课程大纲

2018年7月6日 17:23

|  |  |
| --- | --- |
| springboot | springboot基础，快速搭建，springboot组合注解运行原  理，springboot的web开发，spring data JPA，mybatis整 合，springboot发布 |
| redis | redis简介，雪崩/缓存击穿，  hash、string、list、set、zset，jedis数据分片，jedis数据 分片连接池，jedis的hash一致性，redis主从复制，哨兵集  群，redis-cluster，redis集群的槽道 |
| mycat | mycat简介，mysql主从复制，mycat读写分离，数据库双机 热备，mycat范围分布式配置，mycat配置详解, |
| rabbitmq | rabbimq概括，connection，channle，exchange，消息生产者和消费者，消息传递simple模式，work模式，publish  模式，routing模式，topic模式，秒杀案例 |
| lucene | lucene概述，分词系统，索引(index)概念，倒排索引，索引创建，索引删除，索引更新，词项搜索，布尔搜索，范围搜  索，前缀搜索，多关键字搜索，模糊搜索，通配符搜索 |
| elasticsearch | 搜索系统概括，搜索引擎框架，安装与配置详解，  elasticseacher安装与配置详解，分不是集群搭建 |
| nginx | http服务器，反向代理，入门配置，server配置，location匹 配，负载均衡轮询、权重、session黏着。hash取余 |

京淘项目 京淘项目是一个集合了分布式技术的大型电商项目，可承受

高并发，具有高可用的特点

后台商品管理：负责商品的增删改查，图片上传，商品分类 树

前台首页：商品前台分类树，商品查看，登录，购物车，订 单页面跳转

登录页面：负责用户登录，解决session集群共享 购物车：负责购物车商品管理

订单：负责订单提交，定时任务的处理

分区 day01all的第 1页

订单：负责订单提交，定时任务的处理

|  |  |
| --- | --- |
| springcloud微  服务整合 | 微服务概括  springcloud-eureka:服务治理组件，可以管理微服务群体的 所有服务调度  springcloud-ribbon:负载均衡组件，服务调度中高可用，轮 询访问  springcloud-hystrix:断熔器，异常错误等处理的组件  springcloud-zuul:网关组件，可以用于服务调度的路由转发 和过滤等作用  springcloud-feign:基于接口调用rest的组件  springcloud微服务框架整改京淘项目 |

分区 day01all的第 2页

环境准备

2018年7月8日 22:43

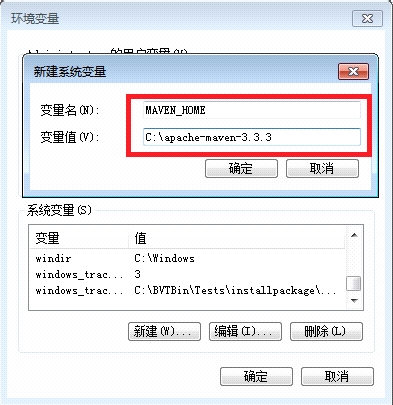
使用资料,maven库

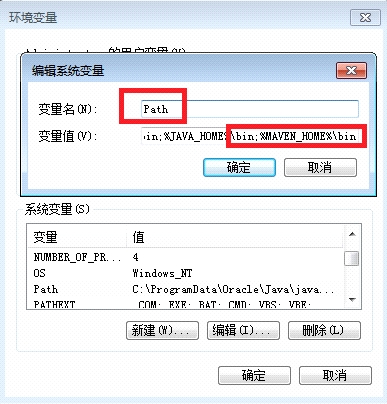
jt\_repo

maven环境变量(可以不配置)

保证当前的window具备maven运行各种命令的环境

与配置jdk类似，需要配置一个MAVEN\_HOME,然后在path 中添加对应的bin

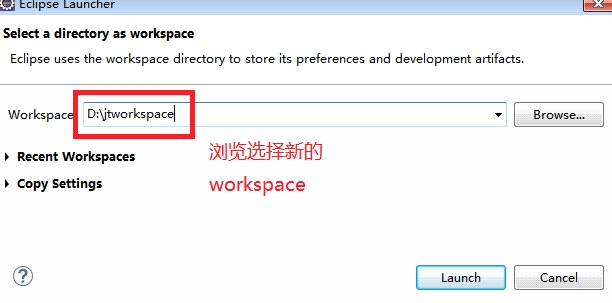
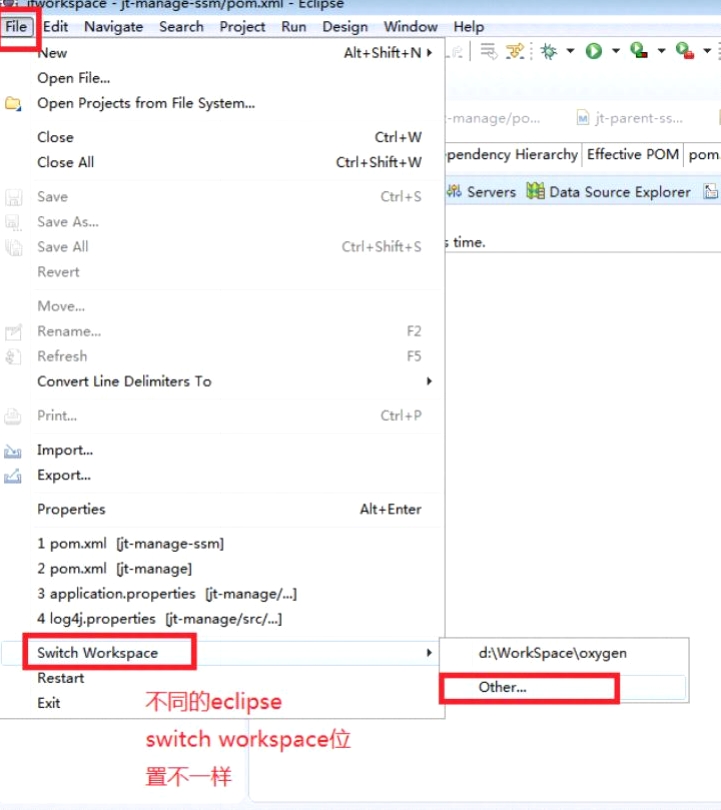
分区 day01all的第 4页

maven库和workspace

• eclipse的workspace替换（如果是新环境忽略）

为了不影响前面工程的内容，我们需要替换workspace 重新配置当前workspace的配置内容

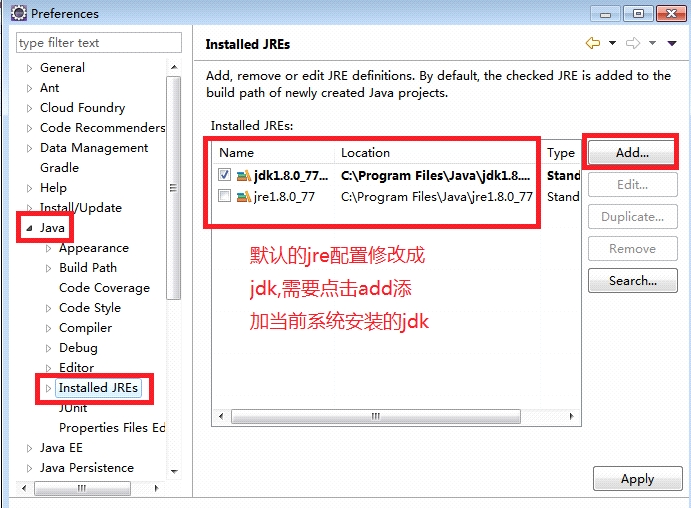
分区 day01all的第 5页

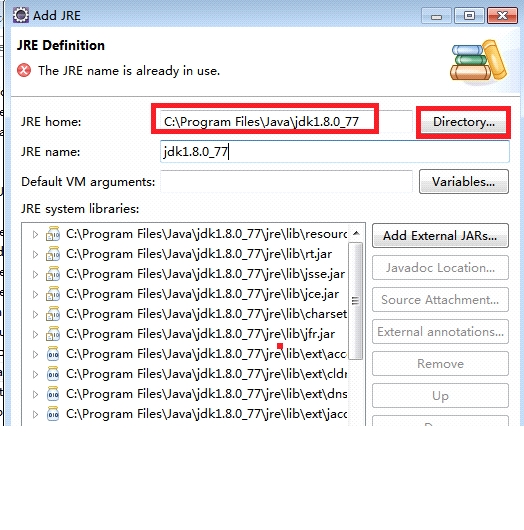
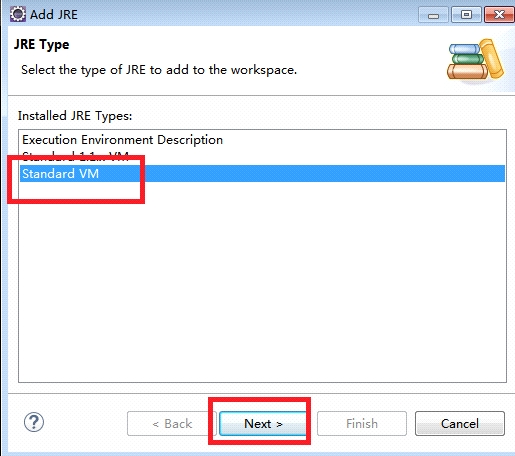
分区 day01all的第 6页

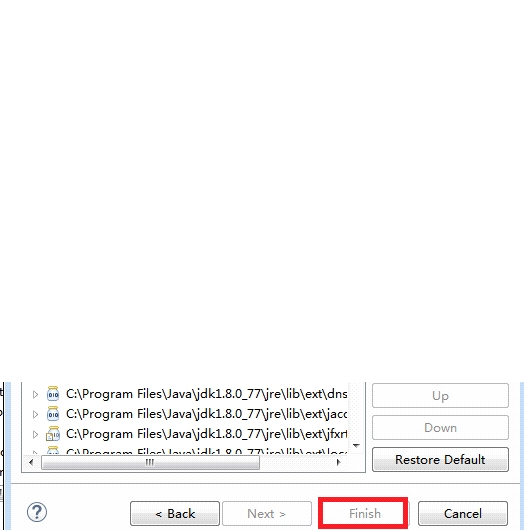
• eclispe的jdk绑定

默认的eclipse使用的jre，maven的install会出现问题， 所以我们将eclipse中的jre替换成jdk

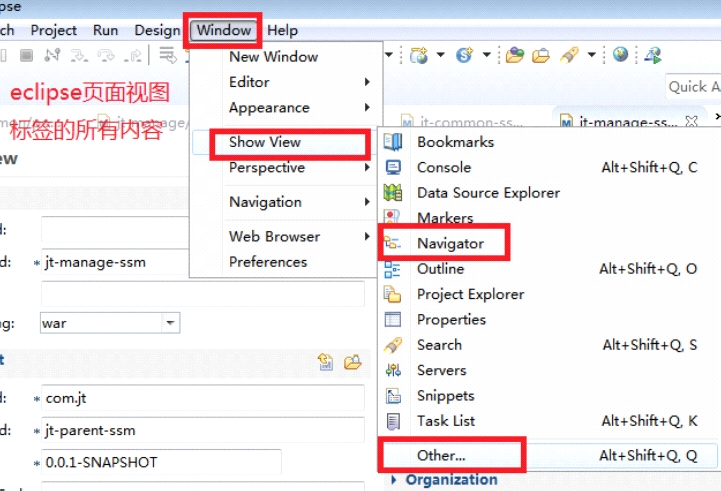
如果前期使用jdk1.7，替换成1.8,可以使用课前资料环 境-widows--jdk-8u\*中jdk1.8的包安装

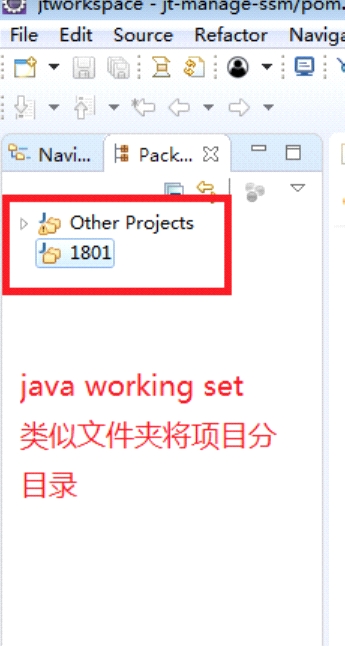
分区 day01all的第 7页

分区 day01all的第 8页

• eclipse的显示标签

navigator和package exlorer

分区 day01all的第 9页

navigator保存了当前项目的所有目录结构，与在磁盘上 直接操作一样

package explorer保存了项目的代码结构，方便管理项

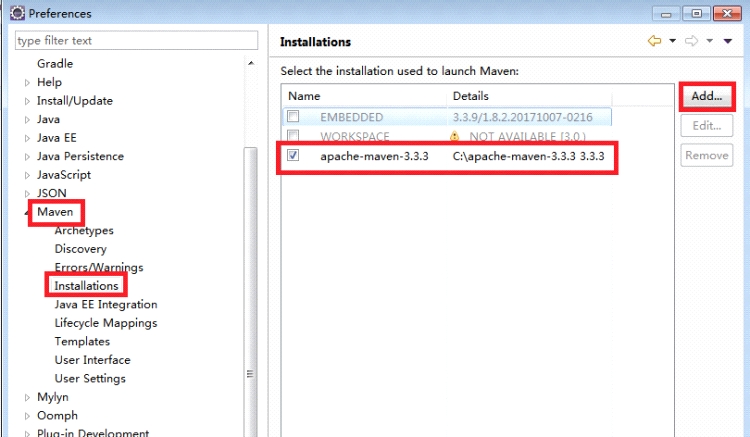
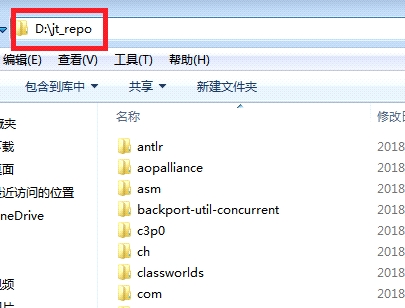
目

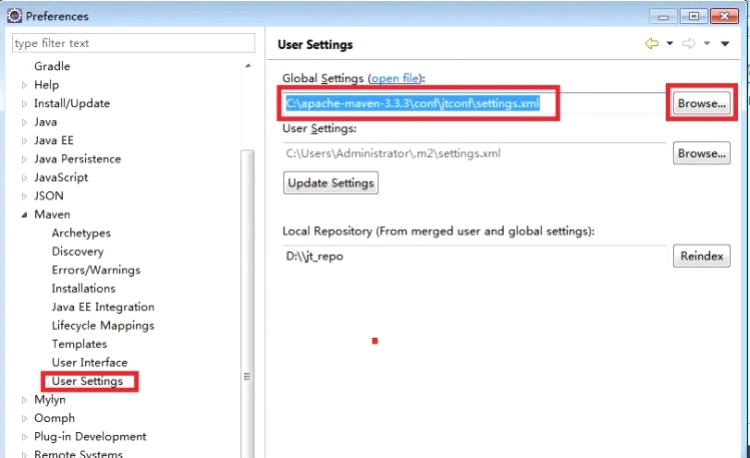
这是我的习惯显示标签，可以不一样。

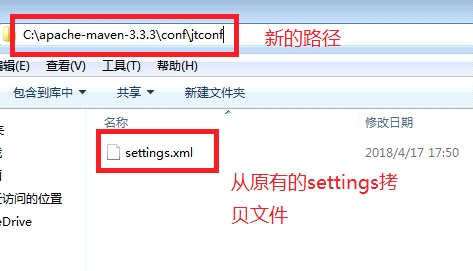
• **maven库绑定(maven要求必须3.3以上)**

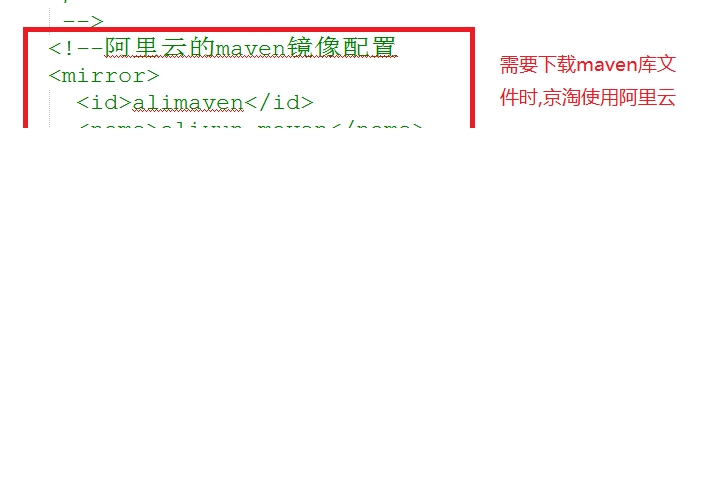
将资料中的maven库解压到无中文路径无空格的文件夹 下

分区 day01all的第 10页

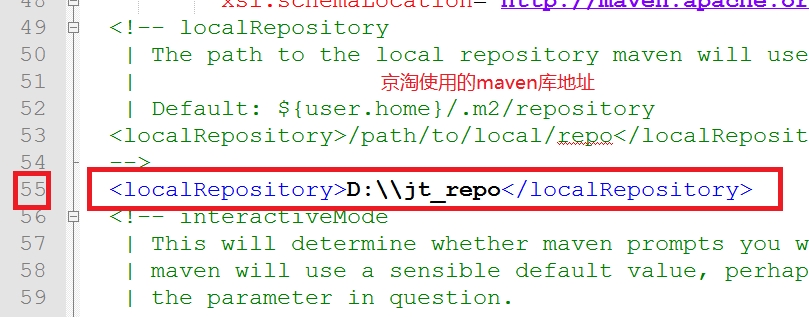
分区 day01all的第 11页

将新的配置文件做到一个新的路径中，与前面不冲突

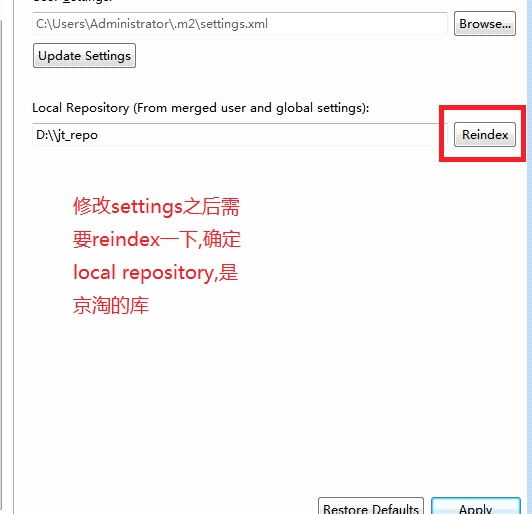
修改镜像，建议阿里云，达内镜像我没有试过(可换可不 换)

分区 day01all的第 12页

修改一下maven库路径

将新绑定的setting文件跟新，就看到指向的maven库是 我们自己的

分区 day01all的第 13页

分区 day01all的第 14页

springboot概述

2018年9月10日 10:09

作业:使用springboot整改前面的项目

什么是springboot

随着动态语言的发展

(node.js,groovy,scala),java需要一个独立运行, 准生产级别的给予Spring框架的快速搭建工程

的工具---springboot.

动态语言不需要频繁的发布部署等等等繁琐 的过程.

Springboot的核心功能

•

•

•

独立运行的spring项目

Springboot 可以用jar包的形式独立运行 (web应用,运行是war包).

内嵌的servlet容器(spring+springmvc独立运 行)

默认内嵌servlet容器是tomcat,可以选择其

他的,undertow jetty.可以在包装工程时, 将内嵌容器一起包装,运行时就不需要以

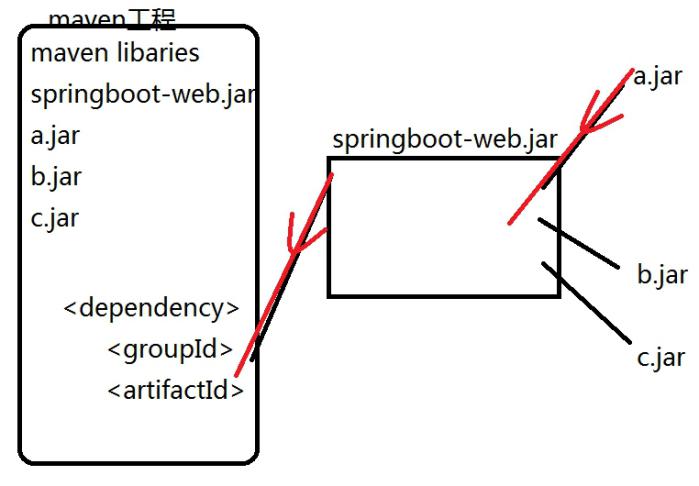
来外部的tomcat(或其他)来部署.

只需要在jdk环境即可运行.

提供简化的starter依赖配置

maven依赖的特点:依赖传递.

分区 day01all 的第 15页

Starter pom简化maven以来,利用maven 的以来传递性,所有复杂的jar包以来,在工 程中都是只需要以来一个上层的

dependency即可.例如,web应用,以来

spring-boot-starter-web,可以看到众多的传递过来的jar包.

• 自动配置

springboot对外宣称的搭建项目的原则"习惯优于配置"(没有配置就按习惯自动配置, 如果有制定配置,就按制定配置);

第一种自动配置:根据jar包,环境选择自动

配置的内容,例如,只要以来了spring-

boot-starter-web,内嵌servlet就存在,在 启动工程之前,所有tomcat相关的习惯配 置,都默认存在.

第二种自动配置:springboot工程可以扫描

所有的spring需要创建的对象(可以做到框 架的"0"xml配置文件).条件就是启动类所

在的同名包,和下级包

分区 day01all 的第 16页

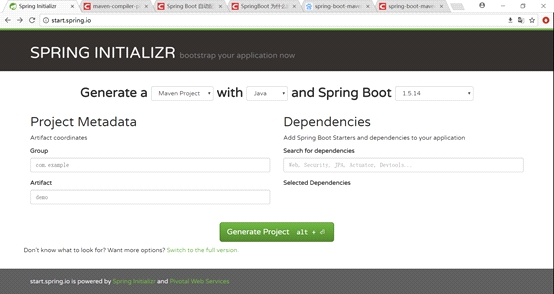
springboot快速搭建

2018年7月8日 22:38

官网转换

访问url http://start.spring.io

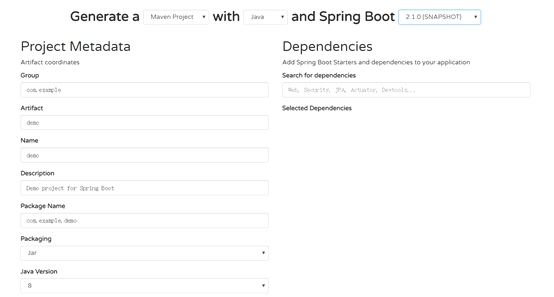
如图1-2所示

可以选择maven项目或者gradle，也可以选择语言

java、groovy，还可以选择对应的Spring Boot版本，目前最高版本是2.1.0（SNAPSHOT），点击Switch to full version开启更多

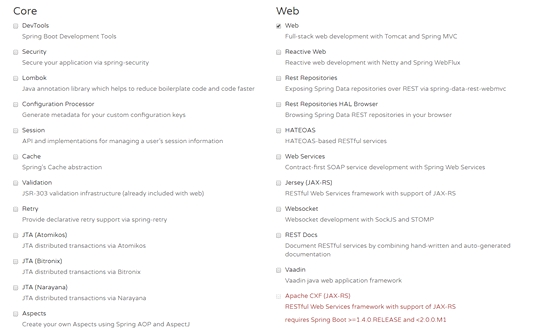
配置内容，如图1-3所示：

分区 day01all 的第 18页

可以以jar形式打包，也可以打包成war，java版本选择1.8，最低要 求1.6。

在下面可以见到选用的依赖技术（即starter pom的依赖）如图1-4

所示：

这里的每一项选择，都会在创建的项目pom文件中添加对应的

starter pom依赖，如图所示我们勾选了Web，所以在快速创建出来的项目的pom文件中会出现如下的依赖文本：

分区 day01all 的第 19页

来的项目的pom文件中会出现如下的依赖文本：

<dependency>

<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-web</artifactId>

</dependency>

点击generate按钮，该网站会根据你的配置自动完成一个zip压缩 包并且下载到本地。这样你可以将下载的文件作为第一个Spring

Boot工程导入你的开发工具中实现其他功能。

从这个下载的包中我们可以

手动搭建第一个maven的springboot

2018年9月10日 10:41

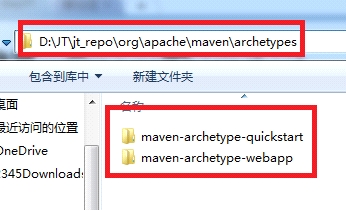
创建maven(quickstart)

springboot在访问jsp页面时需要eclipse 使用webapp骨架,如果没有页面需求,搭建

quickstart的标准java工程即可.

有可能出现的问题:

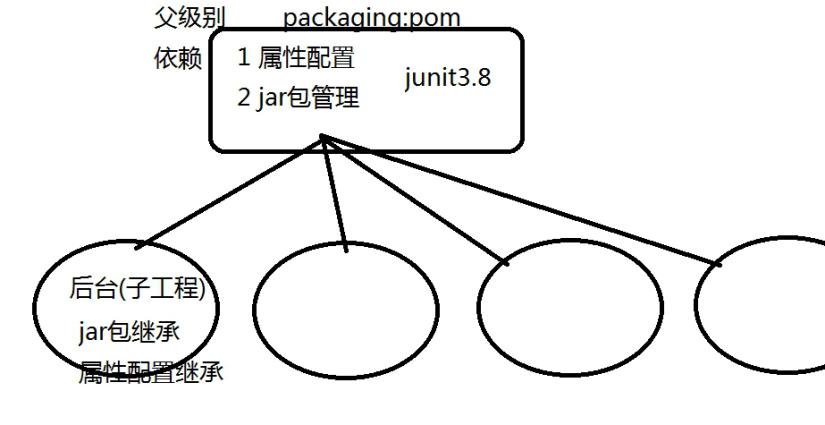
骨架文件出现异常.提示骨架文件的位置; 在maven的库 jt\_repo;

文件错误就出现在这里.

联网之后,将文件删除,重新搭建骨架,通过网 络重新下载.

修改pom文件

添加springboot的父级依赖. maven的继承:

<parent>

分区 day01all 的第 21页

<parent>

<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-parent</artifactId> <version>1.5.2.RELEASE</version>

<relativePath />

</parent>

添加maven的父级以来springboot-parent,使得当前工程具备springboot开发的特性(parent中规定了所有的

properties),springboot工程继承了parent后,会根据当前的版本(1.5.2.RELEASE),定义所有的相关依赖最适合的版本,starter-pom 简化以来时,不需要版本version

添加springboot对web应用的支持;

<dependency>

<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-web</artifactId>

</dependency>

见到的第一个starter-pom简化以来的内容.

发布编译插件

<build>

<plugins>

<plugin>

<groupId>

org.springframework.boot</groupI d>

<artifactId>spring-boot-maven-

plugin</artifactId> </plugin>

</plugins>

</build>

maven工程的发布,run as--install 当前maven工程,会使用

maven的插件,将当前工程按照packaging的类型打成对应的包,jar--jar包,war--war包.springboot在启动时,独立运行的项目,在jar包封装时,指明一个---主清单,寻找到main方法所在的类.

编写一个springboot的启动入口

@SpringBootApplication

public class StarterSpringBoot {

分区 day01all 的第 22页

public static void main(String[] args) {

SpringApplication.run(StarterSpringBoot.class, args);

}

}

当中的main方法就是普通的java运行入口.

核心在于@SpringBootApplication是核心注解,组合注解;

这种启动的代码就可以实现springboot的部分功能的展示

已经以来starter-web的情况下,编写一个controller测试,观察在没有任何xml配置文件情况下,能否完成一个controller的请求和返回.

@Controller

public class DemoController {

@RequestMapping(value="hello",method=Request Method.GET)

@ResponseBody

public String sayHello(){

return "hello 1805,springboot";

}

}

在只做了依赖的情况下,0配置完成了一个简单的web应用访问功能;调用的方法一定有对象,controller是有对象管理的(控制翻

转)

凭什么自动配置可以扫描到controller???

springboot基本常识

2018年9月10日 14:08

入口类中@SpringBootApplication注解

@SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan

Spring4.x版本之后,出现了组合注解的功能,需要的注解功能无需

重写,SpringBootApplication是springboot学习过程中的一个 最核心的注解,

SpringBootConfiguration:读取自定义配置文件使用.

EnableAutoConfiguration:开启自动配置,根据类路径,以来中的

jar包,类名等等内容,完成所有的习惯配置,tomcat的默认端口和默认项目访问路径,就是习惯配置中的一种.

ComponentScan: 开启启动类注解同名包和下级包的所有spring 实力的扫描功能;

springboot的配置文件

banner

启动时看到的字符串艺术图案

编辑一个banner.txt的文件,需要放到resources

src/main/resources

application配置文件

spring boot使用一个全局配置文件 application,格式可以是 properties,也可以是yml.

前者是常用的使用格式,京淘的配置文件就是这种格式

后者是目前企业正在喜欢使用的一种格式,yml更接近与面向数

分区 day01all 的第 24页

后者是目前企业正在喜欢使用的一种格式,yml更接近与面向数 据的内容;配置更简单

举例:端口和访问工程路径

application.properties

server.port=8090

server.context-path=/hello

application.yml

server:

[空格]port:[空格]8090

[空格]context-path:[空格]/hello

练习:

spring.redis.nodes=117.50.14.184:6379 spring.redis.pool.max-idle=8 spring.redis.pool.min-idle=0 spring.redis.pool.max-active=20 spring.redis.pool.max-wait=5000 spring.redis.timeout=1000000

修改成yml格式的配置内容

spring:

[]redis:

[][]nodes:[]117.50.14.184:6379

[][]timeout:[]1000000

[][]pool:

[][][]max-idle:[]8

[][][]min-idle:[]0

[][][]max-active:[]20

[][][]max-wait:[]5000

分区 day01all 的第 25页

[][][]max-wait:[]5000

eclipse中的yml不是直接支持的,有的版本直接有插件,有的还需 要单独安装.

使用xml配置(额外的配置)

springboot提倡的是0配置文件,springboot在spring的基础上做了非常多的拓展在自动配置,但是它能考虑大部分开发场景不代 表所有的开发场景都适应.例如,有些情况,自动配置无法满足我

们的使用.需要扫描的类创建实力,类并不在扫描范围.这时需要

使用单独的xml配置来引入到框架中

需要准备利用xml中bean标签来指向扫描范围外的类全路径名称 cn.tedu.DemoService

<bean class="cn.tedu.DemoService"/>

如何加载这个配置文件到当前框架容器中;

@Configuration:配置类的注解,组合了component @ImportResource:指向额外的配置xml文件,由当前对象负责加 载.

@Configuration

@ImportResource("classpath:applicationContext.xml") public class BeanConfig {

}

分区 day01all 的第 26页

springboot与持久层

2018年9月10日 15:05

springboot与jpa

目的:了解jpa的注解

了解通用持久层的概念(无需编写sql语句) 回忆:

mybatis的持久层使用:

1. spring管理mybatis的实力(mapper),通过xml配 置体现
2. 完成mapper接口类的创建
3. mapper.xml编写select update delete insert标签,id对应接口中的方法
4. 参数,返回要在xml中正确对应; resultMap:一对多,属性与字段对应

jpa介绍

java persistence api,java持久层api.主要的内 容就是使用注解,使用通用持久层访问数据库.

京淘使用jap+mybatis通用mapper.

springboot整合jpa

依赖jar包:

<!-- jpa相关jar包 -->

<dependency>

<groupId>

org.springframework.boot</grou pId>

<artifactId>spring-boot-starter- data-jpa</artifactId>

分区 day01all 的第 27页

data-jpa</artifactId> </dependency>

<dependency>

<groupId>mysql</groupId> <artifactId>mysql-connector- java</artifactId>

</dependency>

配置内容application.properties

#数据源

spring.datasource.driver-class- name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql:///sprin

gboot

spring.datasource.username=root spring.datasource.password=root

#jpa持久层配置

spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true

测试案例

新建一个jap注解的pojo,对应要创建一个表格

5个jpa注解

Entity:封装数据对象

Table:类与表格的对应关系

Id:主键

GeneratedValue:自增是通过这个注解的策略来定 义的strategy=GenerationType.IDENTITY

分区 day01all 的第 28页

义的strategy=GenerationType.IDENTITY @Column 实现属性与字段的对应

jpa提供通用持久层,他的功能就是实现单表的增删改查,无需任何sql语句的配置(底层通过注解,反射, 泛型自动拼接)

继承一下通用持久层的代码接口(spring-data中间 件会利用spring创建对象)

spring家族成员:spring,springmvc,springboot, spring-data(数据层中间件框架),springcloud.

springSession

public interface UserDao extends

JpaRepository<User, Integer>{

}

spring会创建这个dao的实例,根据泛型User能够通 过反射加注解获取user类中的所有注解内容.当前

的dao对象才会知道操作哪张表格(table的name

值,sql自动拼接的时候,from tb\_user)

Integer在做持久化操作,一般数据都需要序列化,序列化的id可以是Long也可以是Integer

插入数据测试

localhost:8090/addUser?name=\*\*&age=\*\*

在表格tb\_user中插入数据

UserController

分区 day01all 的第 29页

UserController

@Controller

public class UserController {

@Autowired

private UserService userSerivce;

@RequestMapping("addUser")

@ResponseBody

public String addUser(String

name,Integer age){

userSerivce.addUser(name,age); return "success";

}

}

UserService

@Service

public class UserService {

@Autowired

private UserDao userDao;

public void addUser(String name, Integer age) {

//利用通用daouserDao实现数据插入

//save,里面传入的参数是pojo的对象 User

//根据表格,对象数据,在对应表格中直接 执行insert命令

User user=new User();

user.setName(name);

user.setAge(age);

userDao.save(user);

分区 day01all 的第 30页

userDao.save(user);

}

}

查询一个user利用id(name)

localhost:8090/findUser?id=1

@RequestMapping("findUser")

@ResponseBody

public User findUser(Integer id){

//id=1,name=hanlaoshi,age=30

//json字符串格

式:{"id":1,"name":"hanlaoshi","age":30}

//List<User> user1 user2

//[{},{}] javaScript object notation 在js 中ajax回调函数接受数据

//直接解析成js对象,无需转化

User

user=userSerivce.findUserById(id); return user;

}

public User findUserById(Integer id) {

User user = userDao.findOne(id); return user;

}

spring框架可以在底层创建接口类的实例化对象. 如果涉及到解耦,完成接口类,和实例化的子类,由

分区 day01all 的第 31页

如果涉及到解耦,完成接口类,和实例化的子类,由 spring管理对象.

泛型+注解+反射+jdbc(hibernate)完成了通用持久 层的编写;

架构师借鉴一下其他框架对应功能的源码,自行定 义自己的框架内容.

springboot与mybatis整合

springboot在整合mybatis之后,任何ssm框架的项目都可以利用springboot来搭建.

pom依赖

<!--mybatis与springboot整合-->

<dependency>

<groupId>

org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot- starter</artifactId>

<version>1.3.0</version>

</dependency>

application.properties配置mybatis的内容

别名包;mapper.xml文件的扫描;驼峰命名(表格字 段与对象的属性)

驼峰命名对应关系:mybatis的配置中可以实现表格

字段与对象属性的对应关系,字段名:order\_name 对象属性必须通过驼峰命名完成对应:orderName select \* from tb\_order;

分区 day01all 的第 32页

select \* from tb\_order;

mybatis.typeAliasesPackage=com.jt.manage. pojo 别名包

mybatis.mapperLocations=classpath:mapper s/\*.xml 扫描映射文件地址

mybatis.configuration.mapUnderscoreToCa

melCase=true 开启驼峰映射关系

利用xml文件完成查询所有user表格的内容 localhost:8090/findAll

UserController

//调用mybatis完成持久层数据的操作

@RequestMapping("findAll")

@ResponseBody

public List<User> findAll(){

return userSerivce.findAll();

}

UserService

@Autowired

private UserMapper userMapper;

public List<User> findAll() {

return userMapper.findAll();

}

UserMapper略

映射文件UserMapper.xml

namespace对应接口类UserMapper的全路径

分区 day01all 的第 33页

namespace对应接口类UserMapper的全路径

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

PUBLIC "-//mybatis.org//DTD Mapper

3.0//EN"

"http://mybatis.org/dtd/mybatis-3-

mapper.dtd">

<mapper

namespace="com.jt.mapper.UserMapper">

<select id="findAll" resultType="User">

SELECT \* FROM tb\_user

</select>

</mapper>

spring凭什么帮你创建mapper接口类实例对象; 回忆:spring和mybatis整合过程,需要扫描mapper 包;

启动类中添加扫描的注解

MapperScan("com.jt.mapper")分区 day01all 的第 34

springboot的jsp页面

2018年9月11日 8:56

springboot工程,jsp页面在内嵌的servlet的容器上直接运行不支持, 内嵌的tomcat,jetty不能以jar的形式运行jsp,使用webapp这种格式.如

果想直接使用jar的包封装页面,必须引入模板引擎themleaf,boostrap

在京淘阶段的页面展示,就利用maven工程的webapp骨架来完成.

搭建第二个springboot工程(webapp骨架)

• 页面访问

springboot运行,在web结构中的jsp需要单独以来jasper的jar 包,支持当前框架运行jsp

<!--servlet依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet- api</artifactId>

<version>3.1.0</version>

<scope>provided</scope> </dependency>

<!--jstl依赖 -->

<dependency>

<groupId>javax.servlet</groupId> <artifactId>jstl</artifactId>

</dependency>

<!--使jsp页面生效 --> <dependency>

<groupId>

分区 day02am 的第 3页

<groupId>

org.apache.tomcat.embed</groupId> <artifactId>tomcat-embed-

jasper</artifactId>

</dependency>

配置页面渲染的前缀后缀(ssm在spring-mvc.xml配

置,properties)

#jsp suffix prefix

spring.mvc.view.prefix=/WEB-INF/views

spring.mvc.view.suffix=.jsp

启动类+controller测试

@SpringBootApplication

@Controller

public class StarterJSPTest {

public static void main(String[] args) {

SpringApplication.run(StarterJSPTest.class, args);

}

//页面跳转的测试

@RequestMapping("/")

public String goIndex(Model model){

return "index";

}

}

分区 day02am 的第 4页

springboot的发布

2018年9月11日 8:56

ssm框架项目使用的是eclipse整合tomcat进行发布缺点:

1. 多个项目同时发布到同一个tomcat,无法区分端口
2. 分布式测试无法执行.

一个tomcat启动一个工程,多个tomcat启动不同工程,这样tomcat可以 自由部署搭建.

发布springboot工程2种方式

1. 打包直接在jdk环境运行jar
2. 运行在外部tomcat容器的包,war包

直接在tomcat外部容器执行打的war包,无法访问功能 内嵌tomcat与外部tomcat冲突--移除内嵌tomcat

springboot的war包在外部运行的tomcat中没有入口--单独编写外 部容器的入口方法

• 移除starter-web中的内嵌tomcat

<dependency>

<groupId>

org.springframework.boot</groupId> <artifactId>spring-boot-starter-

web</artifactId> <exclusions>

<exclusion>

<groupId>

org.springframework.boot</groupId> <artifactId>spring-boot-starter-

tomcat</artifactId> </exclusion>

分区 day02am 的第 5 页

</exclusion>

</exclusions>

</dependency>

编写提供外部tomcat运行的入口方法

public class StarterForTomcat extends

SpringBootServletInitializer{

//覆盖父类方法,将入口指向springboot的main @Override

protected SpringApplicationBuilder

configure(SpringApplicationBuilder builder) {

//参数Builder就是容器创建整个spring容器的起始对 象

return builder.sources(StarterJSPTest.class);

}

}