**淘淘商城**

**第七天**

# 课程计划

1. RESTful介绍
2. 使用RESTful开发接口
3. HttpClient介绍
4. HttpClient接口调用
5. 登录流程分析
6. 搭建SSO系统

# RESTful Web Service介绍

* Roy Thomas Fielding博士2000年提出的
* REST是英文Representational State Transfer的缩写
* 表象化状态转变 或者 表述性状态转移
* REST是Web服务的一种架构风格
* 使用HTTP、URI等广泛流行的标准和协议
* 轻量级、跨平台、跨语言的架构设计。



## REST到底是什么？

* REST是一种设计风格。它不是一种标准，也不是一种软件，而是一种思想。
* REST通常基于使用HTTP，URI，和XML，JSON以及HTML这些现有的广泛流行的协议和标准。

## RESTful是什么

* RESTful对应的中文是 REST式的。
* RESTful Web Service是一种常见的REST的应用，是遵守了REST风格的web服务。

两种主要的web服务

JAX-RS RESTful Web Service

JAX-WS Web Service

## REST 架构的主要原则

* 网络上的所有事物都可被抽象为资源（Resource）
* 每个资源都有一个唯一的资源标识符（Resource Identifier）
* 同一资源具有多种表现形式(xml、json等)
* 对资源的各种操作不会改变资源标识符
* 所有的操作都是无状态的（Stateless）
* 符合REST原则的架构方式即可称为RESTful

### 无状态性

* 无状态性使得客户端和服务器端不必保存对方的详细信息，服务器只需要处理当前 Request，而不必了解前面 Request 的历史。
* 从而可以更容易地释放资源。
* 让服务器充分利用 Pool 技术来提高稳定性和性能。

## 资源操作

http://taotao.com/item/

GET： 获取一个资源   
POST： 创建一个新的资源   
PUT： 修改一个资源的状态   
DELETE ：删除一个资源

资源展现

XML

JSON

……

**原来的方式**

[http://127.0.0.1/user/queryUser/{id}](http://127.0.0.1/user/queryUser/%7bid%7d)  **GET方法，根据用户id获取数据**

<http://127.0.0.1/user/update>User  **POST方法，用户修改**

<http://127.0.0.1/user/save>User  **POST方法，用户新增**

[http://127.0.0.1/user/deleteUser/{id}](http://127.0.0.1/user/delete/%7bid%7d)  **GET/POST方法，用户根据id删除**

**RESTful**

[http://127.0.0.1/user/ {id}](http://127.0.0.1/user/%20%7bid%7d)  **GET方法，根据用户id获取数据**

[http://127.0.0.1/user/](http://127.0.0.1/user/update)  **PUT方法，用户修改**

[http://127.0.0.1/user/](http://127.0.0.1/user/save)  **POST方法，用户新增**

http://127.0.0.1/user/{id}  **DELETE方法，用户根据id删除**

开发的接口，web服务更加的简洁

## REST接口定义



* 幂等性：对同一REST接口的多次访问，得到的资源状态是相同的。
* 安全性：对该REST接口访问，不会使服务器端资源状态发生改变。

## 最佳实践

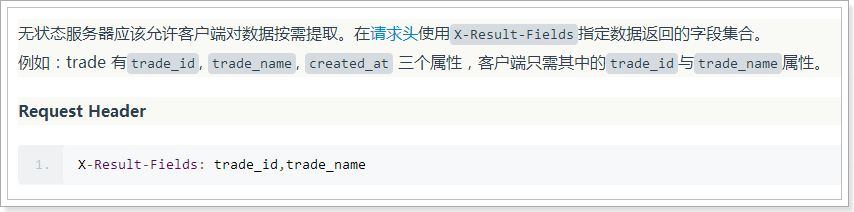
### 最佳实践：REST接口设计

* URL的组成
  + 网络协议（http、https）
  + 服务器地址
  + 接口名称
  + 参数列表
* URL定义限定
  + 不要使用大写字母
  + 使用中线 - 代替下划线 \_
  + 参数列表应该被encode过

### 最佳实践：响应设计

* Content body 仅仅用来传输数据
* 数据要做到拿来就可用的原则，不需要“拆箱”的过程
* 用来描述数据或者请求的元数据放Header中





### http响应状态码



## SpringMVC实现RESTful服务

SpringMVC原生态的支持了REST风格的架构设计。

所涉及到的注解：

@RequestMapping

@PathVariable

@ResponseBody

ResponseEntity

……

# RESTful风格开发商品接口

根据RESTful风格，编写商品功能的增删改查接口

## 编写商品查询接口

在ItemInterfaceController编写

@Controller

@RequestMapping("item/interface")

**public** **class** ItemInterfaceController {

@Autowired

**private** ItemService itemService;

// http://manager.taotao.com/rest/item/interface/{id}

/\*\*

\* 根据id查询用户

\*

\* **@param** id

\* **@return** 返回的类型是ResponseEntity，泛型声明为需要返回的数据类型

\*/

@RequestMapping(value = "{id}", method = RequestMethod.***GET***)

// 返回的是ResponseEntity或者加上@ResponseBody注解的效果是一样的，任选其一即可，也可以都设置。

// @ResponseBody

**public** ResponseEntity<Item> queryItemById(@PathVariable("id") Long id) {

**try** {

Item item = **this**.itemService.queryById(id);

// 查询成功，响应的状态码应为200

// 可以设置HttpStatus枚举的OK

// return ResponseEntity.status(HttpStatus.OK).body(Item);

// 也可以使用ok()方法，效果和上面是一样的

**return** ResponseEntity.*ok*().body(item);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

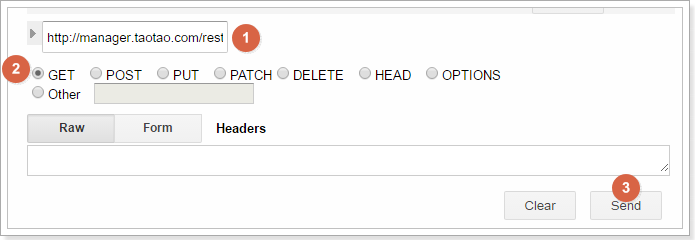
// 如果有异常，设置状态码为500

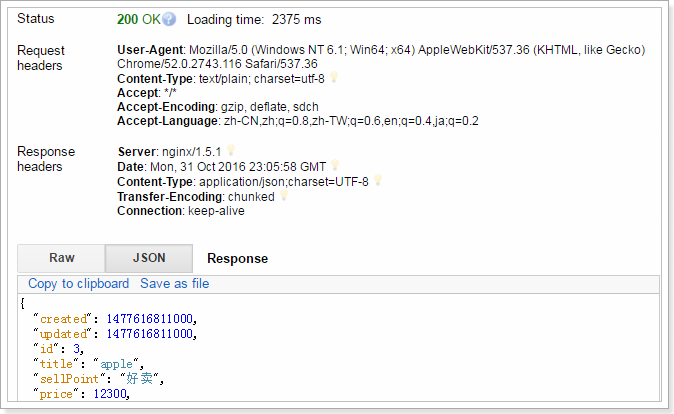
**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

}

使用谷歌浏览器测试工具测试





## 编写商品新增接口

在ItemInterfaceController编写

// http://manager.taotao.com/rest/item/interface

/\*\*

\* 新增

\*

\* **@param** Item

\* **@return**

\*/

@RequestMapping(method = RequestMethod.***POST***)

**public** ResponseEntity<Void> saveItem(Item item) {

**try** {

**this**.itemService.save(item);

// 新增返回201

// 新增成功不需要有返回内容，可以使用build()方法

// return ResponseEntity.status(HttpStatus.CREATED).build();

// 也可以设置body，参数为null即可

**return** ResponseEntity.*status*(HttpStatus.***CREATED***).body(**null**);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

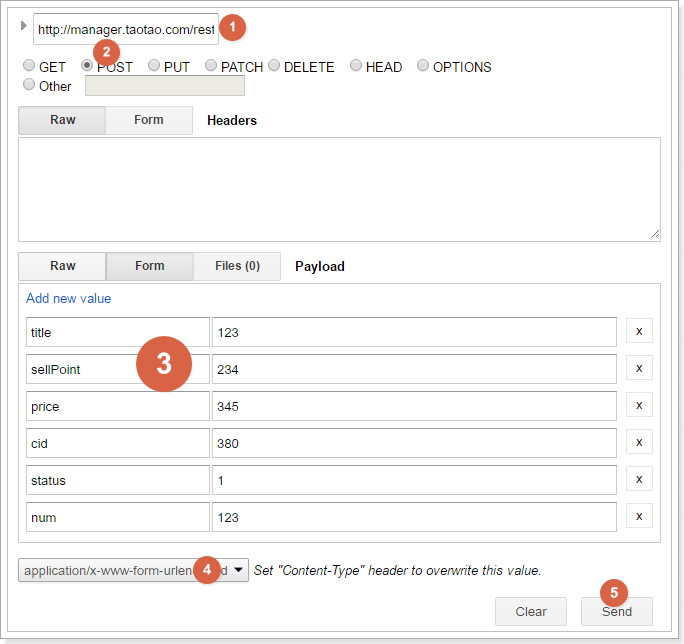
}

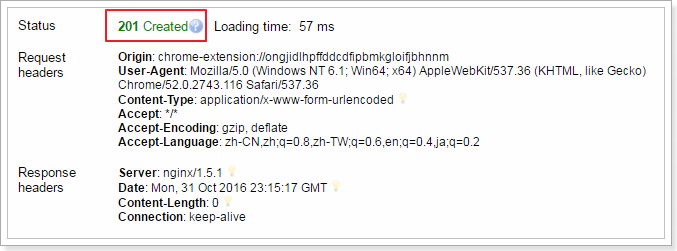
// 如果有异常，设置状态码为500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

使用谷歌浏览器工具测试：





## 编写商品更新接口

在ItemInterfaceController编写

// http://manager.taotao.com/rest/item/interface

/\*\*

\* 更新

\*

\* **@param** Item

\* **@return**

\*/

@RequestMapping(method = RequestMethod.***PUT***)

**public** ResponseEntity<Void> updateItemById(Item item) {

**try** {

**this**.itemService.updateById(item);

// 204

**return** ResponseEntity.*status*(HttpStatus.***NO\_CONTENT***).build();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

// 如果有异常，设置状态码为500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

默认情况下，PUT和DELETE请求是无法提交表单数据的。

解决方案：在web.xml中配置Spring提供的过滤器解决。

<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>

<!-- 将POST请求转化为DELETE或者是PUT

要用\_method指定真正的请求参数 -->

<filter>

<filter-name>HiddenHttpMethodFilter</filter-name>

<filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>

</filter>

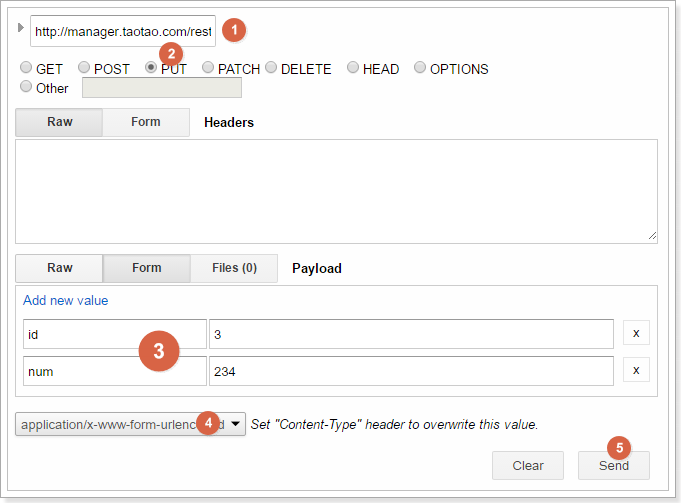
<filter-mapping>

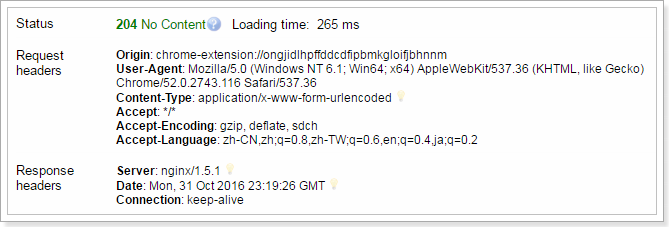
<filter-name>HiddenHttpMethodFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

测试：





## 编写商品删除接口

在ItemInterfaceController编写

// http://manager.taotao.com/rest/item/interface/{id}

/\*\*

\* 删除

\*

\* **@param** id

\* **@return**

\*/

@RequestMapping(value = "{id}", method = RequestMethod.***DELETE***)

**public** ResponseEntity<Void> deleteItemById(@PathVariable("id") Long id) {

**try** {

**this**.itemService.deleteById(id);

// 返回204

**return** ResponseEntity.*status*(HttpStatus.***NO\_CONTENT***).build();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

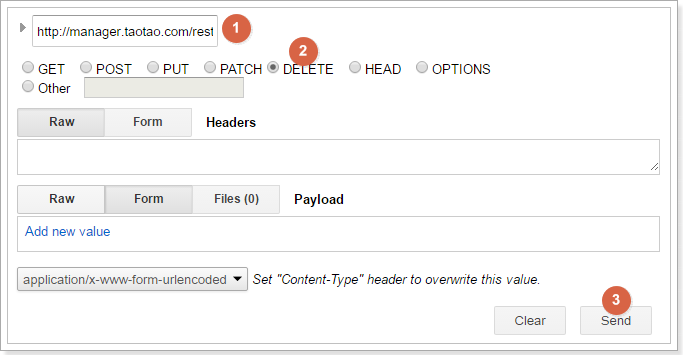
}

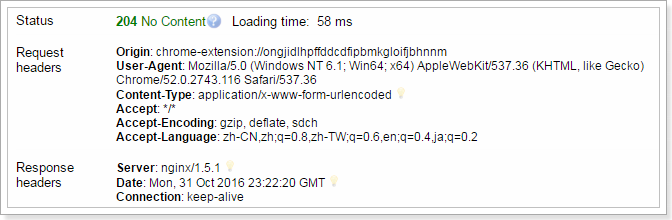
// 如果有异常，设置状态码为500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

测试





# HttpClient

## HttpClient简介

HttpClient 是 Apache Jakarta Common 下的子项目，可以用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。

HttpClient 提供的主要的功能

（1）实现了所有 HTTP 的方法（GET,POST,PUT,DELETE 等）

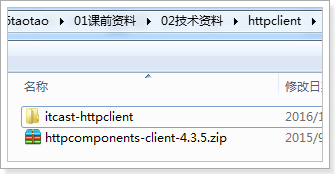
（2）支持自动转向

（3）支持 HTTPS 协议

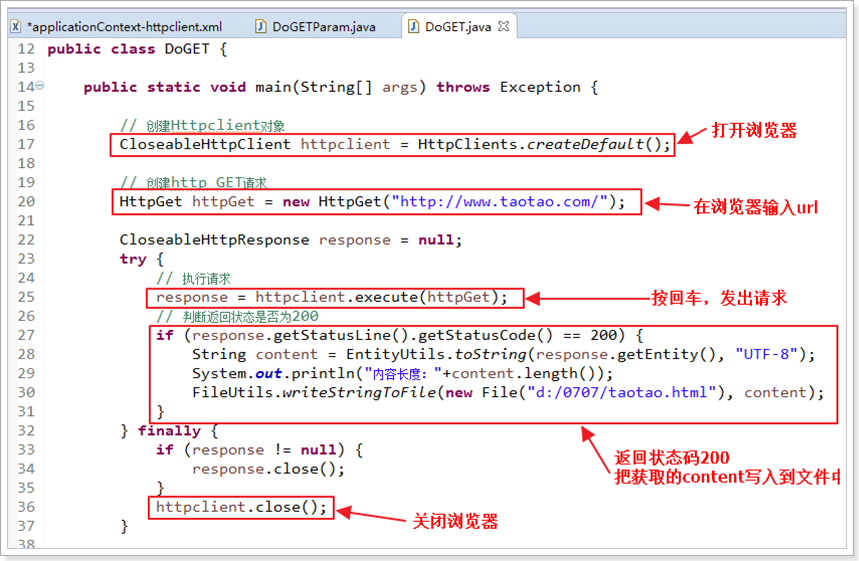
（4）支持代理服务器等

我们所用的solrj就封装了HttpClient

## 学习httpclient



### GET请求



### 带参数的GET请求

**public** **static** **void** main(String[] args) **throws** Exception {

// 创建Httpclient对象

CloseableHttpClient httpclient = HttpClients.*createDefault*();

// http://manager.taotao.com/rest/content?categoryId=32&page=1&rows=20

// 定义请求的参数

// URI uri = new URIBuilder("http://www.baidu.com/s").setParameter("wd",

// "java").build();

URI uri = **new** URIBuilder("http://manager.taotao.com/rest/content").setParameter("categoryId", "32").setParameter("page", "1").setParameter("rows", "20").build();

System.***out***.println(uri);

// 创建http GET请求

HttpGet httpGet = **new** HttpGet(uri);

CloseableHttpResponse response = **null**;

**try** {

// 执行请求

response = httpclient.execute(httpGet);

// 判断返回状态是否为200

**if** (response.getStatusLine().getStatusCode() == 200) {

String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");

System.***out***.println(content);

}

} **finally** {

**if** (response != **null**) {

response.close();

}

httpclient.close();

}

}

### POST请求

**public** **static** **void** main(String[] args) **throws** Exception {

// 创建Httpclient对象

CloseableHttpClient httpclient = HttpClients.*createDefault*();

// 创建http POST请求

HttpPost httpPost = **new** HttpPost("http://www.oschina.net/");

// 在请求中设置请求头,设置请求头，跳过开源中国的访问限制

httpPost.setHeader("User-Agent", "");

CloseableHttpResponse response = **null**;

**try** {

// 执行请求

response = httpclient.execute(httpPost);

// 判断返回状态是否为200

**if** (response.getStatusLine().getStatusCode() == 200) {

String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");

System.***out***.println(content);

}

} **finally** {

**if** (response != **null**) {

response.close();

}

httpclient.close();

}

}

### 带参数的POST请求

**public** **static** **void** main(String[] args) **throws** Exception {

// 创建Httpclient对象

CloseableHttpClient httpclient = HttpClients.*createDefault*();

// 创建http POST请求

HttpPost httpPost = **new** HttpPost("http://manager.taotao.com/rest/item/interface");

// 设置2个post参数，一个是scope、一个是q

List<NameValuePair> parameters = **new** ArrayList<NameValuePair>(0);

// parameters.add(new BasicNameValuePair("scope", "project"));

parameters.add(**new** BasicNameValuePair("price", "123000"));

parameters.add(**new** BasicNameValuePair("title", "httpclient123"));

parameters.add(**new** BasicNameValuePair("cid", "380"));

parameters.add(**new** BasicNameValuePair("status", "1"));

parameters.add(**new** BasicNameValuePair("num", "123"));

// 构造一个form表单式的实体

UrlEncodedFormEntity formEntity = **new** UrlEncodedFormEntity(parameters, "UTF-8");

// 将请求实体设置到httpPost对象中

httpPost.setEntity(formEntity);

CloseableHttpResponse response = **null**;

**try** {

// 执行请求

response = httpclient.execute(httpPost);

// 判断返回状态是否为200

**if** (response.getStatusLine().getStatusCode() == 201) {

String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");

System.***out***.println(content);

}

} **finally** {

**if** (response != **null**) {

response.close();

}

httpclient.close();

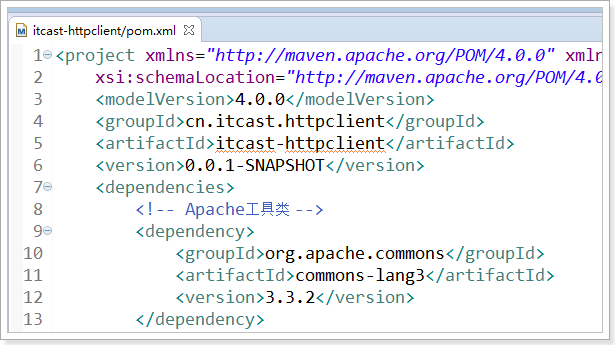
}

}

# 使用HttpClient调用接口

## 加入依赖

Itcast-httpclient加入spring依赖



## 封装HttpClient

**public** **class** APIService {

**private** CloseableHttpClient httpClient;

**public** APIService() {

**super**();

**this**.httpClient = HttpClients.*createDefault*();

}

// 1. 带参数的get请求

/\*\*

\* 带参数的get请求

\*

\* **@param** url

\* **@param** map

\* **@return** 如果响应的状态码是200，则返回响应体的内容。如果不是200，则返回null

\* **@throws** Exception

\*/

**public** String doGet(String url, Map<String, Object> map) **throws** Exception {

// 声明URIBuilder，设置请求url

URIBuilder uriBuilder = **new** URIBuilder(url);

// 判断map不为空

**if** (map != **null**) {

// 遍历map，设置参数

**for** (Map.Entry<String, Object> entry : map.entrySet()) {

uriBuilder.setParameter(entry.getKey(), entry.getValue().toString());

}

}

URI uri = uriBuilder.build();

// 声明httpget请求

HttpGet httpGet = **new** HttpGet(uri);

// 发起请求

CloseableHttpResponse response = **this**.httpClient.execute(httpGet);

// 解析response。如果返回的状态码是200，则返回响应体的内容，如果不是200，则返回null

**if** (response.getStatusLine().getStatusCode() == 200) {

// 获取响应体的内容

String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");

**return** content;

}

**return** **null**;

}

// 2. 不带参数的get请求

/\*\*

\* 不带参数的get请求

\*

\* **@param** url

\* **@return** 如果响应的状态码是200，则返回响应体的内容。如果不是200，则返回null

\* **@throws** Exception

\*/

**public** String doGet(String url) **throws** Exception {

**return** **this**.doGet(url, **null**);

}

// 3. 带参数的post请求

/\*\*

\* 带参数的post请求

\*

\* **@param** url

\* **@param** map

\* **@return**

\* **@throws** Exception

\*/

**public** HttpResult doPost(String url, Map<String, Object> map) **throws** Exception {

// 声明httpPost请求

HttpPost httpPost = **new** HttpPost(url);

// 判断参数map不为空

**if** (map != **null**) {

// 声明list存放参数

List<NameValuePair> param = **new** ArrayList<NameValuePair>();

// 遍历map，设置参数

**for** (Map.Entry<String, Object> entry : map.entrySet()) {

param.add(**new** BasicNameValuePair(entry.getKey(), entry.getValue().toString()));

}

// 声明from表单Entity

UrlEncodedFormEntity entity = **new** UrlEncodedFormEntity(param, "UTF-8");

// 把Entity设置到post请求中

httpPost.setEntity(entity);

}

CloseableHttpResponse response = **this**.httpClient.execute(httpPost);

// 判断response的Entity是否为null如果为null就不能使用EntityUtils解析，否则会报错

**if** (response.getEntity() == **null**) {

**return** **new** HttpResult(response.getStatusLine().getStatusCode(), **null**);

} **else** {

String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");

**return** **new** HttpResult(response.getStatusLine().getStatusCode(), content);

}

}

// 4. 不带参数的post请求

/\*\*

\* 不带参数的post请求

\*

\* **@param** url

\* **@return**

\* **@throws** Exception

\*/

**public** HttpResult doPost(String url) **throws** Exception {

**return** **this**.doPost(url, **null**);

}

// 5. 带参数的put请求

/\*\*

\* 带参数的put请求

\*

\* **@param** url

\* **@param** map

\* **@return**

\* **@throws** Exception

\*/

**public** HttpResult doPut(String url, Map<String, Object> map) **throws** Exception {

// 声明httpPost请求

HttpPut httpPut = **new** HttpPut(url);

// 判断参数map不为空

**if** (map != **null**) {

// 声明list存放参数

List<NameValuePair> param = **new** ArrayList<NameValuePair>();

// 遍历map，设置参数

**for** (Map.Entry<String, Object> entry : map.entrySet()) {

param.add(**new** BasicNameValuePair(entry.getKey(), entry.getValue().toString()));

}

// 声明from表单Entity

UrlEncodedFormEntity entity = **new** UrlEncodedFormEntity(param, "UTF-8");

// 把Entity设置到post请求中

httpPut.setEntity(entity);

}

CloseableHttpResponse response = **this**.httpClient.execute(httpPut);

// 判断response的Entity是否为null如果为null就不能使用EntityUtils解析，否则会报错

**if** (response.getEntity() == **null**) {

**return** **new** HttpResult(response.getStatusLine().getStatusCode(), **null**);

} **else** {

String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");

**return** **new** HttpResult(response.getStatusLine().getStatusCode(), content);

}

}

// 6. 带参数的delete

**public** HttpResult doDelete(String url, Map<String, Object> map) **throws** Exception {

// 声明URIBulider

URIBuilder uriBuilder = **new** URIBuilder(url);

**if** (map != **null**) {

// 设置请求参数

**for** (Map.Entry<String, Object> entry : map.entrySet()) {

uriBuilder.setParameter(entry.getKey(), entry.getValue().toString());

}

}

// 声明httpDelete请求

URI uri = uriBuilder.build();

HttpDelete httpDelete = **new** HttpDelete(uri);

// 发起请求

CloseableHttpResponse response = **this**.httpClient.execute(httpDelete);

// 判断response响应体是否为null

**if** (response.getEntity() == **null**) {

**return** **new** HttpResult(response.getStatusLine().getStatusCode(), **null**);

} **else** {

String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");

**return** **new** HttpResult(response.getStatusLine().getStatusCode(), content);

}

}

}

## 使用APIService调用接口

**public** **class** APIServiceTest {

**private** APIService apiService;

@Before

**public** **void** init() {

**this**.apiService = **new** APIService();

}

// 测试根据id查询商品

@Test

**public** **void** testQueryItem() **throws** Exception {

// http://manager.taotao.com/rest/item/interface/44

String url = "http://manager.taotao.com/rest/item/interface/44";

String content = **this**.apiService.doGet(url);

System.***out***.println(content);

}

// 新增商品

@Test

**public** **void** testSaveItem() **throws** Exception {

// http://manager.taotao.com/rest/item/interface

String url = "http://manager.taotao.com/rest/item/interface";

// title=RESTful Web Service&price=100&cid=1&status=1&num=1

Map<String, Object> map = **new** HashMap<String, Object>();

map.put("title", "Web Service");

map.put("price", 100);

map.put("cid", 1);

map.put("status", 1);

map.put("num", 1);

HttpResult httpResult = **this**.apiService.doPost(url, map);

System.***out***.println(httpResult.getCode());

System.***out***.println(httpResult.getContent());

}

// 更新商品

@Test

**public** **void** testUpdateItem() **throws** Exception {

// http://manager.taotao.com/rest/item/interface

String url = "http://manager.taotao.com/rest/item/interface";

// title=RESTful Web Service&price=100&cid=1&status=1&num=1

Map<String, Object> map = **new** HashMap<String, Object>();

map.put("id", 47);

map.put("title", "机器猫");

HttpResult httpResult = **this**.apiService.doPut(url, map);

System.***out***.println(httpResult.getCode());

System.***out***.println(httpResult.getContent());

}

// 删除商品

@Test

**public** **void** testDeleteItem() **throws** Exception {

// http://manager.taotao.com/rest/item/interface/47

String url = "http://manager.taotao.com/rest/item/interface/47";

HttpResult httpResult = **this**.apiService.doDelete(url, **null**);

System.***out***.println(httpResult.getCode());

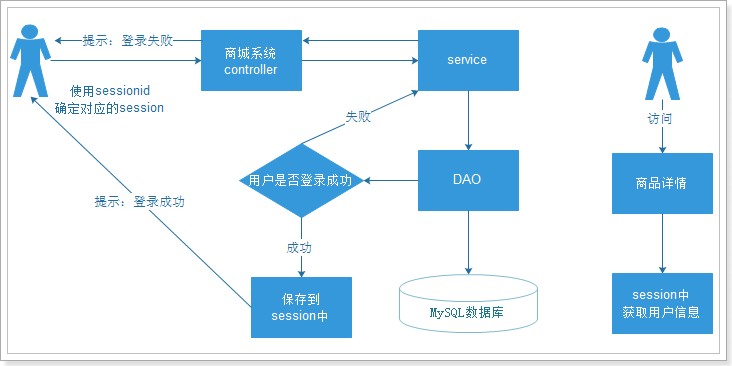
System.***out***.println(httpResult.getContent());

}

}

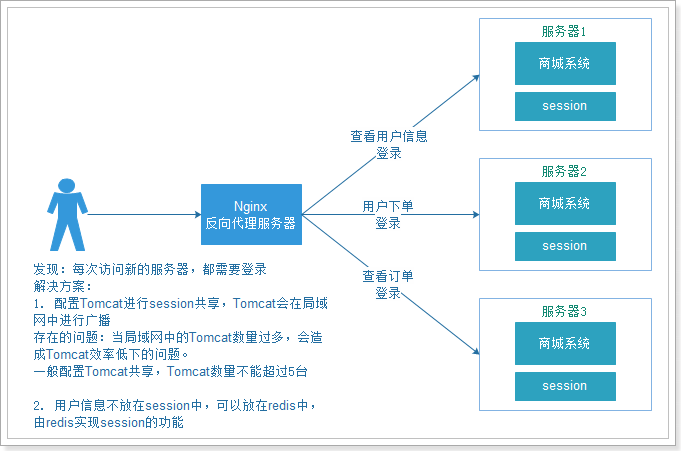
# 单点登录

## 以前的登录流程

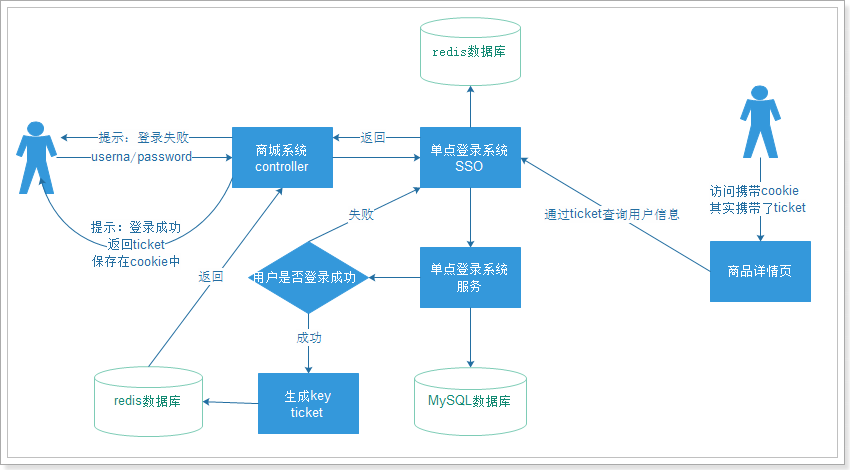


以前的登录流程：用户信息保存在session

## 集群登录分析



## 现在的登录流程



（两个redis其实是一个数据库）

使用redis实现session的功能

# 搭建SSO系统

## sso简介

SSO （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。它是目前比较流行的企业业务整合的解决方案之一。

## 架构分析

表现层：提供手机客户端，或其他系统的调用

服务层：系统内的调用

需要搭建服务层和表现层工程

服务层是聚合工程

taotao-sso 聚合工程父工程

taotao-sso-interface 子工程，提供接口，依赖

taotao-sso-service 子工程，业务处理

表现层

taotao-sso-web

服务层没有pojo和mapper，依赖后台系统的pojo和mapper

## 使用的技术

和前面搭建的工程类似

服务层需要的技术

spring+mybatis

redis

分页助手，工具类

dubbo

表现层需要的技术：

springMVC

Jackson，工具类，等

dubbo

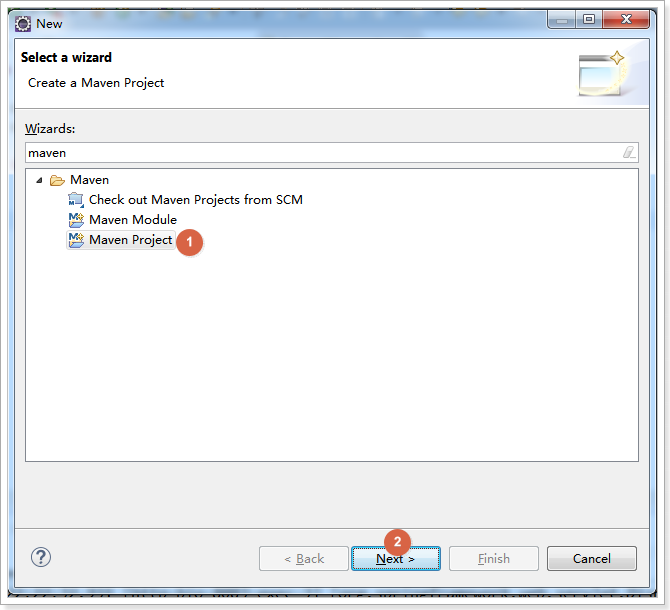
## 搭建taotao-sso

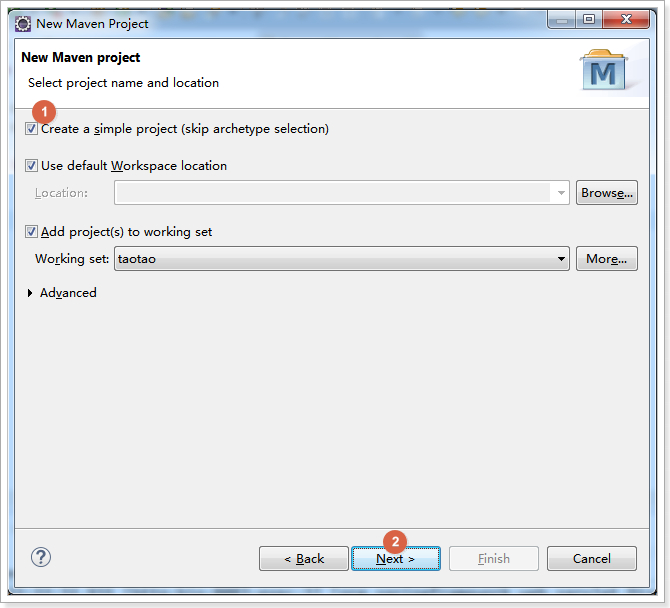
taotao-sso是服务层，类似taotao-content

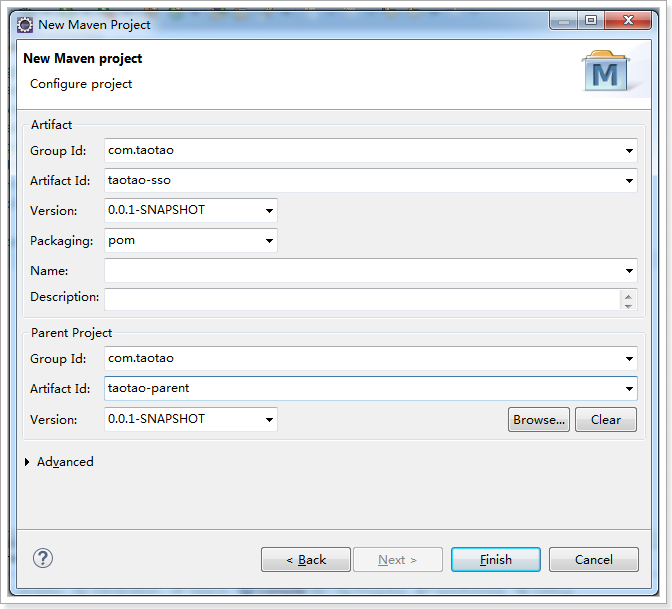
### 创建工程

#### 创建taotoa-sso

创建聚合工程父工程

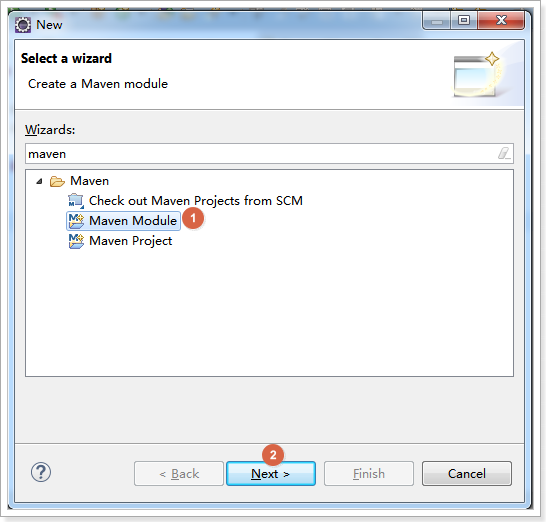


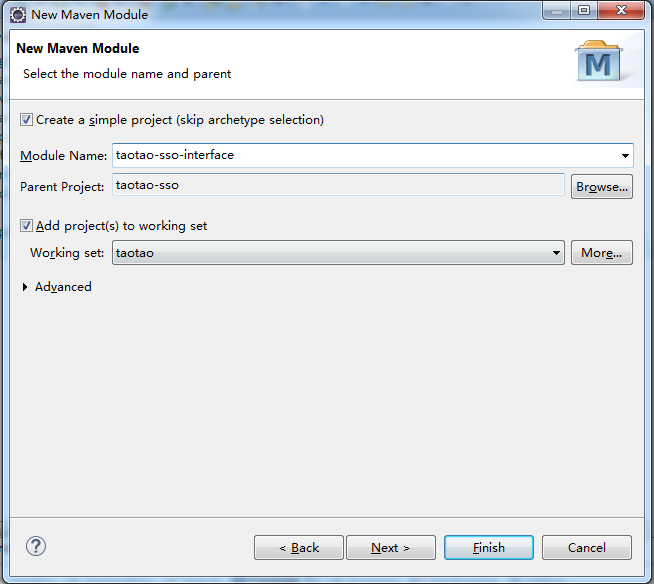


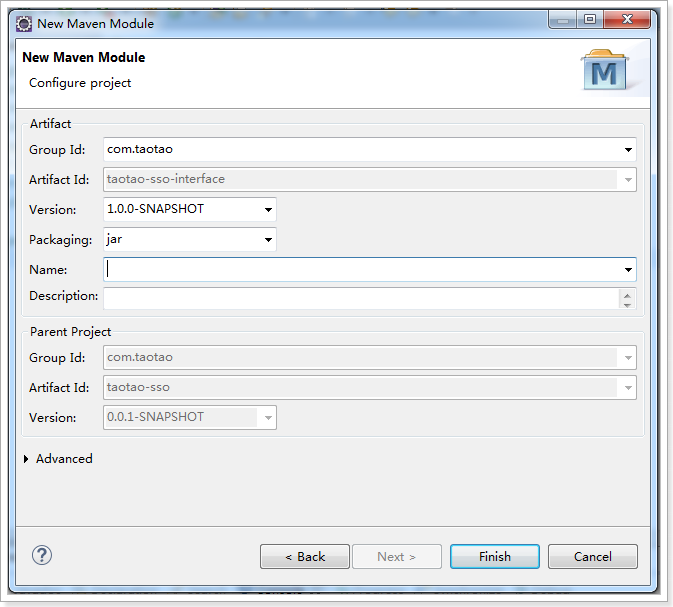


#### 创建taotao-sso-interface

创建聚合工程子工程

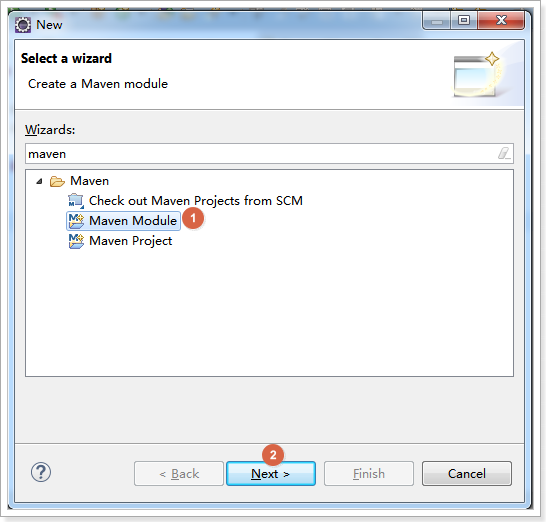


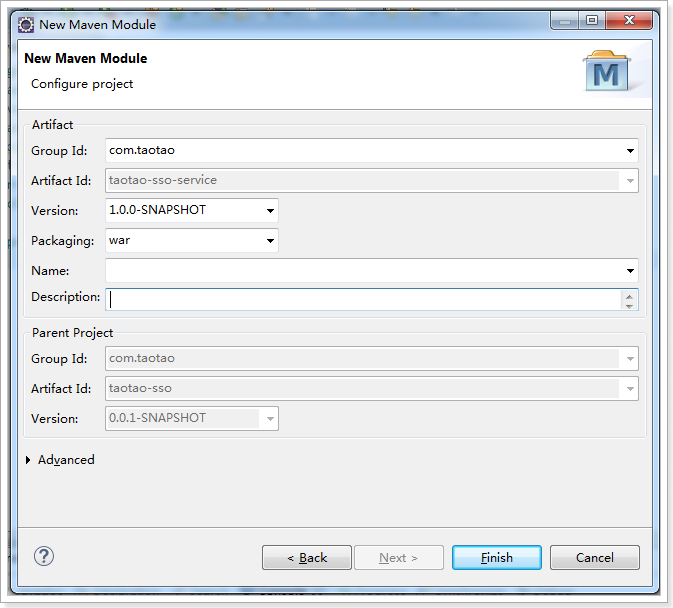




#### 创建taotao-sso-service

创建聚合工程子工程





### 加入依赖

taotao-sso加入依赖

<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.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-sso</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<modules>

<module>taotao-sso-interface</module>

<module>taotao-sso-service</module>

</modules>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- Apache通用工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

</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>

taotao-sso-interface

<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.taotao</groupId>

<artifactId>taotao-sso</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>taotao-sso-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

<dependencies>

<!-- 加入taotao-manager-pojo依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-manager-pojo</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

</project>

taotao-sso-service

<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.taotao</groupId>

<artifactId>taotao-sso</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>taotao-sso-service</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<!-- 加入taotao-manager-mapper依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-manager-mapper</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- 加入taotao-sso-interface依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-sso-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- Spring -->

<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>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<!-- 分页助手 -->

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

</dependency>

<dependency>

<groupId>com.github.jsqlparser</groupId>

<artifactId>jsqlparser</artifactId>

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- dubbo相关 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

</dependency>

<!-- Redis客户端 -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

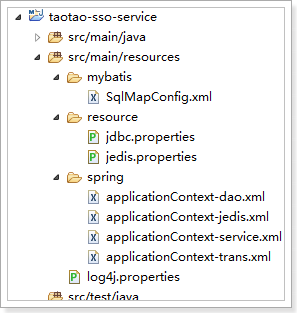
</dependency>

</dependencies>

</project>

### 加入配置文件

配置文件结构



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"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"MyWebApp"* version=*"2.5"*>

<display-name>taotao-sso</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<!-- spring -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/applicationContext\*.xml</param-value>

</context-param>

<!-- spring监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

</web-app>

applicationContext-dao.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"*>

<!-- 配置 读取properties文件 jdbc.properties -->

<context:property-placeholder location=*"classpath:\*.properties"* />

<!-- 配置数据源 -->

<bean id=*"dataSource"* class=*"com.jolbox.bonecp.BoneCPDataSource"*

destroy-method=*"close"*>

<!-- 数据库驱动 -->

<property name=*"driverClass"* value=*"${jdbc.driver}"* />

<!-- 相应驱动的jdbcUrl -->

<property name=*"jdbcUrl"* value=*"${jdbc.url}"* />

<!-- 数据库的用户名 -->

<property name=*"username"* value=*"${jdbc.username}"* />

<!-- 数据库的密码 -->

<property name=*"password"* value=*"${jdbc.password}"* />

<!-- 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：240，如果要取消则设置为0 -->

<property name=*"idleConnectionTestPeriod"* value=*"60"* />

<!-- 连接池中未使用的链接最大存活时间，单位是分，默认值：60，如果要永远存活设置为0 -->

<property name=*"idleMaxAge"* value=*"30"* />

<!-- 每个分区最大的连接数 -->

<property name=*"maxConnectionsPerPartition"* value=*"150"* />

<!-- 每个分区最小的连接数 -->

<property name=*"minConnectionsPerPartition"* value=*"5"* />

</bean>

<!-- 配置SqlSession会话工厂 -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 配置数据源 -->

<property name=*"dataSource"* ref=*"dataSource"* />

<!-- 配置mybatis全局配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"* />

</bean>

<!-- 配置Mapper扫描 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 配置扫描的包 -->

<property name=*"basePackage"* value=*"com.taotao.manager.mapper"* />

</bean>

</beans>

applicationContext-jedis.xml

<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"*>

<!-- 配置jedisPool -->

<bean class=*"redis.clients.jedis.JedisPool"*>

<constructor-arg name=*"host"* value=*"${jedis.host}"* />

<constructor-arg name=*"port"* value=*"${jedis.port}"* />

</bean>

<!-- 配置jedisCluster -->

<bean class=*"redis.clients.jedis.JedisCluster"*>

<!-- 配置的集群节点信息 -->

<constructor-arg name=*"nodes"*>

<set>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host1}"* />

<constructor-arg name=*"port"* value=*"${cluster.port1}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host2}"* />

<constructor-arg name=*"port"* value=*"${cluster.port2}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host3}"* />

<constructor-arg name=*"port"* value=*"${cluster.port3}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host4}"* />

<constructor-arg name=*"port"* value=*"${cluster.port4}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host5}"* />

<constructor-arg name=*"port"* value=*"${cluster.port5}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host6}"* />

<constructor-arg name=*"port"* value=*"${cluster.port6}"* />

</bean>

</set>

</constructor-arg>

</bean>

<!-- 配置redis单机版实现 -->

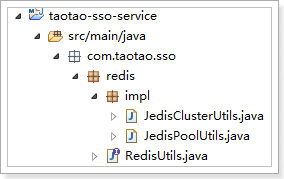
<bean id=*"jedisPoolUtils"* class=*"com.taotao.sso.redis.impl.JedisPoolUtils"* />

<!-- 配置redis集群版实现 -->

<!-- <bean id="jedisClusterUtils" class="com.taotao.sso.redis.impl.JedisClusterUtils" /> -->

</beans>

需要把RedisUtils也复制过来



applicationContext-service.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"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.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"*>

<!-- 配置service扫描 -->

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

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"taotao-sso-service"* />

<!-- 使用multicast广播注册中心暴露服务地址 -->

<!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->

<!-- <dubbo:registry address="N/A" /> -->

<dubbo:registry protocol=*"zookeeper"* address=*"192.168.37.161:2181"* />

<!-- 用dubbo协议在20880端口暴露服务 -->

<dubbo:protocol name=*"dubbo"* port=*"20881"* />

<!-- 声明需要暴露的服务接口 -->

<!-- <dubbo:service interface="com.taotao.content.service.ContentCategoryService" -->

<!-- ref="contentCategoryServiceImpl" /> -->

</beans>

applicationContext-trans.xml

<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"*>

<!-- 定义事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

<!-- 定义事务策略 -->

<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:method name=*"query\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

</tx:attributes>

</tx:advice>

<aop:config>

<!--pointcut元素定义一个切入点，execution中的第一个星号 用以匹配方法的返回类型，

这里星号表明匹配所有返回类型。 com.abc.dao.\*.\*(..)表明匹配cn.itcast.mybatis.service包下的所有类的所有

方法 -->

<aop:pointcut id=*"myPointcut"* expression=*"execution(\* com.taotao.sso.service.\*.\*(..))"* />

<!--将定义好的事务处理策略应用到上述的切入点 -->

<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"myPointcut"* />

</aop:config>

</beans>

redis.properties

#配置redis单机版的参数

jedis.host=192.168.37.161

jedis.port=6379

#配置redis集群版的参数

cluster.host1=192.168.37.161

cluster.port1=7001

cluster.host2=192.168.37.161

cluster.port2=7002

cluster.host3=192.168.37.161

cluster.port3=7003

cluster.host4=192.168.37.161

cluster.port4=7004

cluster.host5=192.168.37.161

cluster.port5=7005

cluster.host6=192.168.37.161

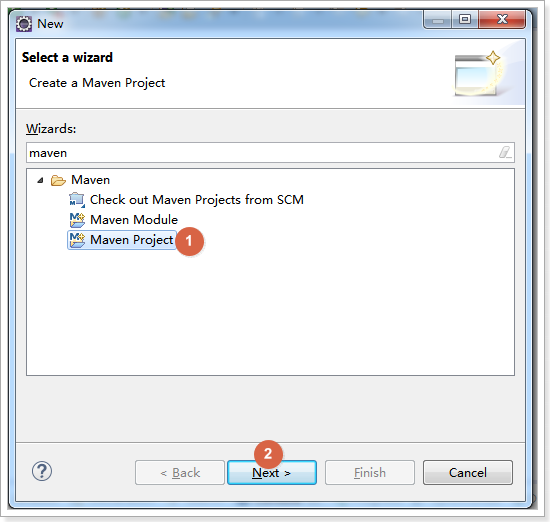
cluster.port6=7006

## 搭建taotao-sso-web

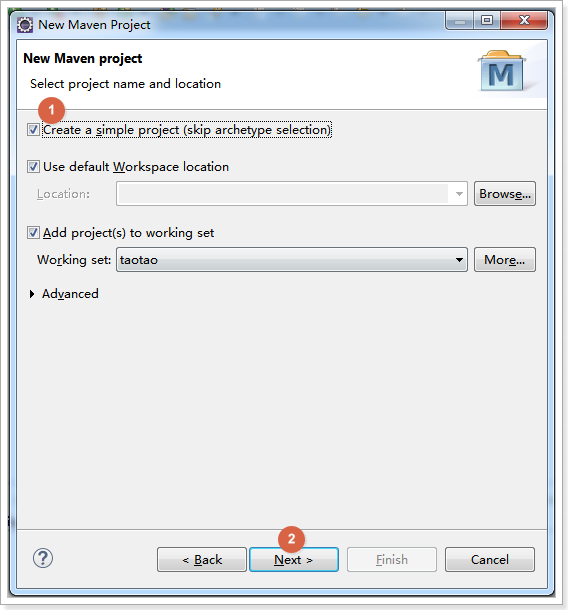
taotao-sso-web是表现层，类似taotao-portal

### 创建taotao-sso-web

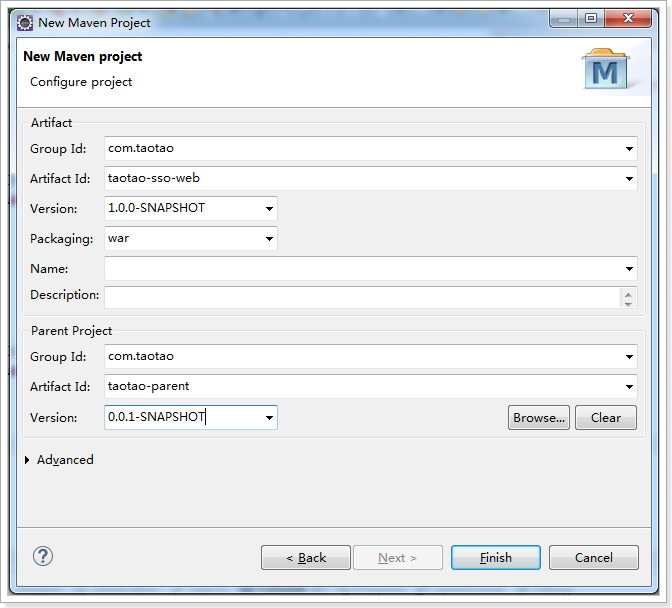
创建Maven工程



创建简单工程



填写创建信息，需要打war包



### 添加依赖

<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.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-sso-web</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<!-- 加入taotao-sso-interface依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-sso-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- Spring -->

<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>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- Apache通用工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

<!-- dubbo相关 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

</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>

### 加入配置文件

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"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"MyWebApp"* version=*"2.5"*>

<display-name>taotao-sso-web</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<!-- 配置解决post提交乱码的问题 -->

<filter>

<filter-name>encoding</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>encoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 配置springMVC -->

<servlet>

<servlet-name>taotao-sso-web</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 设置springMVC配置文件 -->

<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>

<!-- 配置springMVC映射 -->

<servlet-mapping>

<servlet-name>taotao-sso-web</servlet-name>

<!-- 所有的请求进入springMVC -->

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

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"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.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"*>

<!-- 加入properties配置文件 -->

<context:property-placeholder location=*"classpath:resource/\*.properties"* />

<!-- 配置注解驱动 -->

<mvc:annotation-driven />

<!-- 配置controller扫描 -->

<context:component-scan base-package=*"com.taotao.sso.controller"* />

<!-- 配置dubbo服务 -->

<dubbo:application name=*"taotao-sso-web"* />

<!-- 使用广播 -->

<!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->

<dubbo:registry protocol=*"zookeeper"* address=*"192.168.37.161:2181"* />

<!-- 声明要调用的服务，timeout是设置连接超时最长时间,如果不设置，超时时间默认是3秒 -->

<!-- <dubbo:reference interface="com.taotao.content.service.ContentService" -->

<!-- id="contentService" timeout="1000000" /> -->

</beans>