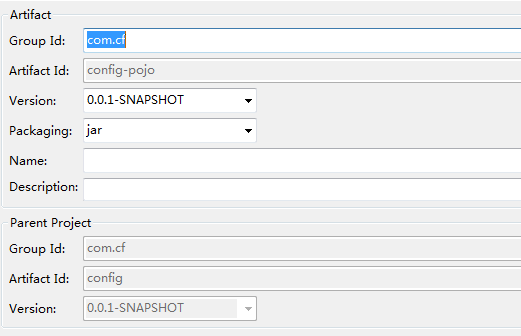


|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.cf</groupId>  <artifactId>config</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging>  <dependencies>  <!-- Apache工具组件 -->  <dependency>  <groupId>com.cf</groupId>  <artifactId>common</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies>  </project> |

### 3.2依次创建各个子工程config-pojo

创建子工程时选择模式：maven module

工程类型：jar



创建成功后，可以在config的pom中看到增加了modules的标签：

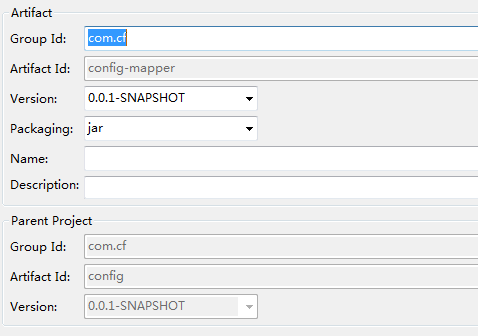
|  |
| --- |
| <modules>  <module>config-pojo</module>  </modules> |

Config-pojo 的pom定义

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>config</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>config-pojo</artifactId>  </project> |

### 3.3、创建子工程mapper

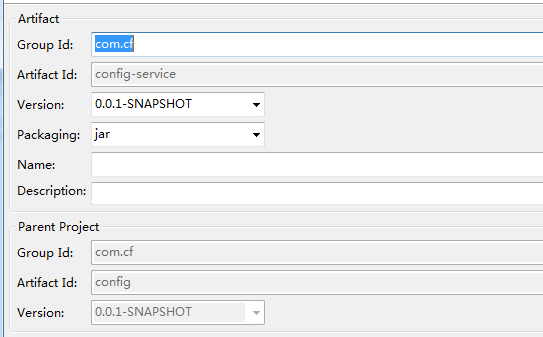
工程类型：jar



|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>config</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>config-mapper</artifactId>    <!-- 依赖管理 -->  <dependencies>  <dependency>  <groupId>com.cf</groupId>  <artifactId>config-pojo</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- Mybatis -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  </dependency>  <dependency>  <groupId>com.github.miemiedev</groupId>  <artifactId>mybatis-paginator</artifactId>  </dependency>  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  </dependency>  <!-- MySql -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <!-- 连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  </dependency>  </dependencies>  <!-- 如果不添加此节点mybatis的mapper.xml文件都会被漏掉。 -->  <build>  <resources>  <resource>  <directory>src/main/java</directory>  <includes>  <include>\*\*/\*.properties</include>  <include>\*\*/\*.xml</include>  </includes>  <filtering>false</filtering>  </resource>  </resources>  </build>  </project> |

### 3.4、创建子工程service

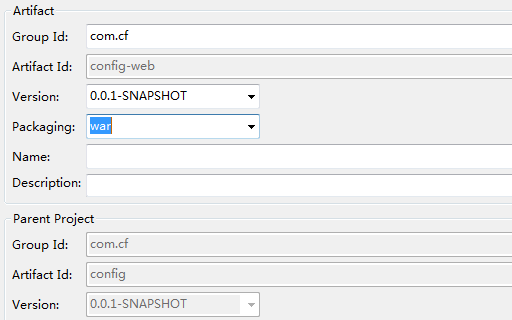
工程类型：jar



|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>config</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>config-service</artifactId>  <!-- 依赖管理 -->  <dependencies>  <dependency>  <groupId>com.cf</groupId>  <artifactId>config-mapper</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  </dependency>  </dependencies>  </project> |

### 3.5、创建子工程config-web

工程类型：war



|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>config</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>config-web</artifactId>  <packaging>war</packaging>  <!-- 依赖管理 -->  <dependencies>  <dependency>  <groupId>com.cf</groupId>  <artifactId>config-service</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- JSP相关 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <scope>provided</scope>  </dependency>  <!-- 文件上传组件 -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  </dependency>  <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8080</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

在src/main/webapp目录添加文件夹WEB-INF，并创建web.xml文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaeehttp://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"web"* version=*"2.5"*>  <display-name>config-web</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  </web-app> |

### 3.6测试maven工程

采用的是插件的模式，使用run as – buile。。-- clean tomcat7:run

在调试过程中提示错误，是因为没有吧parent、comon、config工程安装到本地仓库中。

最后创建了index.jsp

访问:ttp://localhost:8080/

# 三、分布式框架搭建-SSM工程搭建

## 1、安装mysql数据库，建表

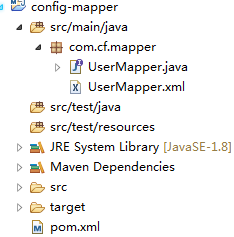
|  |
| --- |
| CREATE TABLE `user` (  `id` bigint(20) NOT NULL AUTO\_INCREMENT,  `username` varchar(50) NOT NULL COMMENT '用户名',  `password` varchar(32) NOT NULL COMMENT '密码，加密存储',  `phone` varchar(20) DEFAULT NULL COMMENT '注册手机号',  `email` varchar(50) DEFAULT NULL COMMENT '注册邮箱',  `created` datetime NOT NULL,  `updated` datetime NOT NULL,  PRIMARY KEY (`id`),  UNIQUE KEY `username` (`username`) USING BTREE,  UNIQUE KEY `phone` (`phone`) USING BTREE,  UNIQUE KEY `email` (`email`) USING BTREE  ) ENGINE=InnoDB AUTO\_INCREMENT=37 DEFAULT CHARSET=utf8 COMMENT='用户表';  INSERT INTO `user` VALUES ('1', 'zhangsan', 'e10adc3949ba59abbe56e057f20f883e', '13888888888', null, '2018-01-28 17:03:55', '2018-01-28 17:03:55'); |

## 2、创建mybatis逆向工程，自动生成代码

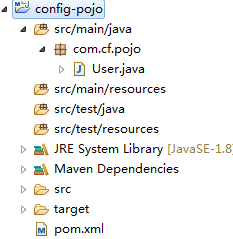
使用命令模式的mybatis-generator-core-1.3.2.jar来自动生成。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN" "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  <generatorConfiguration>  <!-- 数据库驱动包位置 -->  <classPathEntry location="E:\New\sts-bundle\workspace\CmdGenerator\mysql-connector-java-5.1.35.jar" />  <!--<classPathEntry location="xxxxx\ojdbc14.jar" /> -->  <context id="DB2Tables" targetRuntime="MyBatis3">  <commentGenerator>  <property name="suppressAllComments" value="true" />  </commentGenerator>    <!-- 数据库链接URL、用户名、密码 -->  <jdbcConnection driverClass="com.mysql.jdbc.Driver" connectionURL="jdbc:mysql://localhost:3306/test" userId="root" password="123456">  <!--<jdbcConnection driverClass="oracle.jdbc.driver.OracleDriver" connectionURL="jdbc:oracle:thin:@localhost:1521:orcl" userId="xe" password="xe">-->  </jdbcConnection>    <javaTypeResolver>  <property name="forceBigDecimals" value="false" />  </javaTypeResolver>    <!-- 生成模型的包名和位置 -->  <javaModelGenerator targetPackage="com.cf.pojo" targetProject="E:\New\sts-bundle\workspace\CmdGenerator\src">  <property name="enableSubPackages" value="true" />  <property name="trimStrings" value="true" />  </javaModelGenerator>    <!-- 生成的映射文件包名和位置 -->  <sqlMapGenerator targetPackage="com.cf.mapper" targetProject="E:\New\sts-bundle\workspace\CmdGenerator\src">  <property name="enableSubPackages" value="true" />  </sqlMapGenerator>    <!-- 生成DAO的包名和位置 -->  <javaClientGenerator type="XMLMAPPER" targetPackage="com.cf.mapper" targetProject="E:\New\sts-bundle\workspace\CmdGenerator\src">  <property name="enableSubPackages" value="true" />  </javaClientGenerator>    <!-- 要生成那些表(更改tableName和domainObjectName就可以)  <table tableName="user" domainObjectName="User" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false" />  <table tableName="user\_role" domainObjectName="UserRole" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false" />  <table tableName="role" domainObjectName="Role" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false" />  <table tableName="role\_permission" domainObjectName="RolePermission" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false" />  <table tableName="permission" domainObjectName="Permission" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false" />-->  <table tableName="user" domainObjectName="User" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false" />  </context>  </generatorConfiguration> |

## 3、mapper和xml文件添加到config-mapper工程中



## 4、pojo添加到config-pojo工程中



## 5、ssm框架的整合

整合思路：

1. dao层

使用了mybatis，创建SqlMapConifg.xml

还要创建一个applicationContext-dao.xml

1. 配置数据源
2. 需要让spring容器来管理SqlsessionFactory
3. 把mapper的代理对象放到spring容器中。使用扫描包的方式加载mapper的代理对象
4. Service层

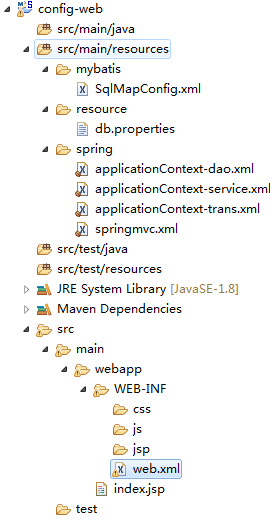
applicationContext-service.xml

applicationContext-trans.xml

1. 事务管理
2. 把service的实现类对象放到spring容器中管理
3. 表现层

springmvc.xml

1. 配置注解驱动
2. 配置视图解析器
3. 扫描controller
4. web.xml
5. spring容器的配置
6. springmvc前端控制器的配置
7. post乱码过滤器



SqlMapConfig.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>  <!-- 配置分页插件 -->  <plugins>  <plugin interceptor=*"com.github.pagehelper.PageHelper"*>  <!-- 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库-->  <property name=*"dialect"* value=*"mysql"*/>  </plugin>  </plugins>  </configuration> |

db.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://127.0.0.1:3306/test?characterEncoding=utf-8  jdbc.username=root  jdbc.password=123456 |

applicationContext-dao.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  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"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 加载配置文件 -->  <context:property-placeholder location=*"classpath:resource/\*.properties"* />  <!-- 数据库连接池 -->  <bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*  destroy-method=*"close"*>  <property name=*"url"* value=*"${jdbc.url}"* />  <property name=*"username"* value=*"${jdbc.username}"* />  <property name=*"password"* value=*"${jdbc.password}"* />  <property name=*"driverClassName"* value=*"${jdbc.driver}"* />  <property name=*"maxActive"* value=*"10"* />  <property name=*"minIdle"* value=*"5"* />  </bean>  <!-- 配置sqlsessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"*></property>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 配置扫描包，加载mapper代理对象 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.cf.mapper"*></property>  </bean>  </beans> |

applicationContext-service.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  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"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 扫描包加载Service实现类 -->  <context:component-scan base-package=*"com.cf.service"*></context:component-scan>  </beans> |

applicationContext-trans.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  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"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 事务管理器 -->  <bean id=*"transactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 数据源 -->  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!-- 通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 传播行为 -->  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"create\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 切面 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* com.cf.service.\*.\*(..))"* />  </aop:config>  </beans> |

springmvc.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:p=*"http://www.springframework.org/schema/p"*  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/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>  <context:component-scan base-package=*"com.cf.controller"* />  <mvc:annotation-driven />  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/jsp/"* />  <property name=*"suffix"* value=*".jsp"* />  </bean>  <!-- 资源映射 -->  <mvc:resources location=*"/WEB-INF/css/"* mapping=*"/css/\*\*"*/>  <mvc:resources location=*"/WEB-INF/js/"* mapping=*"/js/\*\*"*/>    <!-- 定义文件上传解析器 -->  <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!-- 设定默认编码 -->  <property name=*"defaultEncoding"* value=*"UTF-8"*></property>  <!-- 设定文件上传的最大值5MB，5\*1024\*1024 -->  <property name=*"maxUploadSize"* value=*"5242880"*></property>  </bean>  </beans> |

web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaeehttp://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"web"* version=*"2.5"*>  <display-name>config-web</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>    <!-- 加载spring容器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 解决post乱码 -->  <filter>  <filter-name>CharacterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- springmvc的前端控制器 -->  <servlet>  <servlet-name>config-web</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- contextConfigLocation不是必须的， 如果不配置contextConfigLocation， springmvc的配置文件默认在：WEB-INF/servlet的name+"-servlet.xml" -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>config-web</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

# 四、分布式框架搭建-SSM的测试

## 1、框架整合完成后，一般分为先从dao层进行测试，即从dao层提取数据到service处理；第二部才做前台的测试。

编写业务service

UserService.java

|  |
| --- |
| **package** com.cf.service;  **import** com.cf.pojo.User;  **public** **interface** UserService {  User selectByPrimaryKey(Long id);  } |

UserServiceImpl.java

|  |
| --- |
| **package** com.cf.service.impl;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.cf.mapper.UserMapper;  **import** com.cf.pojo.User;  **import** com.cf.service.UserService;  @Service  **public** **class** UserServiceImpl **implements** UserService {  @Autowired  **private** UserMapper userMapper;    @Override  **public** User selectByPrimaryKey(Long id) {  **return** userMapper.selectByPrimaryKey(id);  }  } |

在src/test/java目录创建测试类：（config-web）

TestMybatis.java

|  |
| --- |
| **package** com.cf.test;  **import** org.junit.Before;  **import** org.junit.Test;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **import** com.cf.pojo.User;  **import** com.cf.service.UserService;  **public** **class** TestMybatis {  **private** ApplicationContext ac;  **private** UserService userService;    @Before  **public** **void** before() {  ac = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");  userService = (UserService)ac.getBean("userServiceImpl");  }    @Test  **public** **void** test1() {  User user = userService.selectByPrimaryKey((**long**)1);  System.***out***.println("---------" + user.getUsername());  }  } |

测试成功代表我们的后台整合成功。

B、前台测试方法（config-web）

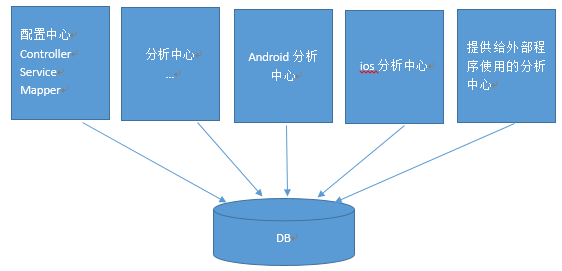
通过浏览器的方法来发送请求，并且接收到从数据库返回的查询信息。

UserController.java

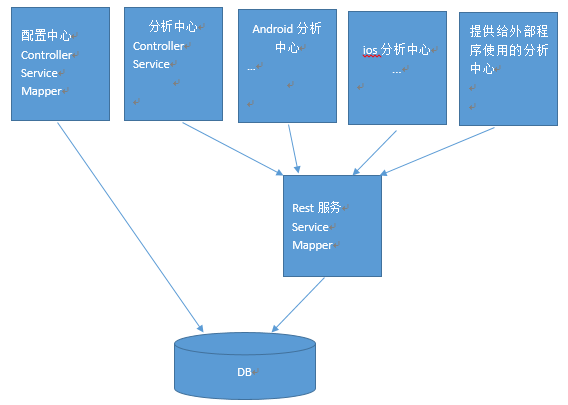
|  |
| --- |
| **package** com.cf.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** com.cf.pojo.User;  **import** com.cf.service.UserService;  @Controller  **public** **class** UserController {  @Autowired  **private** UserService userService;    @RequestMapping("/user/{id}")  @ResponseBody  **public** User getUserById(@PathVariable Long id) {    User user = userService.selectByPrimaryKey(id);  **return** user;  }  } |

# 五、Restful原理分析和服务工程搭建

传统方案，有重复代码。



优选方案，是将服务层提出，单独提供服务，其他各方访问从服务层访问。



在大中型系统开发当中，我们一般都是采用了分层的方式来架构系统，但是为什么我们需要分层进行架构呢？

采用分层架构有利于系统的维护，系统的扩展。这其实就是系统的可维护性和可扩展性。分层就是按照功能把系统切分细分，细分之后就能分布式部署，就能引入伸缩性，就能提高性能。

优点：

1、基于soa（面向服务的架构）理念将服务层抽出对外提供服务

2、可以实现灵活的分布式部署

3、前、后端开发可以分离

缺点：

服务之间使用接口通信，增加了开发工作量。

总结；将前台系统分为两个部分，一部分是服务层web工程，功能就是发布服务

另一部分是表现层，展示页面，没有业务逻辑。

搭建两套框架，restful-web、portal-web

## 1、首先搭建服务层框架

使用的技术：

Mybatis

Spring

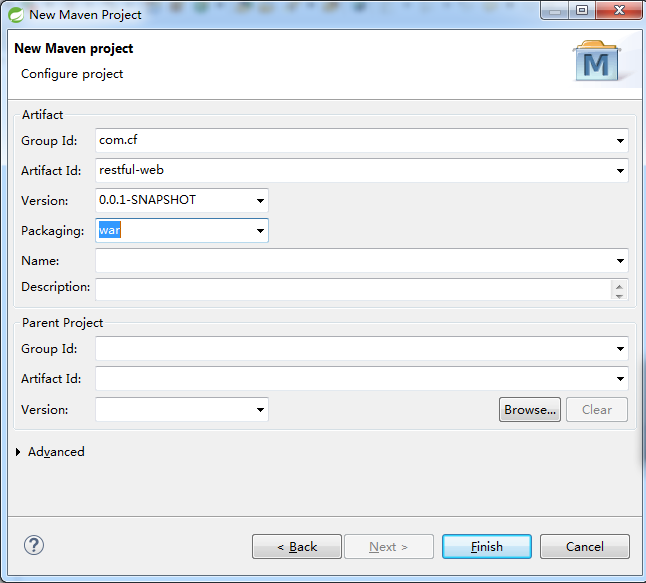
Springmvc

采用json数据传输

### A、restful-web

工程类型：war

继承parent、config-mapper



从config-web工程复制过来，mybatis不用改

Resource目录不用改

applicationContext-dao.xml不用改

applicationContext-service.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  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"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 扫描包加载Service实现类 -->  <context:component-scan base-package=*"com.cf.rest.service"*></context:component-scan>  </beans> |

applicationContext-trans.xml

需要package：com.cf.rest.service.\*.\*(..))

Springmvc.xml修改扫描包路径，删除多余的视图解析器、静态资源。

|  |
| --- |
| <context:component-scan base-package=*"com.cf.rest.controller"* />  <mvc:annotation-driven /> |

Pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>config</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.cf</groupId>  <artifactId>restful-web</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <dependencies>  <!-- 依赖config-mapper工程 -->  <dependency>  <groupId>com.cf</groupId>  <artifactId>config-mapper</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  </dependency>  <!-- httprequest -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <scope>provided</scope>  </dependency>  </dependencies>    <build>  <plugins>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8082</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

Web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaeehttp://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"web"* version=*"2.5"*>  <display-name>restful-web</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>    <!-- 加载spring容器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 解决post乱码 -->  <filter>  <filter-name>CharacterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- springmvc的前端控制器 -->  <servlet>  <servlet-name>restful-web</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- contextConfigLocation不是必须的， 如果不配置contextConfigLocation， springmvc的配置文件默认在：WEB-INF/servlet的name+"-servlet.xml" -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>restful-web</servlet-name>  <url-pattern>/rest/\*</url-pattern>  </servlet-mapping>  </web-app> |

### B、服务调用工程搭建portal-web

工程类型：war

需要继承parent

使用的技术：

Spring

Springmvc

Jsp、js

Httpclient

Pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.cf</groupId>  <artifactId>portal-web</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>    <dependencies>  <dependency>  <groupId>com.cf</groupId>  <artifactId>common</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  </dependency>  <!-- JSP相关 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <scope>provided</scope>  </dependency>  <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8083</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

Web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaeehttp://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"web"* version=*"2.5"*>  <display-name>portal-web</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>    <!-- 加载spring容器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 解决post乱码 -->  <filter>  <filter-name>CharacterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- springmvc的前端控制器 -->  <servlet>  <servlet-name>portal-web</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- contextConfigLocation不是必须的， 如果不配置contextConfigLocation， springmvc的配置文件默认在：WEB-INF/servlet的name+"-servlet.xml" -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>portal-web</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

applicationContext-service.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  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"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 加载配置文件 -->  <context:property-placeholder location=*"classpath:resource/\*.properties"* />    <!-- 扫描包加载Service实现类 -->  <context:component-scan base-package=*"com.cf.portal.service"*></context:component-scan>  </beans> |

springmvc.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:p=*"http://www.springframework.org/schema/p"*  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/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>  <context:component-scan base-package=*"com.cf.portal.controller"* />  <mvc:annotation-driven />    <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/jsp/"* />  <property name=*"suffix"* value=*".jsp"* />  </bean>  <!-- 资源映射 -->  <mvc:resources location=*"/WEB-INF/css/"* mapping=*"/css/\*\*"*/>  <mvc:resources location=*"/WEB-INF/js/"* mapping=*"/js/\*\*"*/>    </beans> |

页面测试类

|  |
| --- |
| package com.cf.portal.controller;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  @Controller  public class IndexController {  @RequestMapping("/index")  public String shouIndex() {  return "index";  }  } |

在WEB—INF下，创建对于的jsp/index.jsp。

# 六、Restful服务发布

需要特别说明的是Restful Web API应该严格遵守Http的四个操作：

GET对应查询操作

POST对应创建操作

PUT对应更新操作

DELETE对应删除操作

不应该再有其它任何操作，也就是说从HttpClient发过来的Request类型只能是上述四种之一。否则不能称为Restful API。

发布一个服务：

1. 需求分析：使用user表，构造增删改查的服务，并将结果以json的格式返回。

在restful-web工程中创建相应的controller、service目录结构和文件。

RestService.java

|  |
| --- |
| **package** com.cf.rest.service;  **import** com.cf.pojo.User;  **public** **interface** RestService {  User selectByPrimaryKey(Long id);  } |

RestServiceImpl.java

|  |
| --- |
| **package** com.cf.rest.service.impl;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.cf.mapper.UserMapper;  **import** com.cf.pojo.User;  **import** com.cf.rest.service.RestService;  @Service  **public** **class** RestServiceImpl **implements** RestService {  @Autowired  **private** UserMapper userMapper;    @Override  **public** User selectByPrimaryKey(Long id) {  **return** userMapper.selectByPrimaryKey(id);  }  } |

RestController.java

|  |
| --- |
| **package** com.cf.rest.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.bind.annotation.RequestParam;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** com.alibaba.fastjson.JSON;  **import** com.cf.pojo.User;  **import** com.cf.rest.service.RestService;  @Controller  **public** **class** RestController {  @Autowired  **private** RestService restService;    /\*\*  \* 查询操作 REST风格 : /user/用户id  \*  \*/  @RequestMapping(value = "/user/{id}", method = RequestMethod.***GET***)  @ResponseBody  **public** String get(@PathVariable("id") Long id) {  User user = restService.selectByPrimaryKey(id);  String result = JSON.*toJSONString*(user);  System.***out***.println("---------------------------get:" + result);    **return** result;  }    /\*\*  \* 创建操作 REST风格  \*  \*/  @RequestMapping(value = "/user/register", method = RequestMethod.***POST***)  @ResponseBody  **public** String register(User user) {  String result = JSON.*toJSONString*(user);  System.***out***.println("---------------------------Post:" + result);    **return** result;  }    /\*\*  \* 删除操作 REST风格  \*  \*/  @RequestMapping(value = "/user/{id}", method = RequestMethod.***DELETE***)  @ResponseBody  **public** String delete(@PathVariable("id") Long id) {  System.***out***.println("---------------------------delete:" + id);    **return** "delete";  }    /\*\*  \* 更新操作 REST风格  \*  \*/  @RequestMapping(value = "/user/update", method = RequestMethod.***PUT***)  @ResponseBody  **public** String update(User user) {  String result = JSON.*toJSONString*(user);  System.***out***.println("---------------------------Put:" + result);    **return** result;  }    /\*\*  \* 传统风格 : /user?name户zhangsan&age=18  \*  \*/  @RequestMapping(value = "/user", method = RequestMethod.***GET***)  @ResponseBody  **public** String search(@RequestParam("name") String queryString) {    System.***out***.println("---------------------------get:" + queryString);    **return** queryString;  }  } |

# 七、Restful服务测试

对服务进行测试验证，有很多种方法，可以使用浏览器插件（FF HttpRequester、RESTClient）、java 小程序（rest-client）、linux的命令行工具curl等或者是自己写代码去调用访问。

如果我们是只发布服务，那么使用这些工具测试即可，但我们是要构造一套完整的系统，所以只有自己来写代码了。

需要在portal-web工程中编写测试类：

HttpClientTest.java

|  |
| --- |
| **package** com.cf.test;  **import** java.io.IOException;  **import** java.net.URISyntaxException;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.apache.http.HttpEntity;  **import** org.apache.http.NameValuePair;  **import** org.apache.http.client.ClientProtocolException;  **import** org.apache.http.client.entity.UrlEncodedFormEntity;  **import** org.apache.http.client.methods.CloseableHttpResponse;  **import** org.apache.http.client.methods.HttpDelete;  **import** org.apache.http.client.methods.HttpGet;  **import** org.apache.http.client.methods.HttpPost;  **import** org.apache.http.client.methods.HttpPut;  **import** org.apache.http.client.utils.URIBuilder;  **import** org.apache.http.entity.StringEntity;  **import** org.apache.http.impl.client.CloseableHttpClient;  **import** org.apache.http.impl.client.HttpClients;  **import** org.apache.http.message.BasicNameValuePair;  **import** org.apache.http.util.EntityUtils;  **import** org.junit.Test;  **public** **class** HttpClientTest {  @Test  **public** **void** doGet() **throws** ClientProtocolException, IOException {  // 创建一个httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();    // 创建一个get对象  HttpGet get = **new** HttpGet("http://localhost:8082/rest/user/2");    // 执行请求  CloseableHttpResponse reponse = httpClient.execute(get);    // 取得响应结果  HttpEntity entity = reponse.getEntity();  String result = EntityUtils.*toString*(entity, "UTF-8");    // 打印结果  System.***out***.println("---------------------------------" + result);    // 关闭链接  reponse.close();  httpClient.close();  }    @Test  **public** **void** doGetWithParam() **throws** ClientProtocolException, IOException, URISyntaxException {  // 创建一个httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();    // 创建一个uri对象  URIBuilder uriBuilder = **new** URIBuilder("http://localhost:8082/rest/user");  uriBuilder.addParameter("name", "123456");  HttpGet get = **new** HttpGet(uriBuilder.build());    // 执行请求  CloseableHttpResponse reponse = httpClient.execute(get);    // 取得响应结果  HttpEntity entity = reponse.getEntity();  String result = EntityUtils.*toString*(entity, "UTF-8");    // 打印结果  System.***out***.println("---------------------------------" + result);    // 关闭链接  reponse.close();  httpClient.close();  }    @Test  **public** **void** doPost() **throws** Exception {  // 创建一个httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();    // 创建一个get对象  HttpPost post = **new** HttpPost("http://localhost:8082/rest/user/register");    // 执行请求  CloseableHttpResponse reponse = httpClient.execute(post);    // 取得响应结果  HttpEntity entity = reponse.getEntity();  String result = EntityUtils.*toString*(entity, "UTF-8");    // 打印结果  System.***out***.println("---------------------------------" + result);    // 关闭链接  reponse.close();  httpClient.close();  }    @Test  **public** **void** doPostWithParam() **throws** Exception {  // 创建一个httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();    // 创建一个get对象  HttpPost post = **new** HttpPost("http://localhost:8082/rest/user/register");    // 创建entity，模拟提交一个表单内容  List<NameValuePair> paraList = **new** ArrayList<>();  paraList.add(**new** BasicNameValuePair("id", "3"));  paraList.add(**new** BasicNameValuePair("username", "王五"));  paraList.add(**new** BasicNameValuePair("password", "123"));    StringEntity entity = **new** UrlEncodedFormEntity(paraList, "UTF-8");  post.setEntity(entity);    // 执行请求  CloseableHttpResponse reponse = httpClient.execute(post);    // 取得响应结果  String result = EntityUtils.*toString*(reponse.getEntity(), "UTF-8");    // 打印结果  System.***out***.println("---------------------------------" + result);    // 关闭链接  reponse.close();  httpClient.close();  }    @Test  **public** **void** doPutWithParam() **throws** Exception {  // 创建一个httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();    // 创建一个get对象  HttpPut put = **new** HttpPut("http://localhost:8082/rest/user/update");    // 创建entity，模拟提交一个表单内容  List<NameValuePair> paraList = **new** ArrayList<>();  paraList.add(**new** BasicNameValuePair("id", "3"));  paraList.add(**new** BasicNameValuePair("username", "test2"));  paraList.add(**new** BasicNameValuePair("password", "123"));    StringEntity entity = **new** UrlEncodedFormEntity(paraList, "UTF-8");  put.setEntity(entity);    // 执行请求  CloseableHttpResponse reponse = httpClient.execute(put);    // 取得响应结果  String result = EntityUtils.*toString*(reponse.getEntity(), "UTF-8");    // 打印结果  System.***out***.println("---------------------------------" + result);    // 关闭链接  reponse.close();  httpClient.close();  }    @Test  **public** **void** doDelete() **throws** ClientProtocolException, IOException {  // 创建一个httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();    // 创建一个get对象  HttpDelete delete = **new** HttpDelete("http://localhost:8082/rest/user/1");    // 执行请求  CloseableHttpResponse reponse = httpClient.execute(delete);    // 取得响应结果  HttpEntity entity = reponse.getEntity();  String result = EntityUtils.*toString*(entity, "UTF-8");    // 打印结果  System.***out***.println("---------------------------------" + result);    // 关闭链接  reponse.close();  httpClient.close();  }  } |

测试中解决的问题：

1. 访问服务后返回值中有中文乱码

在RequestMapping中添加produces属性。

@RequestMapping(value = "/user/register", method = RequestMethod.POST, produces = MediaType.TEXT\_PLAIN\_VALUE + ";charset=utf-8")

如果返回是json的场合，***APPLICATION\_JSON\_VALUE***

1. put类型请求无法获取参数

在web.xml中添加对于过滤器

|  |
| --- |
| <filter>  <filter-name>HttpMethodFilter</filter-name>  <filter-class>org.springframework.web.filter.HttpPutFormContentFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>HttpMethodFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

注意：

需要注意的是，该过滤器只能接受enctype值为application/x-www-form-urlencoded的表单，也就是说，在使用该过滤器时，form表单的代码必须如下：

<form action="" method="put" enctype="application/x-www-form-urlencoded">

......

</form>

# 八、使用HttpClient实现系统之间服务调用

1、将httpclient的测试类进行共同处理，方便后续其他工程调用，并入common工程中。

|  |
| --- |
| **package** com.cf.utils;  **import** java.io.IOException;  **import** java.net.URI;  **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.Map;  **import** org.apache.http.NameValuePair;  **import** org.apache.http.client.entity.UrlEncodedFormEntity;  **import** org.apache.http.client.methods.CloseableHttpResponse;  **import** org.apache.http.client.methods.HttpDelete;  **import** org.apache.http.client.methods.HttpGet;  **import** org.apache.http.client.methods.HttpPost;  **import** org.apache.http.client.methods.HttpPut;  **import** org.apache.http.client.utils.URIBuilder;  **import** org.apache.http.entity.ContentType;  **import** org.apache.http.entity.StringEntity;  **import** org.apache.http.impl.client.CloseableHttpClient;  **import** org.apache.http.impl.client.HttpClients;  **import** org.apache.http.message.BasicNameValuePair;  **import** org.apache.http.util.EntityUtils;  **public** **class** HttpClientUtil {  **public** **static** String doGet(String url, Map<String, String> param) {  // 创建Httpclient对象  CloseableHttpClient httpclient = HttpClients.*createDefault*();  String resultString = "";  CloseableHttpResponse response = **null**;  **try** {  // 创建uri  URIBuilder builder = **new** URIBuilder(url);  **if** (param != **null**) {  **for** (String key : param.keySet()) {  builder.addParameter(key, param.get(key));  }  }  URI uri = builder.build();  // 创建http GET请求  HttpGet httpGet = **new** HttpGet(uri);  // 执行请求  response = httpclient.execute(httpGet);  // 判断返回状态是否为200  **if** (response.getStatusLine().getStatusCode() == 200) {  resultString = EntityUtils.*toString*(response.getEntity(), "UTF-8");  }  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (response != **null**) {  response.close();  }  httpclient.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  **return** resultString;  }  **public** **static** String doGet(String url) {  **return** *doGet*(url, **null**);  }  **public** **static** String doPostWithMap(String url, Map<String, String> param) {  // 创建Httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();  CloseableHttpResponse response = **null**;  String resultString = "";  **try** {  // 创建Http Post请求  HttpPost httpPost = **new** HttpPost(url);  // 创建参数列表  **if** (param != **null**) {  List<NameValuePair> paramList = **new** ArrayList<>();  **for** (String key : param.keySet()) {  paramList.add(**new** BasicNameValuePair(key, param.get(key)));  }  // 模拟表单  UrlEncodedFormEntity entity = **new** UrlEncodedFormEntity(paramList);  httpPost.setEntity(entity);  }  // 执行http请求  response = httpClient.execute(httpPost);  resultString = EntityUtils.*toString*(response.getEntity(), "utf-8");  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  **try** {  response.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  **return** resultString;  }  **public** **static** String doPost(String url) {  **return** *doPostWithMap*(url, **null**);  }    **public** **static** String doPostJson(String url, String json) {  // 创建Httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();  CloseableHttpResponse response = **null**;  String resultString = "";  **try** {  // 创建Http Post请求  HttpPost httpPost = **new** HttpPost(url);  // 创建请求内容  StringEntity entity = **new** StringEntity(json, ContentType.***APPLICATION\_JSON***);  httpPost.setEntity(entity);  // 执行http请求  response = httpClient.execute(httpPost);  resultString = EntityUtils.*toString*(response.getEntity(), "utf-8");  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  **try** {  response.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  **return** resultString;  }    **public** **static** String doPut(String url, Map<String, String> param) {  // 创建Httpclient对象  CloseableHttpClient httpclient = HttpClients.*createDefault*();  String resultString = "";  CloseableHttpResponse response = **null**;  **try** {  // 创建uri  URIBuilder builder = **new** URIBuilder(url);  **if** (param != **null**) {  **for** (String key : param.keySet()) {  builder.addParameter(key, param.get(key));  }  }  URI uri = builder.build();  // 创建http GET请求  HttpPut httpPut = **new** HttpPut(uri);  // 执行请求  response = httpclient.execute(httpPut);  // 判断返回状态是否为200  **if** (response.getStatusLine().getStatusCode() == 200) {  resultString = EntityUtils.*toString*(response.getEntity(), "UTF-8");  }  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (response != **null**) {  response.close();  }  httpclient.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  **return** resultString;  }    **public** **static** String doDelete(String url) {  // 创建Httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();  CloseableHttpResponse response = **null**;  String resultString = "";  **try** {  // 创建Http Post请求  HttpDelete httpDelete = **new** HttpDelete(url);  // 执行http请求  response = httpClient.execute(httpDelete);  resultString = EntityUtils.*toString*(response.getEntity(), "utf-8");  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  **try** {  response.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  **return** resultString;  }  } |

1. 实现一个客户端系统访问restful系统服务；（调用服务的地方放到portal-web工程中来进行）

IndexServiceImpl.java

|  |
| --- |
| **package** com.cf.portal.service.impl;  **import** org.springframework.stereotype.Service;  **import** com.cf.portal.service.IndexService;  **import** com.cf.utils.HttpClientUtil;  @Service  **public** **class** IndexServiceImpl **implements** IndexService {  @Override  **public** String get(Long id) {  //  String json = HttpClientUtil.*doGet*("http://localhost:8082/rest/user/" + id);    **return** json;  }  } |

IndexService.java

|  |
| --- |
| **package** com.cf.portal.service;  **public** **interface** IndexService {  **public** String get(Long id);  } |

IndexController.java

|  |
| --- |
| package com.cf.portal.controller;  import org.springframework.beans.factory.annotation.Autowired;  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.cf.portal.service.IndexService;  @Controller  public class IndexController {  @Autowired  private IndexService indexService;    @RequestMapping("/index")  public String shouIndex() {  return "index";  }    @RequestMapping("/get")  @ResponseBody  public String get(@RequestParam("id") Long id) {  String result = indexService.get(id);  System.out.println("----------------portal---------------" + result);  return result;  }  } |

先启动，启动服务工程，restful-web

然后启动，客户端工程，portal-web，并且在浏览器访问url <http://localhost:8083/get?id=1>。

# 九、业务功能缓存的实现-redis单机版安装

## 1、什么是redis

Redis是用C语言开发的一个开源的高性能键值对（key-value）数据库。它通过提供多种键值数据类型来适应不同场景下的存储需求，Redis支持的键值数据类型主要如下：

字符串类型

散列类型

列表类型

集合类型

有序集合类型。

## 2、redis的应用场景

缓存。

分布式集群架构中的session分离。

聊天室的在线好友列表。

任务队列。（秒杀、抢购、12306等等）

应用排行榜。

网站访问统计。

数据过期处理（可以精确到毫秒）

## 3、Redis的安装

redis是C语言开发，建议在linux上运行，本教程使用Centos6.7作为安装环境。

安装redis需要先将官网下载的源码进行编译，编译依赖gcc环境，如果没有gcc环境，则需要手动安装。

### 3.1、检测是否安装gcc

gcc -v

rpm -qa|grep gcc

### 3.2、联网安装gcc、g++

yum install gcc-c++

### 3.3、安装redis

    版本说明：

    本教程使用redis3.2.11版本。3.0版本主要增加了redis集群功能。

    a、源码下载：

<http://download.redis.io/releases/>

    将redis-3.2.11.tar.gz拷贝到虚机的/usr/local目录下

   b、 解压源码

    # tar -zxvf redis-3.2.11.tar.gz

    c、进入解压后的目录进行编译

    # cd redis-3.2.11

    # make

    d、安装到指定目录,如 /usr/local/redis

    # cd /usr/local/redis-3.2.11

    # make PREFIX=/usr/local/redis install

    e、启动：分为前端启动和后台进程启动

    前端启动-->

    # cd /usr/local/redis/bin

    # ./redis-server

    f、后台启动-->到解压安装包的目录找到redis.conf文件（/usr/local/redis-3.2.11 ）

    redis.conf是redis的配置文件，redis.conf在redis源码目录。

    注意修改port作为redis进程的端口,port默认6379。

    # cd /usr/local/redis-3.2.11

    # cp redis.conf /usr/local/redis/bin/

    # cd /usr/local/redis/bin/

    # vi redis.conf

    修改并保存daemonize no----->daemonize yes

    g、重新启动的时候，需要指定配置启动，查看进程是否启动成功。

    # ./redis-server redis.conf

    # ps -aux | grep redis

    h、启动客户端或工具查看是否情况成功。

    # ./redis-cli

    i、修改防火墙策略，允许外部访问。

    # vi /etc/sysconfig/iptables

    增加：

    -A INPUT -m state --state NEW -m tcp -p tcp --dport 6379 -j ACCEPT

    保存退出。

    重启防火墙

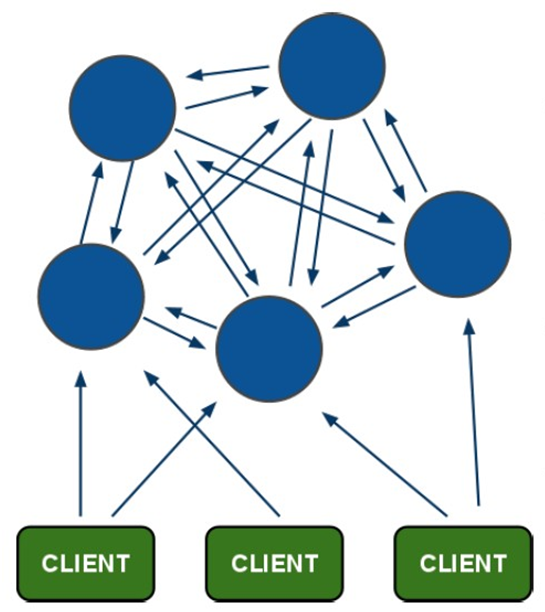
    # service iptables restart

    查看防火墙状态：

    # service iptables status

# 十、业务功能缓存的实现-redis集群环境搭建

## 1、集群架构图：



架构细节:

(1)所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽.

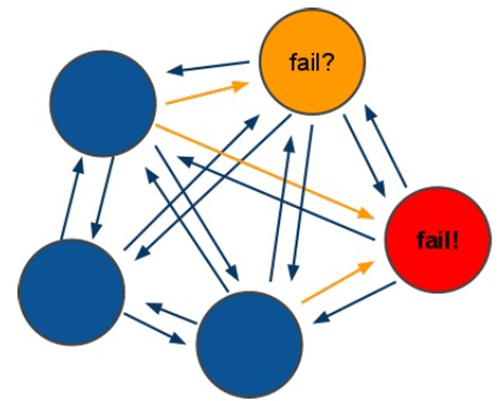
(2)节点的fail是通过集群中超过半数的节点检测失效时才生效.

(3)客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可

(4)redis-cluster把所有的物理节点映射到[0-16383]slot上,cluster 负责维护node<->slot<->value

Redis 集群中内置了 16384 个哈希槽，当需要在 Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点

## 2、redis-cluster投票:容错



(1)领着投票过程是集群中所有master参与,如果半数以上master节点与master节点通信超过(cluster-node-timeout),认为当前master节点挂掉.

(2):什么时候整个集群不可用(cluster\_state:fail)?

a:如果集群任意master挂掉,且当前master没有slave.集群进入fail状态,也可以理解成集群的slot映射[0-16383]不完成时进入fail状态. ps : redis-3.0.0.rc1加入cluster-require-full-coverage参数,默认关闭,打开集群兼容部分失败.

b:如果集群超过半数以上master挂掉，无论是否有slave集群进入fail状态.

ps:当集群不可用时,所有对集群的操作做都不可用，收到((error) CLUSTERDOWN The cluster is down)错误

## 3、ruby环境准备

redis集群管理工具redis-trib.rb依赖ruby环境，首先需要安装ruby环境：

#..redis/src/redis-trib.rb

安装ruby

yum install ruby

yum install rubygems

安装ruby和redis的接口程序(https://rubygems.org/gems/redis/)

拷贝redis-3.0.0.gem至/usr/local下

执行：

gem install /usr/local/redis-3.0.0.gem

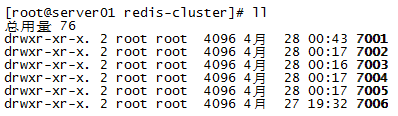
## 4、集群结点规划

这里在同一台服务器用不同的端口表示不同的redis服务器，如下：

主节点：192.168.101.3:7001 192.168.101.3:7002 192.168.101.3:7003

从节点：192.168.101.3:7004 192.168.101.3:7005 192.168.101.3:7006

在/usr/local下创建redis-cluster目录，其下创建7001、7002。。7006目录，如下：



将redis安装目录bin下的文件拷贝到每个700X目录内，同时将redis源码目录src下的redis-trib.rb拷贝到redis-cluster目录下。

修改每个700X目录下的redis.conf配置文件：

port 7001

#bind 192.168.101.3

cluster-enabled yes

在redis解压目录中，将redis-trib.rb复杂到redis-cluster目录

# cd /home/install/redis-3.0.7/src

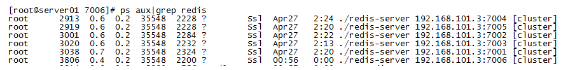
# cp \*.rb /usr/local/redis-cluster/

## 5、启动每个结点redis服务

分别进入7001、7002、...7006目录，执行：

./redis-server ./redis.conf

查看redis进程：



6、创建集群

#./redis-trib.rb create --replicas 1 192.168.31.177:7001 192.168.31.177:7002 192.168.31.177:7003 192.168.31.177:7004 192.168.31.177:7005 192.168.31.177:7006

7、集群环境验证

在任意一个节点下，使用redis-cli

# cd 7001

# ./redis-cli -h 192.168.31.177 -c -p 7001 (-h 地址 -p 端口 -c 代表连接集群)

> cluster nodes列出集群当前已知的所有节点（ node），以及这些节点的相关信息。

注意：在每个节点的根目录会有一个dump.rdb文件，是快照模式，就是把内存中的数据保存到.rdb中。

Redis的所有数据都是保存在内存中，然后不定期的通过异步方式保存到磁盘上(这称为“半持久化模式”(rdb))；

也可以把每一次数据变化都写入到一个append only file(aof)里面(这称为“全持久化模式”)。

集群重启：依次启动各个节点即可，redis会自动集群连接。

单机版关闭：>shutdown

集群中关闭：# redis-cli -p 7001 shutdown ....

# 十一、redis单机和集群环境测试

首先来了解一下redis有哪些客户端

Redis-cli

Redisdesktopmanager

Jedis(java

## 1、jedis单机版

在restful-web工程中添加jar包依赖

|  |
| --- |
| <!-- Redis客户端 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  </dependency>    <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency> |

测试类：

|  |
| --- |
| **package** com.cf.jedis.test;  **import** java.util.HashSet;  **import** org.junit.Test;  **import** redis.clients.jedis.HostAndPort;  **import** redis.clients.jedis.Jedis;  **import** redis.clients.jedis.JedisCluster;  **import** redis.clients.jedis.JedisPool;  **public** **class** JedisTest {  /\*\*  \* 单机版测试  \*  \*/  @Test  **public** **void** testJedisSingle() {  // 创建一个jedis对象  Jedis jedis = **new** Jedis("192.168.241.128", 6379);  // 调用jedis对象的方法，方法名称和redis命令一致  jedis.set("key1", "jedis test");  String result = jedis.get("key1");  System.***out***.println(result);  // 关闭jedis  jedis.close();  }    /\*\*  \* 单机版测试  \*  \*/  @Test  **public** **void** testJedisPool() {  JedisPool pool = **new** JedisPool("192.168.241.128", 6379);  // 创建一个jedis对象  Jedis jedis = pool.getResource();  // 调用jedis对象的方法，方法名称和redis命令一致  jedis.set("key2", "jedis test");  String result = jedis.get("key2");  System.***out***.println(result);  // 关闭jedis  jedis.close();  pool.close();  }  /\*\*  \* 集群测试  \*/  @Test  **public** **void** testJedisCluster() {  HashSet<HostAndPort> nodes = **new** HashSet<>();  nodes.add(**new** HostAndPort("192.168.241.128", 7001));  nodes.add(**new** HostAndPort("192.168.241.128", 7002));  nodes.add(**new** HostAndPort("192.168.241.128", 7003));  nodes.add(**new** HostAndPort("192.168.241.128", 7004));  nodes.add(**new** HostAndPort("192.168.241.128", 7005));  nodes.add(**new** HostAndPort("192.168.241.128", 7006));    JedisCluster cluster = **new** JedisCluster(nodes);  cluster.set("clusterKey", "1000");  String result = cluster.get("clusterKey");  System.***out***.println(result);  cluster.close();  }  } |

单机版：

a、使用jedis直接调用redis单机版的方式

b、使用jedis连接池调用redis单机版的方式，调用完成后需要关闭进行资源释放。

集群版：

c、使用jedis直接调用redis集群版本的方式。cluster不需要连接池。

# 十二、spring和redis的集成

使用spring来管理jedis，包含配置，单机版和集群切换。

新建applicationContext-jedis.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  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"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 连接池配置 -->  <bean id=*"jedisPoolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>  <!-- 最大连接数 -->  <property name=*"maxTotal"* value=*"30"* />  <!-- 最大空闲连接数 -->  <property name=*"maxIdle"* value=*"10"* />  <!-- 每次释放连接的最大数目 -->  <property name=*"numTestsPerEvictionRun"* value=*"1024"* />  <!-- 释放连接的扫描间隔（毫秒） -->  <property name=*"timeBetweenEvictionRunsMillis"* value=*"30000"* />  <!-- 连接最小空闲时间 -->  <property name=*"minEvictableIdleTimeMillis"* value=*"1800000"* />  <!-- 连接空闲多久后释放, 当空闲时间>该值且空闲连接>最大空闲连接数 时直接释放 -->  <property name=*"softMinEvictableIdleTimeMillis"* value=*"10000"* />  <!-- 获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1 -->  <property name=*"maxWaitMillis"* value=*"1500"* />  <!-- 在获取连接的时候检查有效性, 默认false -->  <property name=*"testOnBorrow"* value=*"true"* />  <!-- 在空闲时检查有效性, 默认false -->  <property name=*"testWhileIdle"* value=*"true"* />  <!-- 连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true -->  <property name=*"blockWhenExhausted"* value=*"false"* />  </bean>  <!-- 单机版配置  <bean id="redisClient" class="redis.clients.jedis.JedisPool">  <constructor-arg name="host" value="192.168.241.128"></constructor-arg>  <constructor-arg name="port" value="6379"></constructor-arg>  <constructor-arg name="poolConfig" ref="jedisPoolConfig"></constructor-arg>  </bean>  -->  <!-- 集群版配置 -->  <bean id=*"redisClient"* class=*"redis.clients.jedis.JedisCluster"*>  <constructor-arg name=*"nodes"*>  <set>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.241.128"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7001"*></constructor-arg>  </bean>    <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.241.128"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7002"*></constructor-arg>  </bean>    <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.241.128"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7003"*></constructor-arg>  </bean>    <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.241.128"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7004"*></constructor-arg>  </bean>    <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.241.128"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7005"*></constructor-arg>  </bean>    <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.241.128"*></constructor-arg>  <constructor-arg name=*"port"* value=*"7006"*></constructor-arg>  </bean>  </set>  </constructor-arg>  <constructor-arg name=*"poolConfig"* ref=*"jedisPoolConfig"*></constructor-arg>  </bean>  </beans> |

测试类：

|  |
| --- |
| /\*\*  \* spring与redis单机版测试  \*/  @Test  **public** **void** testSpringJedisSingle() {  ApplicationContext app = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");    JedisPool pool = (JedisPool) app.getBean("redisClient");    // 创建一个jedis对象  Jedis jedis = pool.getResource();  // 调用jedis对象的方法，方法名称和redis命令一致  String result = jedis.get("key2");    System.***out***.println(result);  // 关闭jedis  jedis.close();  pool.close();  }    /\*\*  \* spring与redis集群版测试  \*  \*/  @Test  **public** **void** testSpringJedisCluster() {  ApplicationContext app = **new** ClassPathXmlApplicationContext("classpath:spring/applicationContext-\*.xml");    JedisCluster cluster = (JedisCluster)app.getBean("redisClient");    String result = cluster.get("clusterKey");  System.***out***.println(result);  cluster.close();  } |

# 十三、使用redis实现业务缓存

和框架集成完成之后，现在要用redis来实现我们的业务缓存。一般情况下是直接在数据库中进行查询，这种场景的作用就是为了减轻数据的压力，提高响应的速度。

我们之前集成了两种模式的redis，单机版和集群版；

单机版和集群版的使用是有区别的，

单机版我们是获取一个jedispool，它是一个连接池，通过连接池来getResource()取得jedis对象，然后才是通过jedis这个对象进行redis的操作。

集群版则是直接得到jedisCluster对象，它是可以直接操作redis的。

为了适配这两种模式的redis，所以我们可以在dao层进行处理，专门来编写dao，我们从数据库获取数据时也是在dao层进行的处理。

需要定义一个接口，将常用的操作写成对于的方法：

Redis命令：

* set key1 1
* get key1
* hset hash1 key1 1
* hget hash1 key
* incr a
* expire a 1000
* ttl a
* ttl hash1

接口类

|  |
| --- |
| **package** com.cf.rest.dao;  **public** **interface** JedisClient {  String get(String key);  String set(String key, String value);  String hget(String hkey, String key);  **long** hset(String hkey, String key, String value);  **long** incr(String key);  **long** expire(String key, **int** second);  **long** ttl(String key);  **long** del(String key);  **long** hdel(String hkey, String key);  } |

单机版实现类

|  |
| --- |
| **package** com.cf.rest.dao.impl;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** com.cf.rest.dao.JedisClient;  **import** redis.clients.jedis.Jedis;  **import** redis.clients.jedis.JedisPool;  **public** **class** JedisClientSingle **implements** JedisClient {  @Autowired  **private** JedisPool jedisPool;    @Override  **public** String get(String key) {  Jedis jedis = jedisPool.getResource();  String result = jedis.get(key);  jedis.close();  **return** result;  }  @Override  **public** String set(String key, String value) {  Jedis jedis = jedisPool.getResource();  String result = jedis.set(key, value);  jedis.close();  **return** result;  }  @Override  **public** String hget(String hkey, String key) {  Jedis jedis = jedisPool.getResource();  String result = jedis.hget(hkey, key);  jedis.close();  **return** result;  }  @Override  **public** **long** hset(String hkey, String key, String value) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.hset(hkey, key, value);  jedis.close();  **return** result;  }  @Override  **public** **long** incr(String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.incr(key);  jedis.close();  **return** result;  }  @Override  **public** **long** expire(String key, **int** second) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.expire(key, second);  jedis.close();  **return** result;  }  @Override  **public** **long** ttl(String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.ttl(key);  jedis.close();  **return** result;  }  @Override  **public** **long** del(String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.del(key);  jedis.close();    **return** result;  }  @Override  **public** **long** hdel(String hkey, String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.hdel(key);  jedis.close();  **return** result;  }  } |

集群版实现类

|  |
| --- |
| **package** com.cf.rest.dao.impl;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** com.cf.rest.dao.JedisClient;  **import** redis.clients.jedis.JedisCluster;  **public** **class** JedisClientCluster **implements** JedisClient {  @Autowired  **private** JedisCluster jedisCluster;    @Override  **public** String get(String key) {  **return** jedisCluster.get(key);  }  @Override  **public** String set(String key, String value) {  **return** jedisCluster.set(key, value);  }  @Override  **public** String hget(String hkey, String key) {  // **TODO** Auto-generated method stub  **return** jedisCluster.hget(hkey, key);  }  @Override  **public** **long** hset(String hkey, String key, String value) {  // **TODO** Auto-generated method stub  **return** jedisCluster.hset(hkey, key, value);  }  @Override  **public** **long** incr(String key) {  // **TODO** Auto-generated method stub  **return** jedisCluster.incr(key);  }  @Override  **public** **long** expire(String key, **int** second) {  **return** jedisCluster.expire(key, second);  }  @Override  **public** **long** ttl(String key) {  // **TODO** Auto-generated method stub  **return** jedisCluster.ttl(key);  }  @Override  **public** **long** del(String key) {  // **TODO** Auto-generated method stub  **return** jedisCluster.del(key);  }  @Override  **public** **long** hdel(String hkey, String key) {  // **TODO** Auto-generated method stub  **return** jedisCluster.hdel(hkey, key);  }  } |

并且根据使用的redis版本，把对于的bean打开。

applicationContext-jedis.xml

|  |
| --- |
| <bean id="jedisClientSingle" class="com.cf.rest.dao.impl.JedisClientSingle" />  <bean id="jedisClientCluster" class="com.cf.rest.dao.impl.JedisClientCluster" /> |

实现业务数据缓存：

基本要求：添加的缓存功能不能影响正常的业务。也就是说在RestServiceImpl中读取数据之前，先从缓存中去取数据，如果缓存中没有数据的场合才到数据库去中取，并且把从数据库中取得的数据，也需要放入缓存中，方便下次从缓存中去。如果缓存中存在该数据则直接取出。

RestServiceImpl.java

|  |
| --- |
| package com.cf.rest.service.impl;  import org.apache.commons.lang3.StringUtils;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import com.alibaba.druid.support.json.JSONUtils;  import com.alibaba.fastjson.JSON;  import com.cf.mapper.UserMapper;  import com.cf.pojo.User;  import com.cf.rest.dao.JedisClient;  import com.cf.rest.service.RestService;  import com.cf.utils.JsonUtils;  @Service  public class RestServiceImpl implements RestService {  @Autowired  private UserMapper userMapper;    @Autowired  private JedisClient jedisClient;    @Override  public User selectByPrimaryKey(Long id) {    // 从缓存中提取数据  try {  String result = jedisClient.hget("UserInfo", id + "");  if (!StringUtils.isBlank(result)) {  User user = JsonUtils.jsonToPojo(result, User.class);  return user;  }  } catch (Exception e) {  e.printStackTrace();  }    User user = userMapper.selectByPrimaryKey(id);    // 向缓存中添加数据  try {  String catchStr = JSON.toJSONString(user);  jedisClient.hset("UserInfo", id + "", catchStr);  } catch (Exception e) {  e.printStackTrace();  }  return user;  }  } |

缓存同步问题：

针对缓存中的数据，被其他操作修改的场合

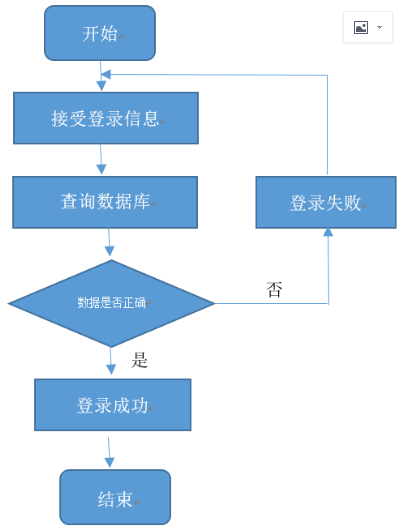
处理：

在做修改或删除时，如果是缓存中的数据，则需要情况缓存即可。下次查询时重新加载就满足了要求。

JedisClient.del, hdel.

# 十四、单点登录SSO原理分析

传统登录验证流程图：



如果的一个工程的系统，就采用这种模式最为简单。如果系统比较多，采用这种模式会导致每个系统都需要登录、注册等。

为什么要使用单点登录系统？

现在业务系统的壮大，很多服务会进行拆分，会做SOA服务，会使用dubbo做微服务，或者分布式的系统，

这样在服务与服务之间，或者系统与系统之间都是通过HTTP或者restful来进行通信的，

在以往的单系统应用中，我们都是把user存入session中的，需要用到的时候随时取，如果取不到就跳转到登录注册页面，非常简单的原理

但是在现如今的分布式应用中，如何保证session同步呢？

解决方案：

1. tomcat有一个session同步方案，就是一个传播机制，打个比方有A B C 3台tomcat，这3台tomcat的user信息都在session中并且保持一致，如果其中一台的user信息变化了，那么就会传播至另外两台，则实现同步，这样做没问题，但是仅仅只是在做tomcat集群的时候tomcat很少的时候会用，一旦集群增大，有100台，那么就互相传播吧，传播是需要性能损耗的，那么整个系统或网站的性能就会被拉低。

2. nginx 非粘性session，说穿了就是一个session绑定传播，起初user的session在tomcatA上，tomcatA宕机了，那么session会把所有的信息传播到tomcatB，以此实现session共享，但是这也有个问题，就是传播的时候需要等待，快的时候1分钟左右，慢的时候要5分钟，用户的耐性有限，所以也不能这么做。

3. 自己研发一套session高性能共享系统，我见过有这么做的公司，但是需要时间人力成本，所以不建议，如果你是BAT，随意。

4. SSO解决方案，目前比较流行的方案，自行开发一套单点登录系统，

什么是SSO

SSO英文全称Single Sign On，单点登录。SSO是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。它包括可以将这次主要的登录映射到其他应用中用于同一个用户的登录的机制。

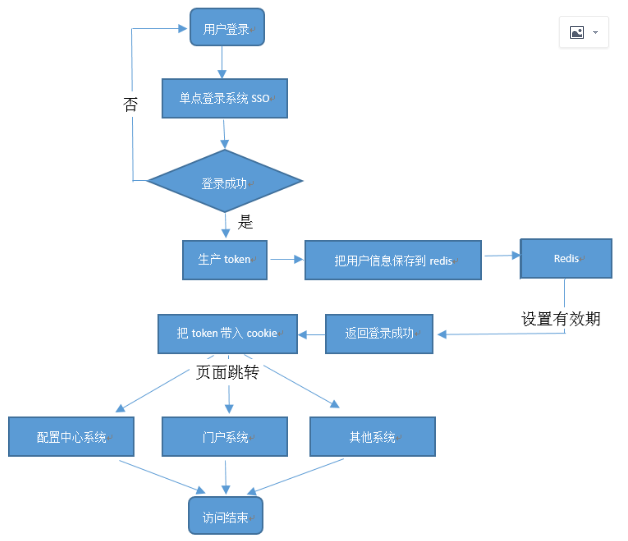
实现单点登录系统，提供服务接口。把session数据存放在redis。

Redis可以设置key的生存时间、访问速度快效率高。

优点：redis存取速度快，不会出现多个节点session复制的问题。效率高。

缺点：需要程序员开发。

SSO登录业务流程图：



这么做的目的就是在一个系统中登录后，访问其他的系统就不用再登录了。

那么系统怎么知道你是否已经登录了呢，这就需要一个标志，这个标志放在哪里呢，我们可以放在cookie中。

生成token：用来标识用户唯一的id，根据这个id用来保存用户信息，key就是token，value就是用户信息。

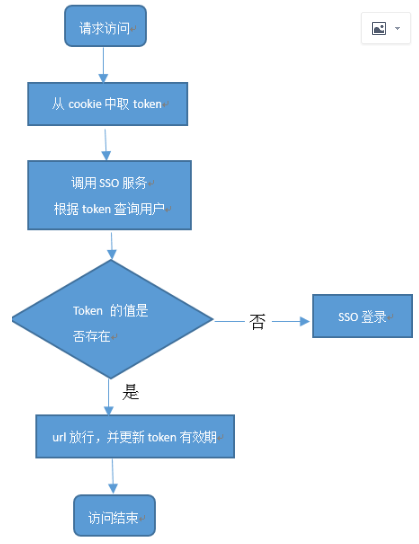
保存到redis中，设置有效时间。

为了保证其他系统能在访问的时候知道你是谁。需要把token写入到cookie中。正常的登录系统实际上也是将sessionid写入到了cookie中。

SSO访问业务流程图：

我们再次访问其他系统时，从cookie中取出token，调用sso服务，根据token查询用户信息。

sso在接收到token之后，在redis中进行查询，存在则代表已经登录，否则需要跳转到sso进行登录处理。



在这个sso中，并没有使用session。

# 十五、单点登录SSO工程搭建

功能：

创建单点登录系统，独立的工程

发布登录、注册、登出的接口服务

单点登录系统实现登录、注册等功能

## 1、创建单点登录工程

特点：

要访问redis

要访问数据库

发布服务

使用的技术：

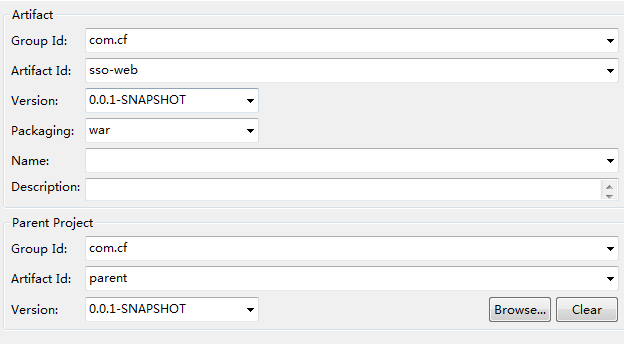
Mybatis

Spring

Springmvc

Redis

创建maven工程：

、 配置文件参考restful-web工程，需要修改部分如下：

applicationContext-trans.xml

|  |
| --- |
| <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* com.cf.sso.service.\*.\*(..))"* /> |

applicationContext-service.xml

|  |
| --- |
| <context:component-scan base-package="com.cf.sso.service"></context:component-scan> |

applicationContext-jedis.xml

|  |
| --- |
| <bean id="jedisClientSingle" class="com.cf.sso.dao.impl.JedisClientSingle" />  <bean id="jedisClientCluster" class="com.cf.sso.dao.impl.JedisClientCluster" /> |

并且把相关的接口类和实现类复制到sso-web对于的路径下。

Springmvc.xml

|  |
| --- |
| <context:component-scan base-package=*"com.cf.sso.controller"* />  <mvc:annotation-driven />  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/jsp/"* />  <property name=*"suffix"* value=*".jsp"* />  </bean>  <!-- 资源映射 -->  <mvc:resources location=*"/WEB-INF/css/"* mapping=*"/css/\*\*"*/>  <mvc:resources location=*"/WEB-INF/js/"* mapping=*"/js/\*\*"*/> |

Pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.cf</groupId>  <artifactId>parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.cf</groupId>  <artifactId>sso-web</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>    <dependencies>  <!-- 依赖config-mapper工程 -->  <dependency>  <groupId>com.cf</groupId>  <artifactId>config-mapper</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  </dependency>  <!-- httprequest -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <scope>provided</scope>  </dependency>  <!-- Redis客户端 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  </dependency>    <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8085</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

测试跳转页面controller

|  |
| --- |
| @Controller  **public** **class** IndexController {  @RequestMapping("/index")  **public** String shouIndex() {  **return** "index";  }    } |

# 十六、单点登录SSO服务规划

规划sso相关的接口发布：

## 1、注册接口

a、检查数据是否可用，然后在进行注册操作。

|  |  |
| --- | --- |
| 请求方法 | GET |
| URL | [http://localhost:8085/user/check/{param}/{type](http://localhost:8085/user/check/%7Bparam%7D/%7Btype)} |
| 参数说明 | 格式如：zhangsan/1，其中zhangsan是校验的数据，type为类型，可选参数1、2分别代表username、phone  可选参数callback：如果有此参数表示此方法为jsonp请求，需要支持jsonp。 |
| 示例 | <http://localhost:8085/user/check/zhangsan/1> |
| 返回值 | {  status: 200 //200 成功  msg: "OK" // 返回信息消息  data: false // 返回数据，true：数据可用，false：数据不可用  } |

### b、用户注册

|  |  |
| --- | --- |
| 请求方法 | POST |
| URL | <http://localhost:8085/user/register> |
| 参数 | username //用户名  password //密码  phone //手机号 |
| 示例 | <http://localhost:8085/user/register> |
| 返回值 | {  status: 400  msg: "注册失败. 请校验数据后请再提交数据."  data: null  } |

## 2、用户登录

|  |  |
| --- | --- |
| 请求方法 | POST |
| URL | <http://localhost:8085/user/login> |
| 参数 | username //用户名  password //密码 |
| 返回值 | {  status: 200  msg: "OK"  data: "adw5cb51321b13d123abcasad" //登录成功，返回token  } |

## 3、通过token查询用户信息（已经登录的场合访问其他系统）

|  |  |
| --- | --- |
| 请求方法 | GET |
| URL | [http://localhost:8085/user/token/{token](http://localhost:8085/user/token/%7Btoken)} |
| 参数 | token //用户登录凭证  callback//jsonp回调方法  参数说明    可选参数callback：如果有此参数表示此方法为jsonp请求，需要支持jsonp。 |
| 示例 | <http://localhost:8085/user/token/fe5cb546aeb3ce1bf37abcb08a40493e> |
| 返回值 | {  status: 200  msg: "OK"  data: "{"id":1,"username":"zhangsan","phone":"123400807944","created":1136476000,"updated":1414119179000}"  } |

## 4、安全退出

|  |  |
| --- | --- |
| 请求方法 | GET |
| URL | [http://localhost:8085/user/logout/{token](http://localhost:8085/user/logout/%7Btoken)} |
| 参数 | token //用户登录凭证  callback//jsonp回调方法  参数说明    可选参数callback：如果有此参数表示此方法为jsonp请求，需要支持jsonp。 |
| 示例 | [http://localhost:8085/user/logout/](http://localhost:8085/user/logout/fe5cb546aeb3ce1bf37abcb08a40493e)ads23ad1adadf232342ac |
| 返回值 | {  status: 200  msg: "OK"  data: ""  } |

鉴于上诉结果都需要返回状态、信息、数据等状态，建议进行自定义类封装。

在common工程中添加Result.java类。

|  |
| --- |
| package com.cf.utils;  import java.util.List;  import com.fasterxml.jackson.databind.JsonNode;  import com.fasterxml.jackson.databind.ObjectMapper;  /\*\*  \* 自定义响应结构  \*/  public class Result {  // 定义jackson对象  private static final ObjectMapper MAPPER = new ObjectMapper();  // 响应业务状态  private Integer status;  // 响应消息  private String msg;  // 响应中的数据  private Object data;  public static Result build(Integer status, String msg, Object data) {  return new Result(status, msg, data);  }  public static Result ok(Object data) {  return new Result(data);  }  public static Result ok() {  return new Result(null);  }  public Result() {  }  public static Result build(Integer status, String msg) {  return new Result(status, msg, null);  }  public Result(Integer status, String msg, Object data) {  this.status = status;  this.msg = msg;  this.data = data;  }  public Result(Object data) {  this.status = 200;  this.msg = "OK";  this.data = data;  }  public Integer getStatus() {  return status;  }  public void setStatus(Integer status) {  this.status = status;  }  public String getMsg() {  return msg;  }  public void setMsg(String msg) {  this.msg = msg;  }  public Object getData() {  return data;  }  public void setData(Object data) {  this.data = data;  }  /\*\*  \* 将json结果集转化为Result对象  \*  \* @param jsonData json数据  \* @param clazz Result中的object类型  \* @return  \*/  public static Result formatToPojo(String jsonData, Class<?> clazz) {  try {  if (clazz == null) {  return MAPPER.readValue(jsonData, Result.class);  }  JsonNode jsonNode = MAPPER.readTree(jsonData);  JsonNode data = jsonNode.get("data");  Object obj = null;  if (clazz != null) {  if (data.isObject()) {  obj = MAPPER.readValue(data.traverse(), clazz);  } else if (data.isTextual()) {  obj = MAPPER.readValue(data.asText(), clazz);  }  }  return build(jsonNode.get("status").intValue(), jsonNode.get("msg").asText(), obj);  } catch (Exception e) {  return null;  }  }  /\*\*  \* 没有object对象的转化  \*  \* @param json  \* @return  \*/  public static Result format(String json) {  try {  return MAPPER.readValue(json, Result.class);  } catch (Exception e) {  e.printStackTrace();  }  return null;  }  /\*\*  \* Object是集合转化  \*  \* @param jsonData json数据  \* @param clazz 集合中的类型  \* @return  \*/  public static Result formatToList(String jsonData, Class<?> clazz) {  try {  JsonNode jsonNode = MAPPER.readTree(jsonData);  JsonNode data = jsonNode.get("data");  Object obj = null;  if (data.isArray() && data.size() > 0) {  obj = MAPPER.readValue(data.traverse(),  MAPPER.getTypeFactory().constructCollectionType(List.class, clazz));  }  return build(jsonNode.get("status").intValue(), jsonNode.get("msg").asText(), obj);  } catch (Exception e) {  return null;  }  }  } |

# 十七、单点登录SSO的服务发布

1. 注册接口

功能分解：接收url中的两个参数，一个是校验的内容，一个是校验的数据类型。，并且反馈结果。

A、用户检查

Dao层

config-mapper工程中添加dao方法

UserMapper.java

|  |
| --- |
| List<User> selectByCondition(User user); |

UserMapper.xml

|  |
| --- |
| <select id=*"selectByCondition"* resultMap=*"BaseResultMap"* parameterType=*"com.cf.pojo.User"* >  select  <include refid=*"Base\_Column\_List"* />  from user  where 1=1  <if test=*"username != null"* >  and username = #{username,jdbcType=VARCHAR}  </if>  <if test=*"password != null"* >  and password = #{password,jdbcType=VARCHAR}  </if>  <if test=*"phone != null"* >  and phone = #{phone,jdbcType=VARCHAR}  </if>  <if test=*"email != null"* >  and email = #{email,jdbcType=VARCHAR}  </if>  </select> |

sso-web工程中编写service

UserService.java

|  |
| --- |
| **package** com.cf.sso.service;  **import** com.cf.pojo.User;  **import** com.cf.utils.Result;  **public** **interface** UserService {  Result checkData(String context, Integer type);  Result createUser(User user);  Result userLogin(String username, String password);  Result getUserByToken(String token);  Result userLogout(String token);  } |

UserServiceImpl.java

|  |
| --- |
| **package** com.cf.sso.service.impl;  **import** java.util.Date;  **import** java.util.List;  **import** java.util.UUID;  **import** org.apache.commons.lang3.StringUtils;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** org.springframework.util.DigestUtils;  **import** com.alibaba.fastjson.JSON;  **import** com.cf.mapper.UserMapper;  **import** com.cf.pojo.User;  **import** com.cf.sso.dao.JedisClient;  **import** com.cf.sso.service.UserService;  **import** com.cf.utils.JsonUtils;  **import** com.cf.utils.Result;  @Service  **public** **class** UserServiceImpl **implements** UserService {  @Autowired  **private** UserMapper userMapper;  @Autowired  **private** JedisClient jedisClient;  @Override  **public** Result checkData(String context, Integer type) {  //构造查询条件  User user = **new** User();  **if** (type == 1) {  user.setUsername(context);  } **else** **if** (type == 2) {  user.setPhone(context);  }  List<User> result = userMapper.selectByCondition(user);  **if** (result == **null** || result.size() == 0) {  **return** Result.*ok*(**true**);  }  **return** Result.*ok*(**false**);  }  @Override  **public** Result createUser(User user) {  user.setCreated(**new** Date());  user.setUpdated(**new** Date());  user.setPassword(DigestUtils.*md5DigestAsHex*(user.getPassword().getBytes()));  userMapper.insert(user);  **return** Result.*ok*();  }  @Override  **public** Result userLogin(String username, String password) {  User user = **new** User();  user.setUsername(username);  List<User> list = userMapper.selectByCondition(user);  **if** (list == **null** || list.size() == 0) {  **return** Result.*build*(400, "用户名或密码错误");  }  User result = list.get(0);  **if** (!DigestUtils.*md5DigestAsHex*(password.getBytes()).equals(result.getPassword())) {  **return** Result.*build*(400, "用户名或密码错误");  }  // 验证通过的场合  String token = UUID.*randomUUID*().toString();  result.setPassword(**null**);  jedisClient.set("USER\_SESSION\_KEY" + ":" + token, JsonUtils.*objectToJson*(result));  jedisClient.expire("USER\_SESSION\_KEY" + ":" + token, 900);    **return** Result.*ok*(token);  }  @Override  **public** Result getUserByToken(String token) {  String json = jedisClient.get("USER\_SESSION\_KEY" + ":" + token);  **if** (StringUtils.*isBlank*(json)) {  **return** Result.*build*(400, "此session已经过期，请重新登录");  }  jedisClient.expire("USER\_SESSION\_KEY" + ":" + token, 900);  **return** Result.*ok*(JsonUtils.*jsonToPojo*(json, User.**class**));  }  @Override  **public** Result userLogout(String token) {  **long** delCnt = jedisClient.del("USER\_SESSION\_KEY" + ":" + token);  **if** (delCnt == 0) {  **return** Result.*build*(400, "此session无效，无法登出");  }  **return** Result.*ok*();  }  } |

UserController.java

|  |
| --- |
| **package** com.cf.sso.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** com.cf.pojo.User;  **import** com.cf.sso.service.UserService;  **import** com.cf.utils.Result;  @Controller  **public** **class** UserController {  @Autowired  **private** UserService userService;    @RequestMapping(value = "user/check/{param}/{type}", method = RequestMethod.***GET***)  @ResponseBody  **public** Result checkData(@PathVariable("param") String param, @PathVariable("type") Integer type) {  Result result = userService.checkData(param, type);  System.***out***.println("checkdata-------------------" + result);  **return** result;  }    @RequestMapping(value = "user/register", method = RequestMethod.***POST***)  @ResponseBody  **public** Result createUser(User user) {  Result result = userService.createUser(user);  System.***out***.println("createUser-------------------" + result);  **return** result;  }    @RequestMapping(value = "user/login", method = RequestMethod.***POST***)  @ResponseBody  **public** Result userLogin(String username, String password) {  Result result = userService.userLogin(username, password);  System.***out***.println("userLogin-------------------" + result);  **return** result;  }    @RequestMapping(value = "user/token/{token}", method = RequestMethod.***GET***)  @ResponseBody  **public** Result getUserByToken(@PathVariable("token") String token) {  Result result = userService.getUserByToken(token);  System.***out***.println("getUserByToken-------------------" + result);  **return** result;  }    @RequestMapping(value = "user/logout/{token}", method = RequestMethod.***GET***)  @ResponseBody  **public** Result logout(@PathVariable("token") String token) {  Result result = userService.userLogout(token);  System.***out***.println("logout-------------------" + result);  **return** result;  }  } |

启动测试： clean tomcat7:run

http://localhost:8085/user/check/zhangsan/1

其他功能在后续开发中测试。

# 十八、单点登录系统SSO的注册功能实现

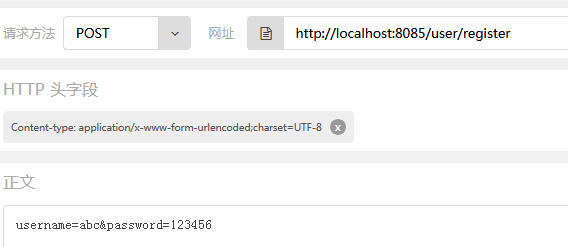
前期准备：

Jquery导入工程中

创建对于的login.jsp、register.jsp

测试验证服务接口：

由于家里put过滤器，所以在用测试工具时需要指定Content-type:application/x-www-form-urlencoded;charset=UTF-8



注册页面功能分解：

1. 用户名不能重复
2. 密码和确认必须一致
3. 用户名、密码不能为空
4. 手机号码不能空，并且不能重复
5. 验证通过后调用服务进行注册。

Register.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <title>SSO注册页面</title>  <script type=*"text/javascript"* src=*"/js/jquery-1.6.4.js"*></script>  </head>  <body>  <h2 align=*center*>注册</h2>  <center>  <form id=*"personRegForm"* method=*"post"* onsubmit="return false;">  <table border=*"0"*>  <tr>  <td>用户名：</td>  <td><input type=*"text"* id=*"regName"* name=*"username"* /></td>  </tr>  <tr>  <td>密码：</td>  <td><input type=*"password"* id=*"pwd"* name=*"password"* /></td>  </tr>  <tr>  <td>重复密码：</td>  <td><input type=*"password"* id=*"pwdRepeat"* name=*"pwdRepeat"* /></td>  </tr>  <tr>  <td>手机：</td>  <td><input type=*"text"* id=*"phone"* maxlength=*"11"* name=*"phone"* /></td>  </tr>  <tr align=*"center"*>  <td colspan=*"2"*><input type=*"button"* id=*"registsubmit"*  value=*"立即注册"* onclick="REGISTER.reg();" /></td>  </tr>  </table>  </form>  您有账号？<a href=*"/sso/login"*><front color=*"GREEN"*>点击直接登录</front></a>  </center>  <script type=*"text/javascript"*>  **var** REGISTER={  param:{  //单点登录系统的url  surl:""  },  inputcheck:**function**(){  //不能为空检查  **if** ($("#regName").val() == "") {  alert("用户名不能为空");  $("#regName").focus();  **return** **false**;  }  **if** ($("#pwd").val() == "") {  alert("密码不能为空");  $("#pwd").focus();  **return** **false**;  }  **if** ($("#phone").val() == "") {  alert("手机号不能为空");  $("#phone").focus();  **return** **false**;  }  //密码检查  **if** ($("#pwd").val() != $("#pwdRepeat").val()) {  alert("确认密码和密码不一致，请重新输入！");  $("#pwdRepeat").select();  $("#pwdRepeat").focus();  **return** **false**;  }  **return** **true**;  },  beforeSubmit:**function**() {  //检查用户是否已经被占用  $.ajax({  url : REGISTER.param.surl + "/user/check/"+escape($("#regName").val())+"/1?r=" + Math.random(),  success : **function**(data) {  **if** (data.data) {  //检查手机号是否存在  $.ajax({  url : REGISTER.param.surl + "/user/check/"+$("#phone").val()+"/2?r=" + Math.random(),  success : **function**(data) {  **if** (data.data) {  REGISTER.doSubmit();  } **else** {  alert("此手机号已经被注册！");  $("#phone").select();  }  }  });  } **else** {  alert("此用户名已经被占用，请选择其他用户名");  $("#regName").select();  }  }  });    },  doSubmit:**function**() {  $.post("/user/register",$("#personRegForm").serialize(), **function**(data){  **if**(data.status == 200){  alert('用户注册成功，请登录！');  REGISTER.login();  } **else** {  alert("注册失败！");  }  });  },  login:**function**() {  location.href = "/sso/login";  **return** **false**;  },  reg:**function**() {  **if** (**this**.inputcheck()) {  **this**.beforeSubmit();  }  }  };  </script>  </body>  </html> |

SsoController.java

|  |
| --- |
| package com.cf.sso.controller;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  @Controller  @RequestMapping("/sso")  public class SsoController {  @RequestMapping("/register")  public String register() {  return "register";  }    @RequestMapping("/login")  public String login() {  return "login";  }  } |

# 十九、单点登录系统SSO的登录功能实现

登录功能的分解：

1. 注册完成后跳转到登录页面
2. 登录时需要验证用户名和密码不能为空
3. 点击登录时，把用户名和密码提交到sso系统的服务接口
4. Sso服务接口进行登录验证，并返回结果
5. 页面跳转（参考ssocontroller的login方法）
6. Login.jsp页面编写

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <%@ page trimDirectiveWhitespaces=*"true"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns=*"http://www.w3.org/1999/xhtml"*>  <head>  <title>SSO登录页面</title>  <script type=*"text/javascript"* src=*"/js/jquery-1.6.4.js"*></script>  </head>  <body>  <h2 align=*center*>登录</h2>  <center>  <form id=*"formlogin"* method=*"post"* onsubmit="return false;">  <table border=*"0"*>  <tr>  <td>用户名：</td>  <td><input type=*"text"* id=*"loginname"* name=*"username"* tabindex=*"1"* /></td>  </tr>  <tr>  <td>密码：</td>  <td><input type=*"password"* id=*"nloginpwd"* name=*"password"* tabindex=*"2"* /></td>  </tr>  <tr align=*"center"*>  <td colspan=*"2"*><input type=*"button"* id=*"loginsubmit"* value=*"登录"* tabindex=*"3"* /></td>  </tr>  </table>  </form>  <span><a href=*"/sso/register"*>免费注册&gt;&gt;</a></span>  </center>  <script type=*"text/javascript"*>  **var** redirectUrl = "${redirect}";  **var** LOGIN = {  checkInput : **function**() {  **if** ($("#loginname").val() == "") {  alert("用户名不能为空");  $("#loginname").focus();  **return** **false**;  }  **if** ($("#nloginpwd").val() == "") {  alert("密码不能为空");  $("#nloginpwd").focus();  **return** **false**;  }  **return** **true**;  },  doLogin : **function**() {  $.post("/user/login", $("#formlogin").serialize(), **function**(  data) {  **if** (data.status == 200) {  alert("登录成功！");  **if** (redirectUrl == "") {  location.href = "http://localhost:8083/index";  } **else** {  location.href = redirectUrl;  }  } **else** {  alert("登录失败，原因是：" + data.msg);  $("#loginname").select();  }  });  },  login : **function**() {  **if** (**this**.checkInput()) {  **this**.doLogin();  }  }  };  $(**function**() {  $("#loginsubmit").click(**function**() {  LOGIN.login();  });  });  </script>  </body>  </html> |

1. 登录页面的回调功能

通过在url中指定回调的url，登录成功后进行跳转。如：localhost:8085/sso/login?redirect=http://www.baidu.com

在SsoController.java

|  |
| --- |
| @RequestMapping("/login")  **public** String login(String redirect, Model model) {  model.addAttribute("redirect", redirect);  **return** "login";  } |

再次测试就会在登录成功后跳转到指定首页。

# 二十、业务系统与单点登录系统的整合

什么时候使用登录回调？

一般是在用户访问某个url或某个页面时为必须登录的情况，登录完成后需要跳转到登录前的页面。

这个用户权限在拦截器中进行实现。

如：在portal-web中国，一般首页不需要登录就可以访问，但是如果要访问个人中心之类的页面就要求必须登录。

## 1、在改造首页portal-web工程前，先index.jsp稍微改造一下：

默认情况下是需要读取登录状态，而且页面跳转正常。

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=utf-8"*  pageEncoding=*"utf-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>首页</title>  </head>  <body>  <h2 align=*center*>Portal index</h2>  <ul>  <li id=*"loginbar"*>您好！<a href=*"javascript:login()"*>[登录]</a>&nbsp;<a href=*"javascript:regist()"*>[免费注册]</a></li>  <li><s></s><a href=*"http://localhost:8083/my/mypage"*>个人中心</a></li>  </ul>  <script type=*"text/javascript"*>  **function** login() {  **return** location.href = "http://localhost:8085/sso/login";  }  **function** regist() {  **return** location.href = "http://localhost:8085/sso/register";  }  </script>  </body>  </html> |

在我们之前的代码中，并没有实现在生产token后，把token写入到cookie中的操作，所以即使我们现在登录后，页面仍然显示未登录。

如何实现？

在门户系统portal-web中点击登录链接跳转到登录页面，登录成功后，跳转到门户系统的首页。

在门户系统中需要从cookie中把token值取出来。所以必须在sso-web的登录成功后把token写入cookie。并且cookie的值必须在系统之间共享。

## 2、Cookie的共享？

A、设置demain；比如[www.abc.com](http://www.abc.com); sso.abc.com search.abc.com

需要设置domain为： .abc.com

1. 设置path： /

像我们的demo都是localhost，则不需要设置domain，设置path为 / 即可。

## 3、添加cookie的共同处理类

|  |
| --- |
| **package** com.cf.utils;  **import** java.io.UnsupportedEncodingException;  **import** java.net.URLDecoder;  **import** java.net.URLEncoder;  **import** javax.servlet.http.Cookie;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  /\*\*  \*  \* Cookie 工具类  \*  \*/  **public** **final** **class** CookieUtils {  /\*\*  \* 得到Cookie的值, 不编码  \*  \* **@param** request  \* **@param** cookieName  \* **@return**  \*/  **public** **static** String getCookieValue(HttpServletRequest request, String cookieName) {  **return** *getCookieValue*(request, cookieName, **false**);  }  /\*\*  \* 得到Cookie的值,  \*  \* **@param** request  \* **@param** cookieName  \* **@return**  \*/  **public** **static** String getCookieValue(HttpServletRequest request, String cookieName, **boolean** isDecoder) {  Cookie[] cookieList = request.getCookies();  **if** (cookieList == **null** || cookieName == **null**) {  **return** **null**;  }  String retValue = **null**;  **try** {  **for** (**int** i = 0; i < cookieList.length; i++) {  **if** (cookieList[i].getName().equals(cookieName)) {  **if** (isDecoder) {  retValue = URLDecoder.*decode*(cookieList[i].getValue(), "UTF-8");  } **else** {  retValue = cookieList[i].getValue();  }  **break**;  }  }  } **catch** (UnsupportedEncodingException e) {  e.printStackTrace();  }  **return** retValue;  }  /\*\*  \* 得到Cookie的值,  \*  \* **@param** request  \* **@param** cookieName  \* **@return**  \*/  **public** **static** String getCookieValue(HttpServletRequest request, String cookieName, String encodeString) {  Cookie[] cookieList = request.getCookies();  **if** (cookieList == **null** || cookieName == **null**) {  **return** **null**;  }  String retValue = **null**;  **try** {  **for** (**int** i = 0; i < cookieList.length; i++) {  **if** (cookieList[i].getName().equals(cookieName)) {  retValue = URLDecoder.*decode*(cookieList[i].getValue(), encodeString);  **break**;  }  }  } **catch** (UnsupportedEncodingException e) {  e.printStackTrace();  }  **return** retValue;  }  /\*\*  \* 设置Cookie的值 不设置生效时间默认浏览器关闭即失效,也不编码  \*/  **public** **static** **void** setCookie(HttpServletRequest request, HttpServletResponse response, String cookieName,  String cookieValue) {  *setCookie*(request, response, cookieName, cookieValue, -1);  }  /\*\*  \* 设置Cookie的值 在指定时间内生效,但不编码  \*/  **public** **static** **void** setCookie(HttpServletRequest request, HttpServletResponse response, String cookieName,  String cookieValue, **int** cookieMaxage) {  *setCookie*(request, response, cookieName, cookieValue, cookieMaxage, **false**);  }  /\*\*  \* 设置Cookie的值 不设置生效时间,但编码  \*/  **public** **static** **void** setCookie(HttpServletRequest request, HttpServletResponse response, String cookieName,  String cookieValue, **boolean** isEncode) {  *setCookie*(request, response, cookieName, cookieValue, -1, isEncode);  }  /\*\*  \* 设置Cookie的值 在指定时间内生效, 编码参数  \*/  **public** **static** **void** setCookie(HttpServletRequest request, HttpServletResponse response, String cookieName,  String cookieValue, **int** cookieMaxage, **boolean** isEncode) {  *doSetCookie*(request, response, cookieName, cookieValue, cookieMaxage, isEncode);  }  /\*\*  \* 设置Cookie的值 在指定时间内生效, 编码参数(指定编码)  \*/  **public** **static** **void** setCookie(HttpServletRequest request, HttpServletResponse response, String cookieName,  String cookieValue, **int** cookieMaxage, String encodeString) {  *doSetCookie*(request, response, cookieName, cookieValue, cookieMaxage, encodeString);  }  /\*\*  \* 删除Cookie带cookie域名  \*/  **public** **static** **void** deleteCookie(HttpServletRequest request, HttpServletResponse response,  String cookieName) {  *doSetCookie*(request, response, cookieName, "", -1, **false**);  }  /\*\*  \* 设置Cookie的值，并使其在指定时间内生效  \*  \* **@param** cookieMaxage cookie生效的最大秒数  \*/  **private** **static** **final** **void** doSetCookie(HttpServletRequest request, HttpServletResponse response,  String cookieName, String cookieValue, **int** cookieMaxage, **boolean** isEncode) {  **try** {  **if** (cookieValue == **null**) {  cookieValue = "";  } **else** **if** (isEncode) {  cookieValue = URLEncoder.*encode*(cookieValue, "utf-8");  }  Cookie cookie = **new** Cookie(cookieName, cookieValue);  **if** (cookieMaxage > 0)  cookie.setMaxAge(cookieMaxage);  **if** (**null** != request) {// 设置域名的cookie  String domainName = *getDomainName*(request);  System.***out***.println(domainName);  **if** (!"localhost".equals(domainName)) {  cookie.setDomain(domainName);  }  }  cookie.setPath("/");  response.addCookie(cookie);  } **catch** (Exception e) {  e.printStackTrace();  }  }  /\*\*  \* 设置Cookie的值，并使其在指定时间内生效  \*  \* **@param** cookieMaxage cookie生效的最大秒数  \*/  **private** **static** **final** **void** doSetCookie(HttpServletRequest request, HttpServletResponse response,  String cookieName, String cookieValue, **int** cookieMaxage, String encodeString) {  **try** {  **if** (cookieValue == **null**) {  cookieValue = "";  } **else** {  cookieValue = URLEncoder.*encode*(cookieValue, encodeString);  }  Cookie cookie = **new** Cookie(cookieName, cookieValue);  **if** (cookieMaxage > 0)  cookie.setMaxAge(cookieMaxage);  **if** (**null** != request) {// 设置域名的cookie  String domainName = *getDomainName*(request);  System.***out***.println(domainName);  **if** (!"localhost".equals(domainName)) {  cookie.setDomain(domainName);  }  }  cookie.setPath("/");  response.addCookie(cookie);  } **catch** (Exception e) {  e.printStackTrace();  }  }  /\*\*  \* 得到cookie的域名  \*/  **private** **static** **final** String getDomainName(HttpServletRequest request) {  String domainName = **null**;  String serverName = request.getRequestURL().toString();  **if** (serverName == **null** || serverName.equals("")) {  domainName = "";  } **else** {  serverName = serverName.toLowerCase();  serverName = serverName.substring(7);  **final** **int** end = serverName.indexOf("/");  serverName = serverName.substring(0, end);  **final** String[] domains = serverName.split("\\.");  **int** len = domains.length;  **if** (len > 3) {  // www.xxx.com.cn  domainName = "." + domains[len - 3] + "." + domains[len - 2] + "." + domains[len - 1];  } **else** **if** (len <= 3 && len > 1) {  // xxx.com or xxx.cn  domainName = "." + domains[len - 2] + "." + domains[len - 1];  } **else** {  domainName = serverName;  }  }  **if** (domainName != **null** && domainName.indexOf(":") > 0) {  String[] ary = domainName.split("\\:");  domainName = ary[0];  }  **return** domainName;  }  } |

在什么场合进行cookie的设置？

在我们的sso-web的登录成功后进行设置

UserService.java

|  |
| --- |
| Result userLogin(String username, String password, HttpServletRequest request, HttpServletResponse response); |

UserController.java

|  |
| --- |
| @RequestMapping(value = "user/login", method = RequestMethod.***POST***)  @ResponseBody  **public** Result userLogin(String username, String password, HttpServletRequest request, HttpServletResponse response) {  Result result = userService.userLogin(username, password, request, response);  System.***out***.println("userLogin-------------------" + result);  **return** result;  } |

UserServiceImpl.java

|  |
| --- |
| // 添加写cookie的逻辑，cookie的有效期是关闭浏览器就失效  CookieUtils.*setCookie*(request, response, "SSO\_TOKEN", token); |

## 4、在业务系统A（portal-web）中读取cookie信息

从cookie中取出token，然后后根据这个token去取用户信息，调用sso-web服务来完成。需要支持jsonp调用。

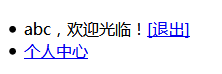
UserController.java(sso-web)

|  |
| --- |
| @RequestMapping(value = "user/token/{token}", method = RequestMethod.***GET***)  @ResponseBody  **public** Object getUserByToken(@PathVariable("token") String token, String callback) {  Result result = userService.getUserByToken(token);  System.***out***.println("getUserByToken-------------------" + result);    **if** (StringUtils.*isBlank*(callback)) {  **return** result;  } **else** {  // jsonp调用  MappingJacksonValue mappingJackson = **new** MappingJacksonValue(result);  mappingJackson.setJsonpFunction(callback);  **return** mappingJackson;  }  } |

## 5、在portal-web这个系统index.jsp页面中添加读取cookie的操作。

页面加载时去读取cookie，并显示相应的用户信息。

|  |
| --- |
| <script type=*"text/javascript"* src=*"js/jquery-1.6.4.js"*></script>  <script type=*"text/javascript"* src=*"js/jquery.cookie.js"*></script>  **var** S = SSO = {  checkLogin : **function**(){  **var** \_ticket = $.cookie("SSO\_TOKEN");  **if**(!\_ticket){  **return** ;  }  $.ajax({  url : "http://localhost:8085/user/token/" + \_ticket,  dataType : "jsonp",  type : "GET",  success : **function**(data){  **if**(data.status == 200){  **var** username = data.data.username;  **var** html = username + "，欢迎光临！<a href=\"http://localhost:8085/user/logout/" + \_ticket + " \" >[退出]</a>";  $("#loginbar").html(html);  }  }  });  }  }  $(**function**(){  // 查看是否已经登录，如果已经登录查询登录信息  S.checkLogin();  }); |



目前我们实现了单点登录的登录后，其他系统也采用这种方式来获取token，为了更好的体现这个用户是否已经登录，我们来定制一些页面进行强制登录验证。

## 6、拦截url进行强制登录

当用户访问“个人中”页面的时候，进行url拦截，并强制登录。

完善登出功能，登出后跳转到protal-web的index页面

|  |
| --- |
| @RequestMapping(value = "user/logout/{token}", method = RequestMethod.***GET***)  @ResponseBody  **public** **void** logout(@PathVariable("token") String token, HttpServletRequest request, HttpServletResponse response) **throws** IOException {  Result result = userService.userLogout(token);  System.***out***.println("logout-------------------" + result);    response.sendRedirect("http://localhost:8083/index");  } |

强制登录？

创建拦截器：

1. 需要实现handlerInterceptor接口

创建一个LoginIntercopter ，继承handlerInterceptor接口，实现它的preHandle回调方法。

1. 实现调用sso-web服务取得token

Portal-web

UserService.java

|  |
| --- |
| **public** **interface** UserService {  User getUserByToken(String token);  } |

UserServiceImpl.java

|  |
| --- |
| package com.cf.portal.service.impl;  import com.cf.pojo.User;  import com.cf.portal.service.UserService;  import com.cf.utils.HttpClientUtil;  import com.cf.utils.Result;  public class UserServiceImpl implements UserService {  @Override  public User getUserByToken(String token) {    String json = HttpClientUtil.doGet("http://localhost:8085/user/token/" + token);  System.out.println("-------------------getUserByToken " + json);    Result result = Result.formatToPojo(json, User.class);  if (result.getStatus() == 200) {  User user = (User)result.getData();  return user;  }  return null;  }  } |

Pom.xml

|  |
| --- |
| <dependency>  <groupId>com.cf</groupId>  <artifactId>config-mapper</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency> |

1. 实现拦截逻辑(portal-web)

LoginIntercopter.java

|  |
| --- |
| **package** com.cf.intercopter;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.servlet.HandlerInterceptor;  **import** org.springframework.web.servlet.ModelAndView;  **import** com.cf.pojo.User;  **import** com.cf.portal.service.UserService;  **import** com.cf.utils.CookieUtils;  **public** **class** LoginIntercopter **implements** HandlerInterceptor {  @Autowired  **private** UserService userService;  @Override  **public** **void** afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)  **throws** Exception {  // **TODO** Auto-generated method stub    }  @Override  **public** **void** postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)  **throws** Exception {  // **TODO** Auto-generated method stub    }  @Override  **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {  // 判断cookie是否存在  String token = CookieUtils.*getCookieValue*(request, "SSO\_TOKEN");  User user = userService.getUserByToken(token);  **if** (user == **null**) {  response.sendRedirect("http://localhost:8085/sso/login" + "?redirect=" + request.getRequestURL());  **return** **false**;  }  System.***out***.println("-------------------preHandle " + user.getUsername());  request.setAttribute("user", user);  **return** **true**;  }  } |

IndexController.java

|  |
| --- |
| @RequestMapping("/my/mypage")  **public** String mypage() {  **return** "mypage";  } |

springmvc.xml

|  |
| --- |
| <mvc:interceptors>  <mvc:interceptor>  <mvc:mapping path=*"/my/\*\*"*/>  <bean class=*"com.cf.intercopter.LoginIntercopter"*></bean>  </mvc:interceptor>  </mvc:interceptors> |

首先访问，正常

<http://localhost:8083/index>

然后访问需要登录页面，localhost:8083/my/mypage

跳转到单点登录页面：<http://localhost:8085/sso/login?redirect=http://localhost:8083/my/mypage>

登录成功后跳转到之前的页面：http://localhost:8083/my/mypage

注意：目前只演示了把portal-web加入到单点系统中来进行统一登录认证，其他系统参考上述方法进行改造。