### SSM项目搭建

1. 博客：

<https://www.hollischuang.com/>

1. Apache:

<http://tomcat.apache.org/tomcat-7.0-doc/servletapi/index.html>

1. Spirng:

<https://docs.spring.io/spring/docs/4.3.21.RELEASE/spring-framework-reference/htmlsingle/>

1. Mybatis:

<http://www.mybatis.org/mybatis-3/zh/java-api.html>

1. 面试：

<https://github.com/Snailclimb/JavaGuide>

<https://github.com/doocs/advanced-java>

注意：先要先装好jdk,maven,idea,mysql

业务与技术的关系：

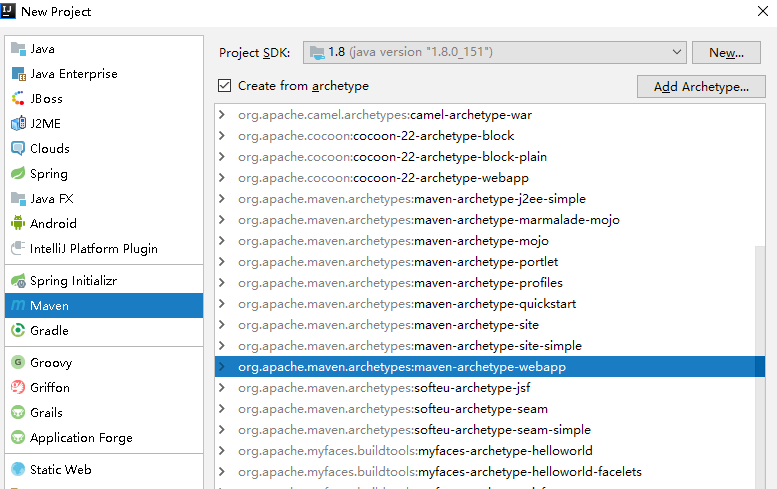
业务是技术的基础和依托，技术是业务稳定和提升的保证，两者相辅相成。

如何熟悉一个新产品：（理性认识-->感性认识的升华）

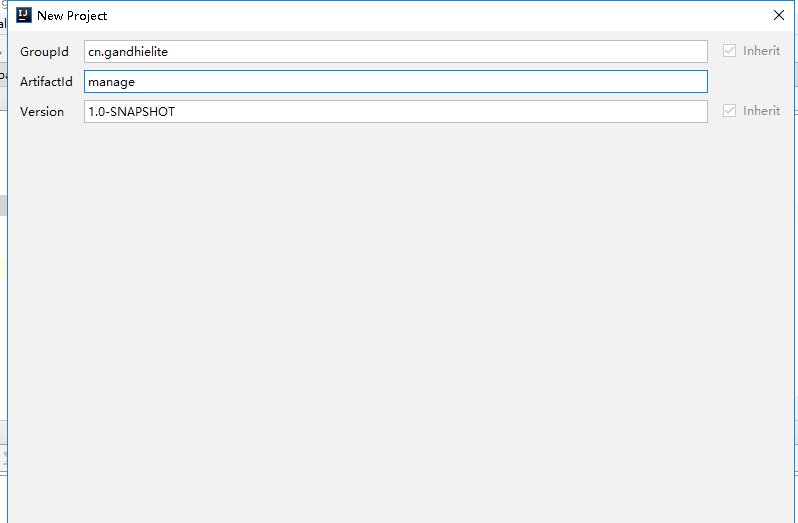
使用 了解业务逻辑 技术实现 使用

##### 1．基础项目构建：

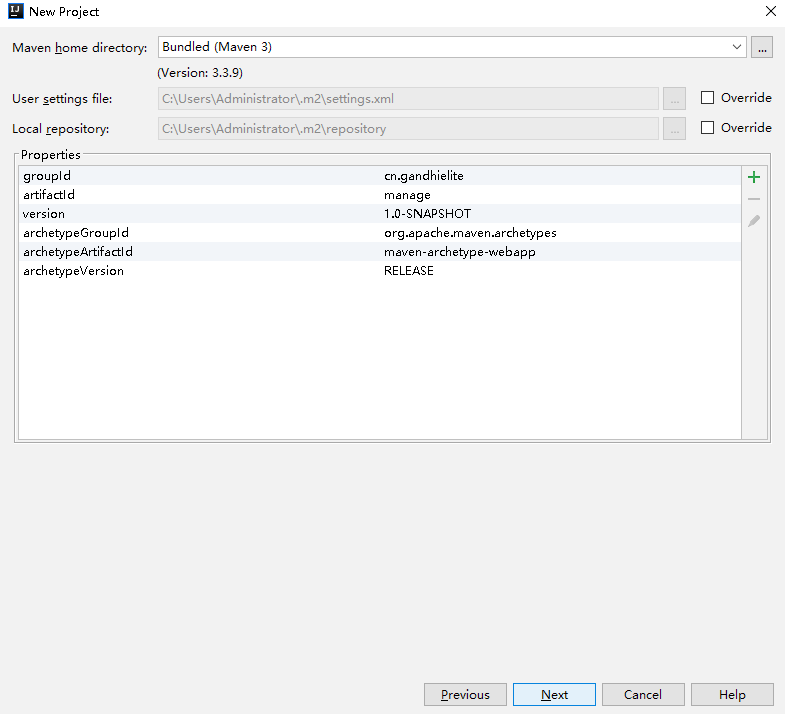
步骤一：选择maven webapp项目构建。



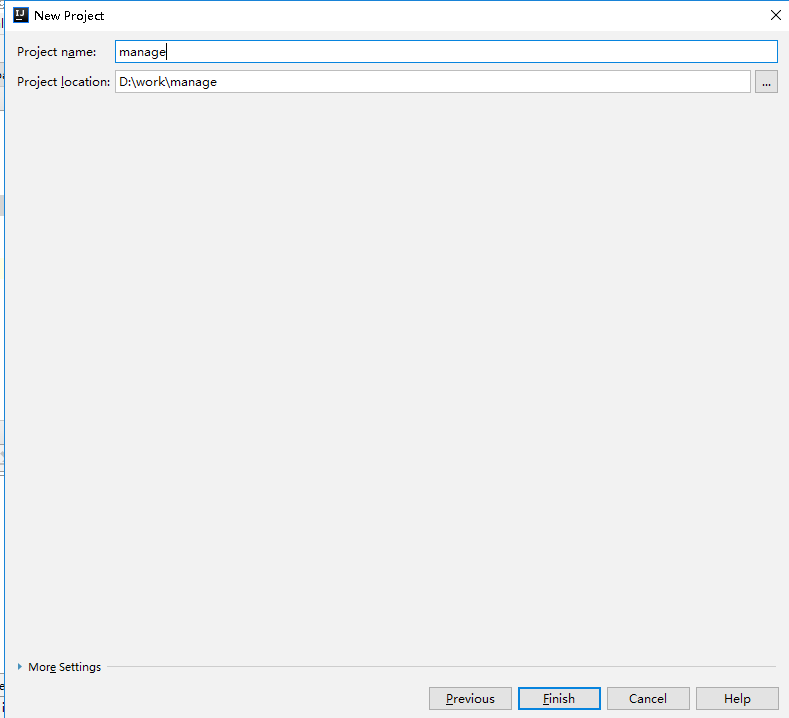
步骤二：填写项目名称。groupId一般是公司域名反写,artifactId是项目名称



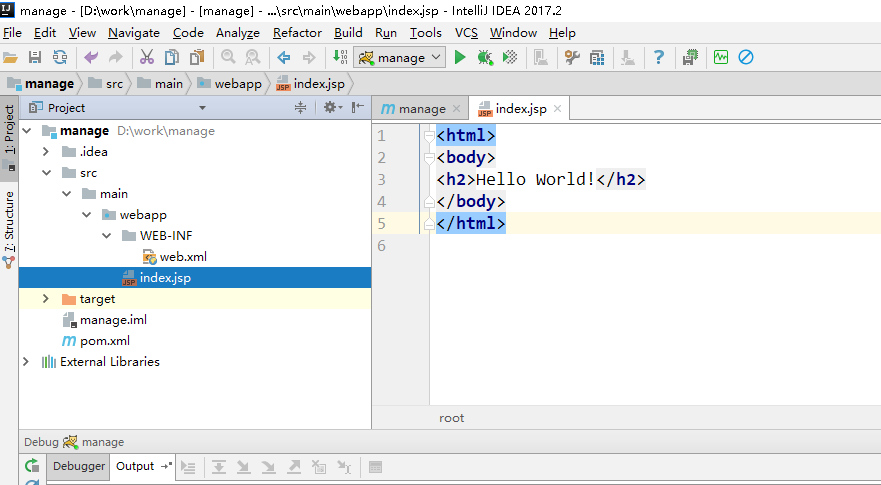
步骤三：选择maven的路径，一般是自己的maven仓库，需要之前配置好。



步骤四：设置项目在硬盘中的名称。



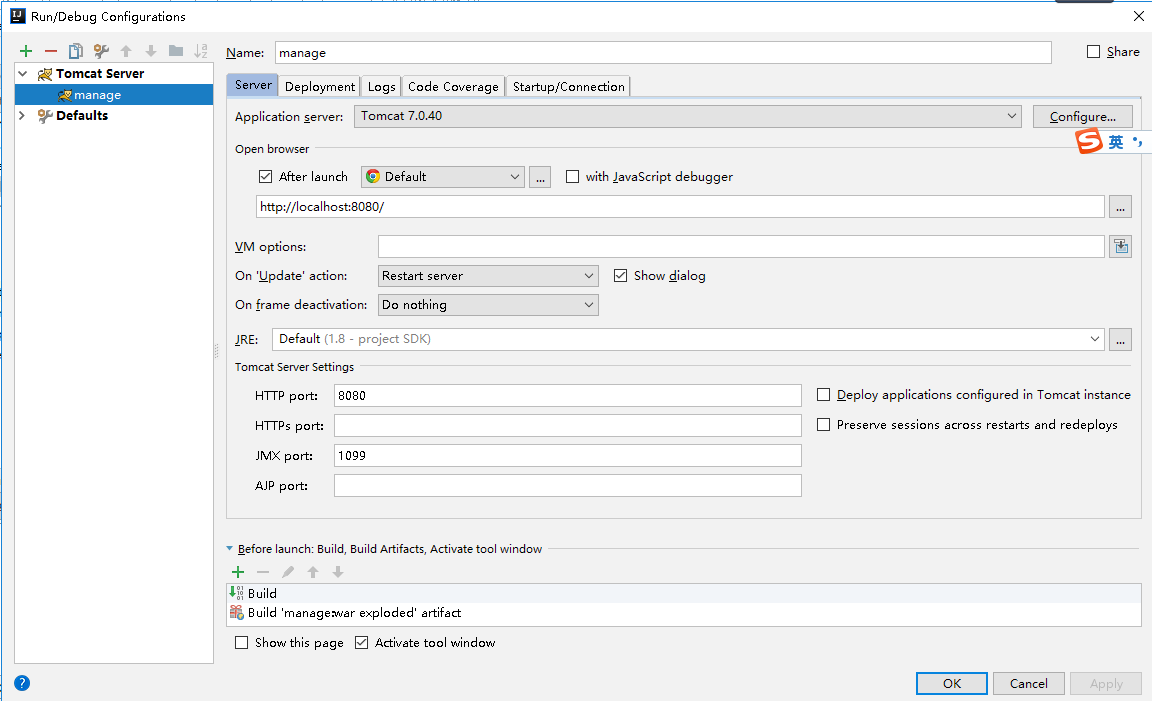
步骤五：点击finish后的项目结构：



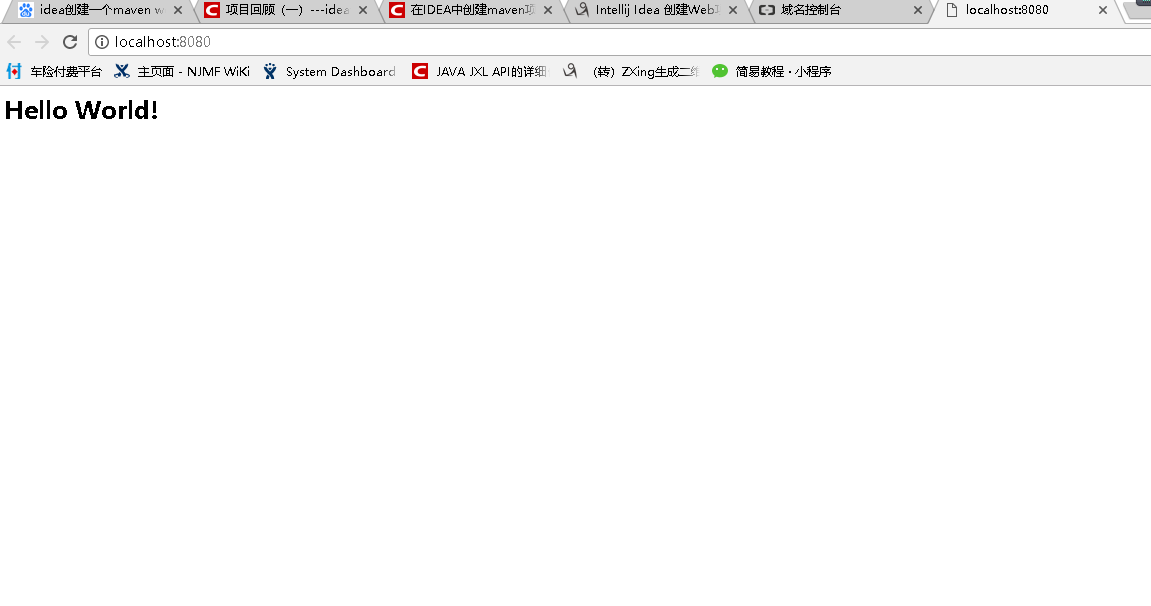
步骤六：idea配置tomcat容器。

Vm options: -Dfile.encoding=UTF-8 可防止启动时日志乱码

Deployment: 选择包类型部署, manage:war exploded 用于开发manage.war 用于生产部署

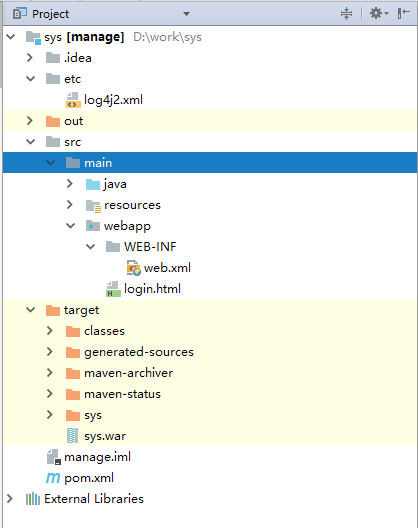


步骤七：启动tomcat。表示基础的项目搭建成功



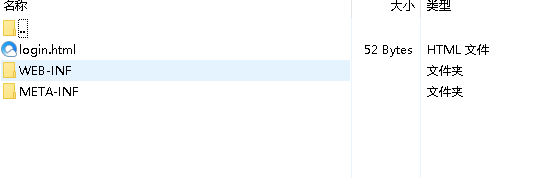
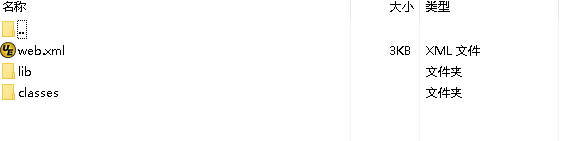
##### 2. 项目进阶：

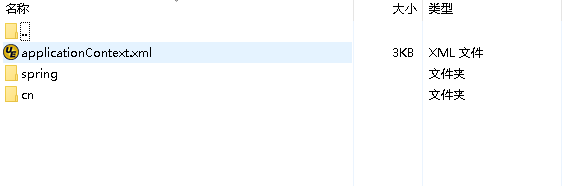
1.项目结构：



java包放java代码，resources包放配置文件，webapp包放静态资源文件

classpath:指的是src目录（classes），项目打包后没有main,java,resources,webapp包了。如下图：



1.项目配置文件，创建

步骤一：pom.xml增加依赖，可拷贝之前的sys项目中 的pom包

步骤二：web.xml文件配置

<url-pattern>/</url-pattern>表示的为只匹配/login和/demo后面不带.xxx的网页

<url-pattern>/\*</url-pattern>表示匹配的为所有网页包括.jsp/.js/.html的这些网页

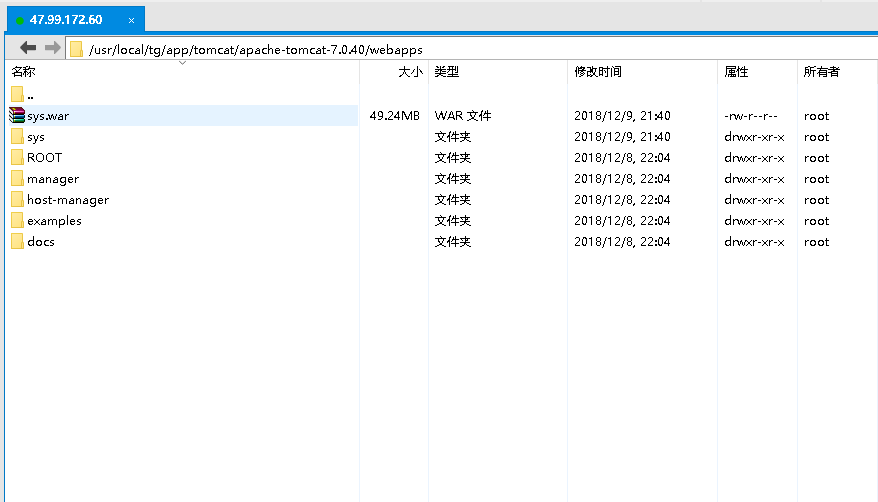
Springmvc配置中url-pattern应该使用/不然会导致jsp页面无法被解析，由于都被DispatcherServlet给转发了

步骤三：spring文件配置

2.Tomcat容器配置

步骤四：当一个web项目可以正常启动时,将该项目打包放到linux中。

例如将sys.war包拖入/usr/local/tg/app/tomcat/apache-tomcat-7.0.40/webapps下，启动tomcat时会自动解压war包。



注意：在/conf/Catalina/localhost包下需要新增一个sys.xml（和war包名一致）文件用于指定项目的访问映射关系。

sys.xml中内容：

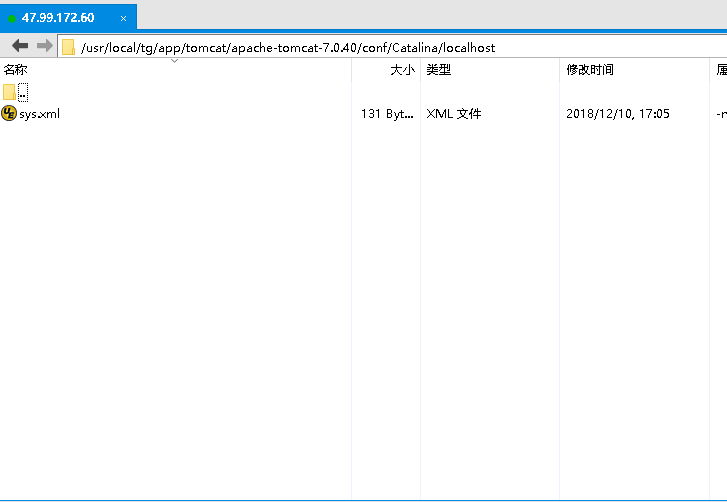
<Context path="/sys" docBase="/usr/local/tg/app/tomcat/apache-tomcat-7.0.40/webapps/sys" debug="0" privileged="true">

</Context>

其中path是指项目的发布路径，也就是访问路径，假如像上边那样填写，就要这样访问：http://localhost:8080/sys/login.hmtl；docBase是指项目的WebContent(eclipse)或WebRoot(myeclipse)或webapp(idea)目录，很好理解，你的项目最终发布，就是发布的这个目录，通过配置，直接让tomcat指向这个目录，这样就可以运行项目啦。

注意：xml的文件名一定要和发布路径一致！在本例中xml文件名必须为：sys

参照：https://blog.csdn.net/hwhua1986/article/details/78436128



3.idea 和 Linux中tomcat项目部署的差异：

1.idea中项目的根路径： “/”

2.Linux中项目的根路径：“/sys”(sys项目)

保持一致的方式是：删除ROOT文件夹，server.xml中配置

Host节点下添加： <Context docBase=”sys” path=”” />

表示：当访问<http://localhost:8080>时就访问到了项目的根路径/sys。

未配置xml则需要访问地址<http://localhost:8080>/sys

##### 3. 常见异常问题

1.云服务器中Tomcat启动慢，有时需要10分钟才能启动好，Tomcat作为轻量级应用服务器，一般不会出现这种问题。

查看信息发现：

Creation of SecureRandom instance for session ID generation using [SHA1PRNG] took [601,851] milliseconds.

在JVM环境中解决

打开$JAVA\_PATH/jre/lib/security/java.security这个文件,找到下面的内容:

securerandom.source=file:/dev/urandom

替换成

securerandom.source=file:/dev/./urandom

参考：<https://www.jianshu.com/p/576d356dc163>

2.springnvc绑定前台日期字符串失败：

前台传递日期字符串，后台使用Date接受。出现转换失败问题。

解决方式：

1. 在日期Date字段前加上注解：

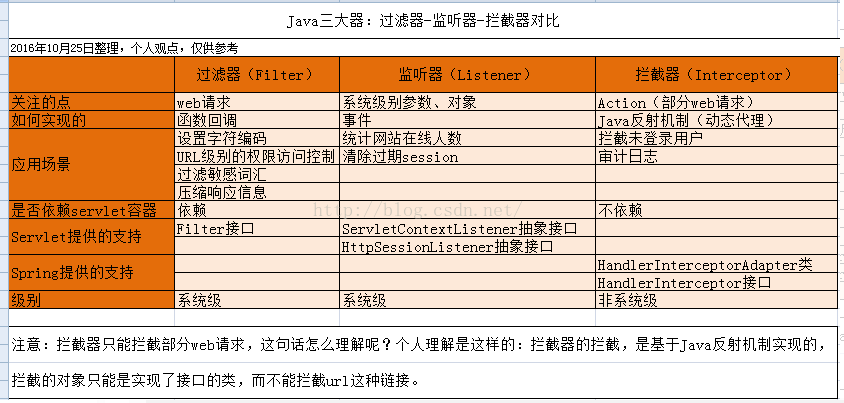
@DateTimeFormat(pattern="yyyy-MM-dd")

1. 后台返回日期格式化字段前加上注解：

@JsonFormat(pattern = "yyyy-MM-dd",timezone = "GMT+8")

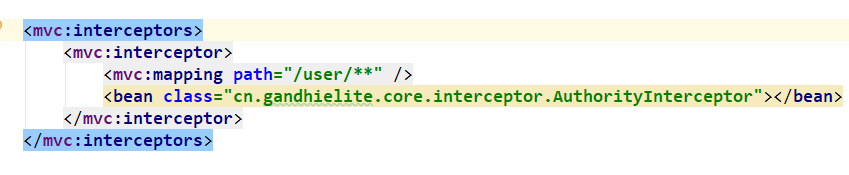
或@JSONField(format = "yyyy-MM-dd")

##### 4. 过滤器 拦截器 监听器



web.xml 的加载顺序是：context-param -> listener -> filter -> servlet

interceptor的配置在springmvc.xml（struts2.xml）中



1. Listener的应用场景（事件驱动，支持8种接口）

加载log4j2.xml日志文件

统计网站在线人数

1. Filter 的应用场景

设置字符编码

url级别的权限访问控制

1. Interceptor的应用场景（基于aop思想对controller访问前后的控制）

用户权限控制

##### 用户权限控制设计

后台做了权限控制：

用户登录后容器中保存当前登录信息->Filter过滤url信息->Interceptor做权限控制（接口访问）

表设计：

用户表：admin;

角色表：role;

用户角色关联关系表：admin\_role\_relation;

权限表： authority;

角色权限管理关系表： role\_authority\_relation;

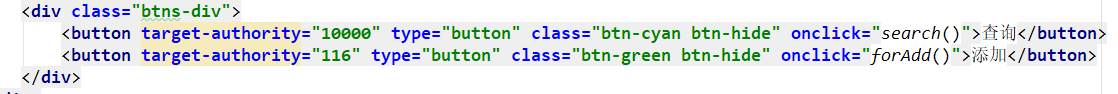
资源URL表： resources;

权限资源关联关系表：authority\_resources\_relation;

前台做了权限控制：

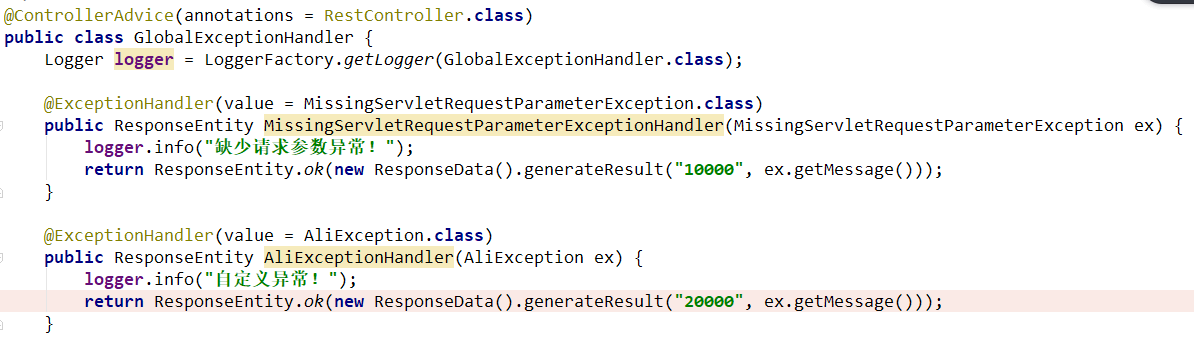
登录成功进入index页面后立即加载登录用户的权限信息并存在sessionStorage中。





当数据库没有配置对应的权限时，sessionStorage没有对应的authorityId，如10000是无效的，则查询按钮不显示。有效时候清除btn-hide类。

##### 全局异常处理器



使用注解代码xml文件配置：

1.@ControllerAdvice(annotations = RestController.class)

表示带有RestController注解的异常最终在这个被这个全局异常处理器捕获处理

2.@ExceptionHandler(value = MissingServletRequestParameterException.class)

缺少参数这类异常将由此注解下方法处理。

3.return ResponseEntity.ok(new ResponseData().generateResult("10000", ex.getMessage()));

返回ResponseEntity，最终返回页面的是json字符串。

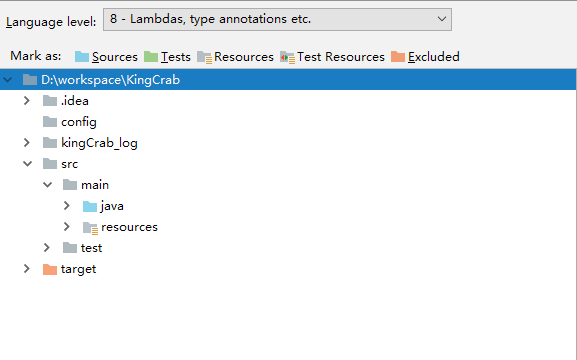
4.注意：为什么有的异常throw出去后还需要在方法上做声明throws，有的则不需要呢？

这个看throw出去的异常继承于那个异常类，RuntimeException(无需声明)，Exception(需声明)，一般自定义异常继承RuntimeException

##### 单元测试

设置文件类别：有时目录下无法创建java文件，如下可以将文件夹标记为具体的类型。

想要创建java文件，可以将文件夹标记为Sources。创建测试类则标记为Tests。



1.软件测试的四个阶段：单元测试-->集成测试-->系统测试-->回归测试

2.测试方法：

黑盒测试（测试人员 功能测试）侧重功能验证

白盒测试（开发人员 单元测试）侧重软件内部结构验证

灰盒测试（黑盒和白盒的中间态 集成测试）

3.单元测试的步骤：

初始化测试环境、准备测试数据。

调用需要被测试的单元。

收集结果，并与期望值比较。

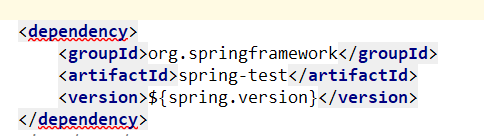
测试数据清理。

1. idea自动生成test类：

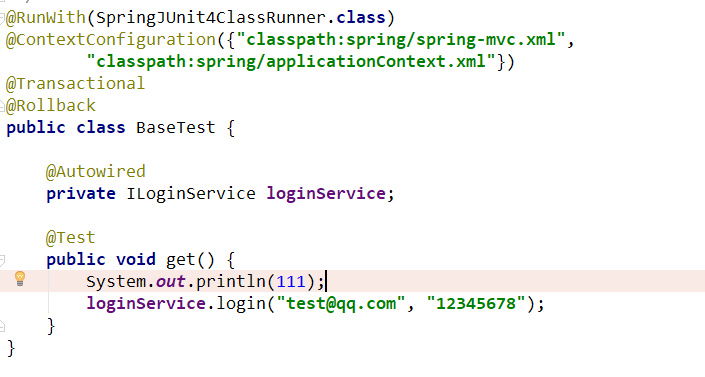
file-> setting -> keymap 搜索：test

设置快捷键

1. 和spring的集成
2. 引入spring-test 包



1. 配置spring xml的文件路径,这样便可以注入对象了



##### 文件的上传下载导出

1. 文件上传（图片）

文件上传三要素：

Enctype:mutipart/form-data,

提交方式：post,

Form表格，input框，file类型：form-input-file

接受单个文件：

MultipartFile file 接受单个文件

file.transferTo(new File("...")) 文件保存

接受多个文件：

HttpServletRequest request 参数接受

MultipartHttpServletRequest mpRequest

= (MultipartHttpServletRequest) request; 接受多个文件

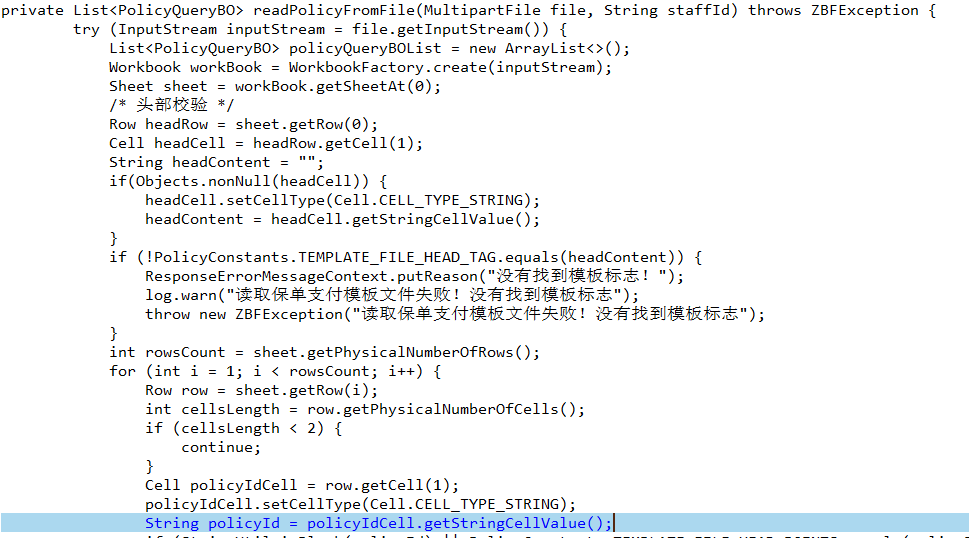
MultipartFile file = mpRequest.getFile(fileName); 获取指定名字的文件

1. 上传excel（poi）

工作簿---页---行---单元格

Workbook---sheet---row---cell

读列名---读内容



1. 导出为excel（jxl）

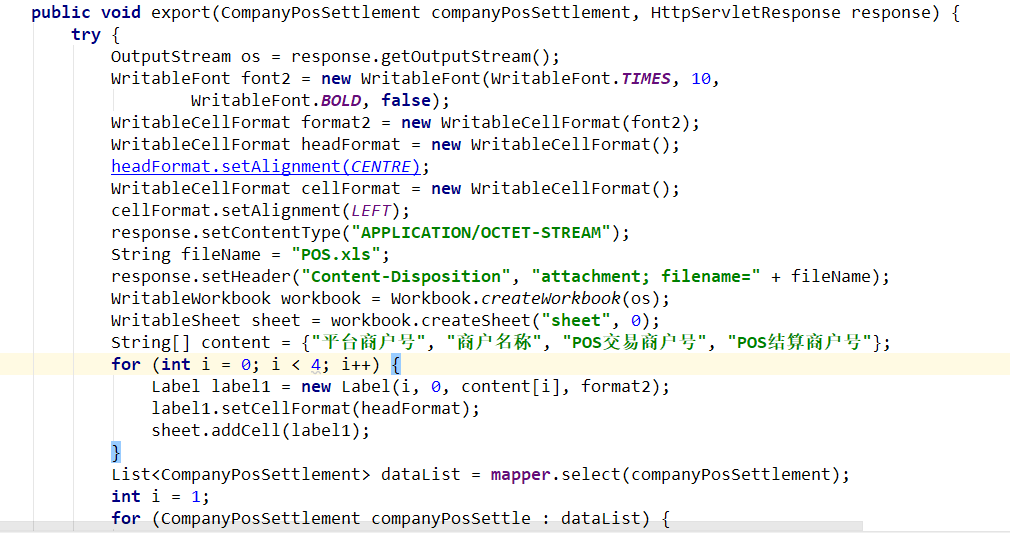
工作簿---页面—单元格---添加单元格

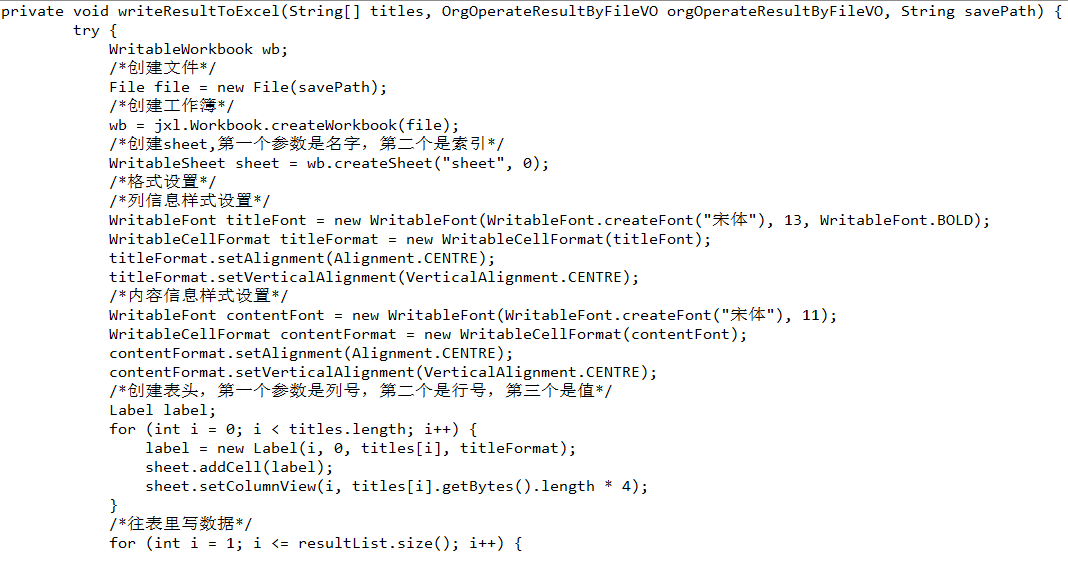
WritableWorkbook--- WritableSheet--- Label--- sheet.addCell(lable)

注意：

workbook.createWorkbook(outputStream) 页面导出

workbook.createWorkbook(file) 导出到指定的文件





1. 文件下载

两个头一个流

Content-type: 文件的mime类型

Content-Disposition: 告诉浏览器以下载的方式打开

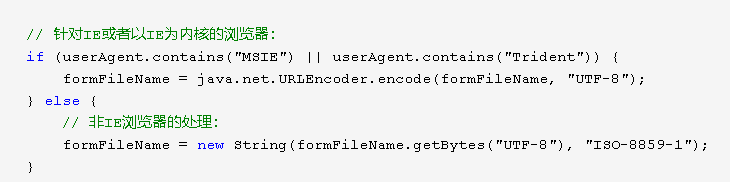
inputStream: 读资源文件

response.setHeader("Content-Type", "APPLICATION/OCTET-STREAM");

response.setHeader("Content-Disposition", "attachment; filename=" + fileName);

文件名为中文时对fileName的处理：

String userAgent = request.getHeader("USER-AGENT");



##### Oracel 数据库

1. rownum 伪列

是对查询到的结果集加的序列，强调先有结果集。并且它是动态的序列。每次从1开始！

对于rownum > 10这样的条件是查询不到符合的结果。因为对于rownum为1的结果被过滤后，下一条记录的rownum又是1，所以不可能有记录。

1. rowid 伪列

是数据的物理地址，唯一存在的。

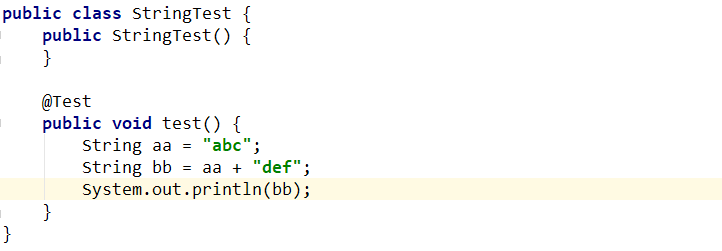
1. 分页查询：

select \* from (select rownum rn,wt.\* from wallet\_trade wt where rownum <= 20) a

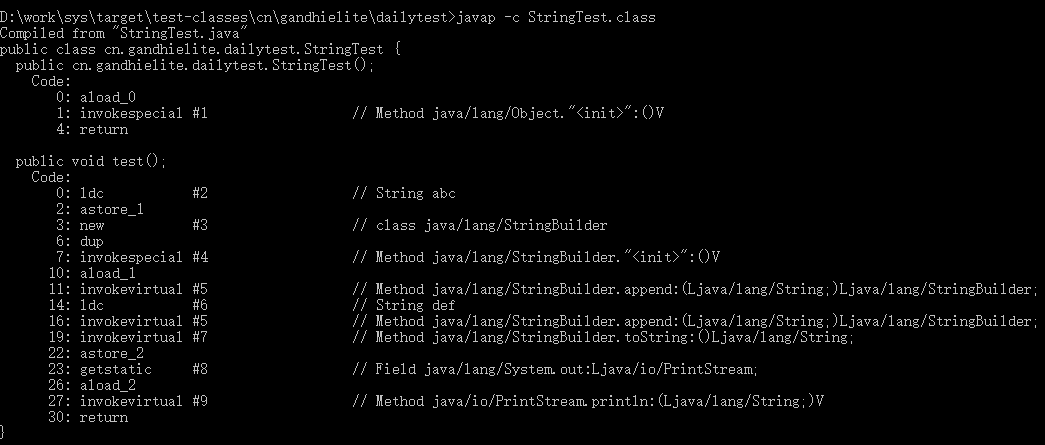
where rn > 10;

##### Java基础源代码

String:



编译后的字节码文件如下：javap –c StringTest.class



在一个方法中需要大量引用实例域变量的时候，使用方法中的局部变量代替引用可以减少getfield操作(栈中声明对象的引用)的次数，提高性能。

Integer:

使用移位操作提升性能：

q = (i \* 52429) >>> (16+3); //q = i / 10

i\*52429/524288=52429.0/524288=0.1000003814697......

6位的精度已经足够多了，所以就是i\*0.1

r = i - ((q << 3) + (q << 1)); //r = i – 10\*q

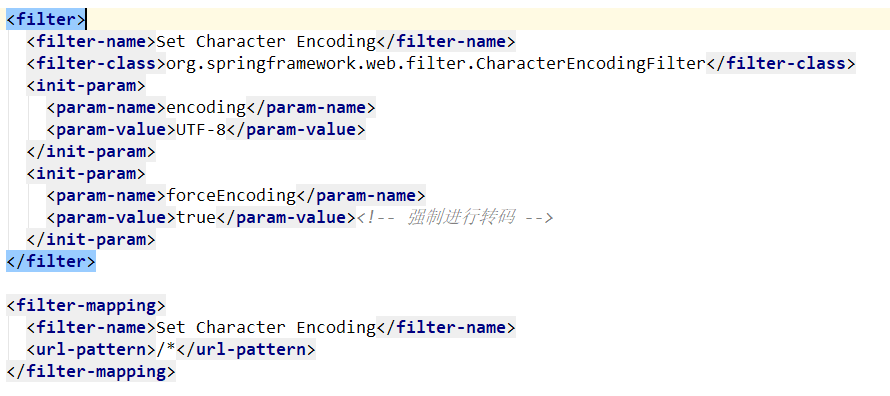
Enum:

ArrayList:

HashMap:

##### 请求响应中文乱码问题（CharacterEncodingFilter）

Web.xml中配置过滤器：



Jvm编码，容器编码（tomcat），系统编码

一个汉字：ISO-8859-1编码成1个字节；GBK编码成2个字节；UTF-8编码成3个字节

获取平台默认字符集（Jvm）：Charset.defaultCharset()

各种编码的解释：https://www.cnblogs.com/fuyoucaoyu/articles/5707911.html

##### PageHelper的使用

1. 如何实现分页

Page: 分页对象

PageInfo: 分页结果的封装

PageHelper：分页核心处理类

PageInterceptor：分页拦截器

分页的本质就是mybatis在查询前按要求添加limit(mysql),rownum(oracle)关键字和分页信息(pageNum,pageSize)。

Executor是mybatis执行数据库操作的接口, PageInterceptor对该接口的查询方法做了拦截，因此mybatis在做查询的时候会先做拦截器处理（即分页操作）然后再执行相应的查询。

PageHelper.startPage(int pageNum, int pageSize)执行这条语句会经过以下过程：

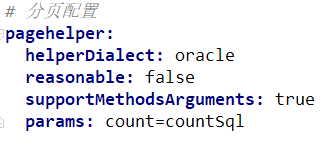
1. 创建Page对象，设置pageNum,pageSize的值和分页的起止行startRow,endRow
2. 将Page对象保存到当前线程的副本中ThreadLocal,即分页信息只和当前线程相关，多线程不受影响
3. 分页查询为何只针对下一次查询生效

当完成一次分页查询后会调用LOCAL\_PAGE.remove(),移除当前线程保存的分页对象信息。当进行下一次查询时就不会进行分页操作了。

注意：查询操作还是会被PageInterceptor拦截，但是执行dialect.skip(ms, parameter, rowBounds)方法时因为Page对象为空，故会返回true即跳过分页操作。

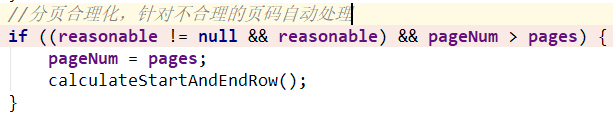
1. Resonable的设置问题

Springboot中有关于分页的配置



表示取消分页合理化。

分页合理化（reasonable=true）：表示当查询的页码pageNum大于总页数pages时，会返回pageNum=pages时的查询结果。



##### 日志信息脱敏处理

打印日志时，银行卡，姓名，身份证，电话号码等敏感信息脱敏处理

1. 重写toString方法，对部分字段进行格式化处理，完成脱敏展示
2. 对日志框架进行重写，复杂。（尝试）

##### 消息转换器HttpMessageConverter

参考：https://my.oschina.net/lichhao/blog/172562

HttpInputMessage（一次请求的抽象，读报文） ，HttpOutputMessage（一次响应的抽象，写报文）

RequestResponseBodyMethodProcessor（入参和出参）

HandlerMethodArgumentResolver（入参数据转换）

HandlerMethodReturnValueHandler（返回数据转换）

##### Springboot- CommandLineRunner

作用：项目启动后加载配置信息到内存

项目启动后会调用CommandLineRunner实现类的run方法。

CommandLineRunner可以有多个实现类，启动顺序可以由@Order注解控制，参数值越小越先执行

@Order(1)

|  |
| --- |
| @Component  @Order(1)  public class InitializationProcessor implements CommandLineRunner {  private Logger logger = LoggerFactory.getLogger(InitializationProcessor.class);  private static final List<String> MODULE\_HOLDER = Arrays.asList("all","app");  @Autowired  private SystemFixedParamMapper systemFixedParamMapper;  @Override  public void run(String... strings) {  Condition condition = new Condition(SystemFixedParam.class);  condition.selectProperties("key","value").createCriteria().andIn("module", MODULE\_HOLDER);  List<SystemFixedParam> sysList = systemFixedParamMapper.selectByExample(condition);  for (SystemFixedParam temp : sysList) {  String value = temp.getValue();  System.setProperty(temp.getKey(), value.trim());  logger.info("加载到\*\*\*\*\*temp.getKey() =" + temp.getKey() + "\*\*\*\*\*temp.getValue() =" + value.trim() + "\*\*\*");  }  }  } |

##### Springboot 解决跨域问题

@Configuration等价于spring xml中的<beans> 配置spring容器(应用上下文)

第一个方法为添加拦截器

第二个方法为添加跨域支持（全局），如果想要特殊处理，可以在单个controller中添加@CrossOrigin注解。

|  |
| --- |
| @Configuration  public class WebConfig extends WebMvcConfigurerAdapter {  @Autowired  private AuthorityInterceptor authorityInterceptor;  @Override  public void addInterceptors(InterceptorRegistry registry) {  registry.addInterceptor(authorityInterceptor).addPathPatterns("/\*\*");  }  @Override  public void addCorsMappings(CorsRegistry registry) {  registry.addMapping("/\*\*")  .allowedOrigins("\*")  .allowCredentials(true)  .allowedMethods(HttpConstants.METHOD\_GET,  HttpConstants.METHOD\_POST,  HttpConstants.METHOD\_OPTIONS)  .allowedHeaders(HttpHeaders.ACCEPT,  HttpHeaders.ORIGIN)  .exposedHeaders(HttpHeaders.CONTENT\_DISPOSITION)  .maxAge(3600);  }  } |

对于前后端分离的项目，跨域问题是肯定存在的。

解决方法通常有：

1. Jsonp jsonp是如何解决跨域问题的？
2. Spring-boot中使用上述方式解决跨域

allowedOrigins：允许跨域请求的域名（ip+port）

addMapping: 请求路径

allowedMethods：跨域请求的方式

##### Springboot集成redis

1. pom引入jar包

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency> |

1. application.yml增加配置

|  |
| --- |
| redis:  host: 172.31.227.1  port: 40020  password: leilong!2018#redis  timeout: 5000  database: 5  pool:  max-active: 3000  max-wait: 1500  max-idle: 1000  testOnBorrow: false  testOnReturn: false |

1. 直接注入

使用了装饰者模式进行了一层包装

StringRedisTemplate继承了RedisTemplate<String, String>

|  |
| --- |
| @Service  public class RedisService implements IRedisService {  @Autowired  private StringRedisTemplate stringRedisTemplate;  @Autowired  private RedisTemplate<Object, Object> redisTemplate;  @Override  public String getStr(String key) {  return stringRedisTemplate.opsForValue().get(key);  }  @Override  public void setStr(String key, String val) {  stringRedisTemplate.opsForValue().set(key, val);  }  @Override  public void setStr(String key, String val, long time, TimeUnit timeUnit) {  stringRedisTemplate.opsForValue().set(key,val,time,timeUnit);  }  @Override  public void setStrExpire(String key, long time, TimeUnit timeUnit) {  stringRedisTemplate.expire(key,time,timeUnit);  }  @Override  public void del(String key) {  stringRedisTemplate.delete(key);  }  @Override  public Object getObj(Object o) {  return redisTemplate.opsForValue().get(o);  }  } |

##### Springboot集成rabbitmq

1. pom引入jar包

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

1. application.xml中增加配置

ip+15672可打开mq后台管理系统

|  |
| --- |
| rabbitmq:  host: 172.31.218.254  port: 5672  username: mqterm  password: mqterm201809#term  publisher-confirms: true  virtual-host: /  listener:  retry:  #最大重试次数  max-attempts: 5  #是否开启消费者重试（为false时关闭消费者重试，这时消费端代码异常会一直重复收到消息）  enabled: true  #重试间隔时间（单位毫秒）  initial-interval: 2000  #重试次数超过上面的设置之后是否丢弃（false不丢弃时需要写相应代码将该消息加入死信队列）  default-requeue-rejected: false |

1. 队列配置

|  |
| --- |
| @Configuration  public class RabbitConfig {    @Bean  public Queue Queue() {  return new Queue("hello");  }  } |

1. 注入使用

rabbitTemplate是springboot 提供的默认实现

|  |
| --- |
| //生产者  @Component  public class HelloSender {    @Autowired  private AmqpTemplate rabbitTemplate;    public void send() {  String context = "hello " + new Date();  System.out.println("Sender : " + context);  this.rabbitTemplate.convertAndSend("hello", context);  }    }  //消费者  @Component  @RabbitListener(queues = "hello")  public class HelloReceiver {    @RabbitHandler  public void process(String hello) {  System.out.println("Receiver : " + hello);  }  } |

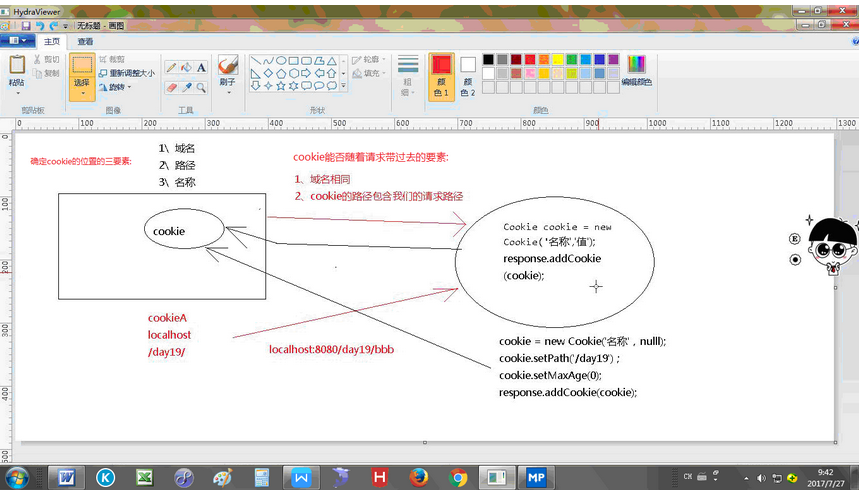
##### Cookie和Session

|  |
| --- |
| 会话:用户打开一个浏览器访问页面,访问网站的很多页面,访问完成后将浏览器关闭的过程称为是一次会话  本质就是多次请求多次响应  会话技术  Cookie:将数据保存到客户端浏览器  Session:将数据保存到服务器端 |

Cookie常用API:

|  |
| --- |
| 向浏览器保存数据response  void addCookie(Cookie cookie)  获得从浏览器带过来的Cookie,request  Cookie[] getCookies()  创建Cookies对象  new Cookies(String name,String value)  获取Cookie中的键  getName()  获取Cookie中的值  getValue()  设置Cookie的有效路径  setPath(String path)  设置Cookie的有效时间  setMaxAge(int maxAge)  Cookie的分类:  会话级别的Cookie:默认的Cookie.关闭浏览器器Cookie就会被销毁  持久级别的Cookie:Cookie在设定的有效时间内存在.  销毁持久级的Cookie:  Cookie cookie = new Cookie("history",null);  cookie.setPath("/day14");  cookie.setMaxAge(0);  response.addCookie(cookie);  对于setPath(String name)的理解:  Cookie c = new Cookie("name","123");  c.setPath(/day14/demo);  c.setMaxAge(60\*60);  response.addCookie(c);  创建一个Cookie,该对象对/day14/demo目录及其子目录是可见的,有效的.  即访问/day14/demo目录及其子目录时才会将该Cookie对象携带过去.才可以拿到该Cookie存储的值.  若访问/day14/RegtistServlet,该Cookie对象不能传递过去,也就拿不到该Cookie存储的值.  访问/day14/demo-login/同理 |

浏览器携带Cookie的条件：



Session:

|  |
| --- |
| http的缺陷  1.http协议的最大缺陷,无状态.客户端向服务端请求,完成相应完成就结束了.当第二次访问 时,服务器不知道客户端来过.使用session可以解决这个问题.sessionId.  2.不够安全  1.它是基于Cookie的!!!,SessionID是通过Cookie传递的  2.Session的获取  request.getSession()  3.Session存取和移除数据  \* setAttribute(String name,Object value);  \* getAttribute(String name);  \* removeAttribute(String name);  \*  3.Session作为域对象  session何时创建和销毁?作用范围:  \* 创建:服务器端第一次调用getSession()创建session.  \* 销毁:三种情况销毁session:  \* 1.session过期. 默认过期时间为30分钟.  \* 2.非正常关闭服务器.如果正常关闭session序列化到硬盘,再次开启session的文件会消失.  \* 3.手动调用session.invalidate();  \* 作用范围:多次请求.(一次会话)  注意区别：  session.invalidate()是让当前浏览器的session销毁，意味着session中的所有属性均失效。而removeAttribute()可以指定销毁session中的某个属性  4.例:  往Session中保存信息:  request.getSession().setAttribute("cart",map);  将Session中的信息清除:  request.getSession().setAttribute("cart",null);  5.例子:  当跳转的页面不变的时候,会优先查看本地缓存  解决方法:?time = new Date().getTime()  <img src="/day15/AServlet" />:这也是一种请求 |

##### Springboot中实现session共享

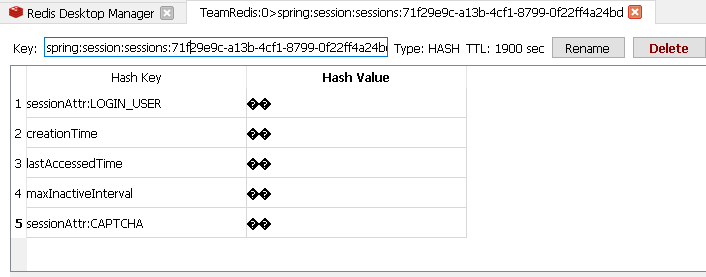
Springboot中使用redis来实现session共享

|  |
| --- |
| @Configuration  @EnableRedisHttpSession(maxInactiveIntervalInSeconds = 1800)  public class RedisSessionConfig {  } |

配置后，session中的信息会存入到配置的redis中。

Redis中的key为sessionid:

spring:session:sessions:sessionid(具体的值)



进行集群部署服务的时候。假如有两台服务器A,B。用户从A服务器中登录后。当调用

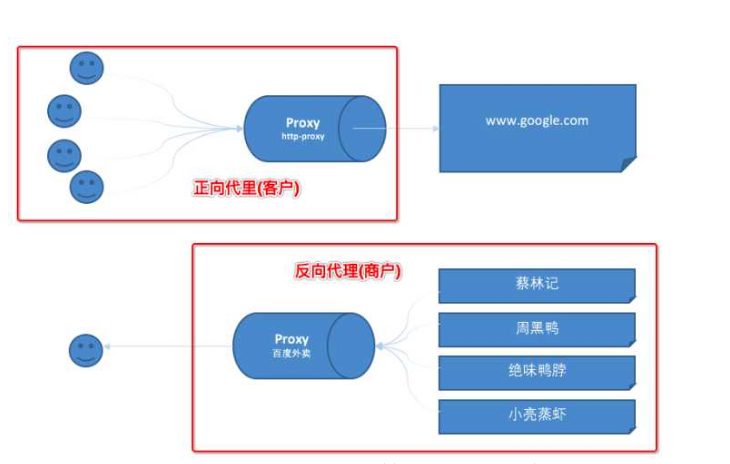
session().setAttribute(“LOGIN\_USER”, staffBO);信息会自动存入到redis中。用户的下一次请求可能请求到B服务器中，当调用session().getAttribute(“LOGIN\_USER”)B服务器也能从redis中取到staffBO的信息。即通过操作session来完成对redis的操作。

##### Nginx

Nginx (engine x) 是一个高性能的HTTP和反向代理服务

正向代理和方向代理：

1. 正向代理代理的是客户端，如翻墙
2. 反向代理代理的是服务端，如做集群时，nginx负载均衡来决定该请求由哪个服务端处理



Nginx实现负载均衡策略：

1. 轮询 默认配置，每个请求会按时间顺序逐一分配到不同的后端服务器
2. 权重方式 在轮询策略的基础上指定轮询的几率, weight参数用于指定轮询几率，weight的默认值为1,weight的数值与访问比率成正比
3. ip\_hash 基于客户端IP的分配方式，这个方法确保了相同的客户端的请求一直发送到相同的服务器，以保证session会话。这样每个访客都固定访问一个后端服务器，可以解决session不能跨服务器的问题
4. least\_conn把请求转发给连接数较少的后端服务器。轮询算法是把请求平均的转发给各个后端，使它们的负载大致相同；但是，有些请求占用的时间很长，会导致其所在的后端负载较高。这种情况下，least\_conn这种方式就可以达到更好的负载均衡效果
5. url\_hash 按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，要配合缓存命中来使用