

Week 12

day 5

内容

7天左右的课程，EE最核心的知识。

Maven、mvc

项目一 7天

提问问题

做一个独立的思考者。不要第一反应就去求助。

在提问的时候，应当带上自己的思考和理解在里面。

有没有自己去总结自己在开发过程中遇到的一些异常？

NullPointerException-空指针

ClassNotFoundException

Bug-list。面试的时候问你，你在开发过程中遇到哪些印象深刻的问题？

提问题的顺序：先思考、搜索引擎、求助于老师或者同学

提问问题应当尽可能去提供非常详尽的信息。包括报错截图。

体系梳理

JavaSE：个人版本

JavaEE：企业版本

对于一个企业来说关键的是：网站(http)、服务器(tomcat)、数据库(mysql)

tomcat

部署应用的方式

- 直接部署：直接在webapps下新建一个文件夹（要符合webapp的目录格式），这个文件夹就是一个应用，文件夹的名字就是应用名
- 虚拟映射：不希望直接在webapps下创建文件夹，但也想发布一个应用
(实际上在webapps下没有这个文件夹，但是相当于发在webapps下，所以叫虚拟映射)
 - conf/catalina/localhost下，新建应用名.xml文件（在里面的docBase指定具体的应用路径）
 - conf/server.xml：在host节点下，添加一个Context节点

直接访问localhost:8080，获得的是哪个文件：ROOT应用下的index.html文件（默认应用下的默认文件）

静态资源和动态资源

静态资源：一成不变的：html, css, js, 图片等

动态资源：具有可变性（即访问同一个url，响应报文里面的内容是会不断变化的）（最后还是要返回一个静态的）

servlet技术：用来开发动态web资源（如：刷新页面会显示当前的时间）

Servlet

概念

servlet = server + applet

- 运行在服务器端的小程序
- servlet中的代码供服务器调用（main方法再服务器的bin目录下）
- 用于开发动态web资源

文档

<http://tomcat.apache.org/tomcat-8.5-doc/servletapi/overview-summary.html>

servlet类只会实例化对象一次，也就是单例：每一个类对应一个对象

如何编写servlet

- 继承GenericServlet或者HttpServlet
- 编译成class文件，部署在应用中
- 配置对应的映射关系，供外部访问

过程：

- 写一个类extends GenericServlet或者 HttpServlet
- javac编译不通过：因为javax.servlet 不在jdk中，所以要通过 `-classpath jar包路径`，因为servlet是在web 服务器上运行的，所以tomcat的lib中一定有servlet-api.jar（把jar包往cmd里面拖）
- java执行不了，因为没有main方法，因为servlet是供服务器中的其他方法调用的，所以要放在服务器中才能运行

如何将class文件和tomcat关联

以发布应用的形式来实现，把class文件放入一个应用的classes文件夹中

应用根目录：（标准的JavaEE项目的目录结构）

----- 静态资源文件

-----WEB-INF(里面的东西不能被浏览器访问到)

-----classes (class文件)

-----lib (所依赖的包)

-----web.xml (应用配置)

怎么通过url运行这个servlet

通过虚拟映射, 当访问/serv时, 让tomcat通过name, 把/serv和所要执行的class文件联系起来, 反射创建对象, 并调用该对象的service方法 (url-pattern -> class)

localhost:8080/serv (因为应用名设置为了ROOT, 访问某个servlet的时候, 一定要记得看它的应用名是哪个)

```
<servlet>
  <servlet-name>first</servlet-name>
  <servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>first</servlet-name>
  <url-pattern>/serv</url-pattern>
</servlet-mapping>
```

直接sout, 输出的内容是直接到log中

如果想让内容输出到浏览器页面中: 内容应当写入响应报文的正文中

servletRequest对象和servletResponse对象就代表了请求报文和响应报文

通过res.getWriter().println("msg")

出现问题了, 看log

其他开发servlet的方式

IDEA中创建一个project, java enterprise, 设置server, 设置web application4.0

继承HttpServlet(虽然是抽象类, 但是里面没有抽象方法)(虚拟映射可以使用web.xml, 也可以使用注解) `@WebServlet("/first")`: 只写一个字符串就默认是设置ulrpattern

支持get方法, 就重写doGet, 支持post方法, 就重写doPost

A subclass of `HttpServlet` must override at least one method, usually one of these:

- `doGet`, if the servlet supports HTTP GET requests
- `doPost`, for HTTP POST requests

HttpServlet的service方法, 首先以父类的service方法为入口, 然后执行自己的service方法, 然后根据请求方法进行进一步的区分处理

所以分析servlet执行的时候, 要找到入口: service方法

或者直接create new servlet (推荐)

servlet执行流程

当用户在浏览器输入：`localhost:8080/app/servlet`

- 浏览器构建请求报文
- 通过网络传输到达目的主机，被监听8080端口的Connector接收到
- Connector根据请求报文生成request对象和一个response对象
- 将这两个对象转发给engine
- engine根据域名，转发给对应的host
- host根据url中的内部路径，转发给具体的context
- context也是根据url中的内部路径，并通过应用中配置的映射关系，如果存在就找到对应的servlet-class，如果不存在就返回404
- 如果是首次访问servlet，就会先调用init方法，再调用service方法，否则只会调用service方法
- service方法执行完毕，response对象被返回给connector
- connector根据response对象中的内容，构建一个响应报文
- 通过网络传输，响应报文发送的客户端，浏览器解析报文

IDEA和tomcat关联方式

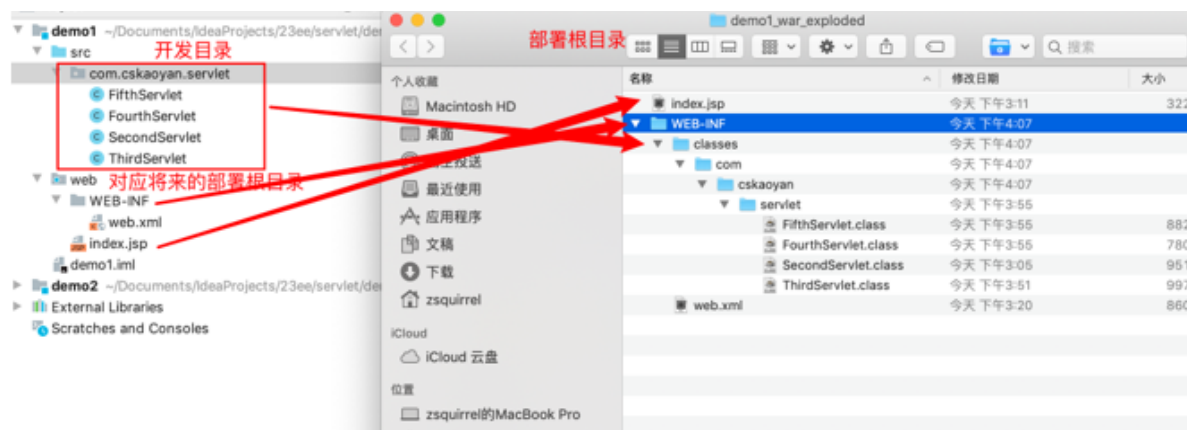
运行时，通过idea里面的log，可以看到运行日志

tomcat如何部署idea的ee项目

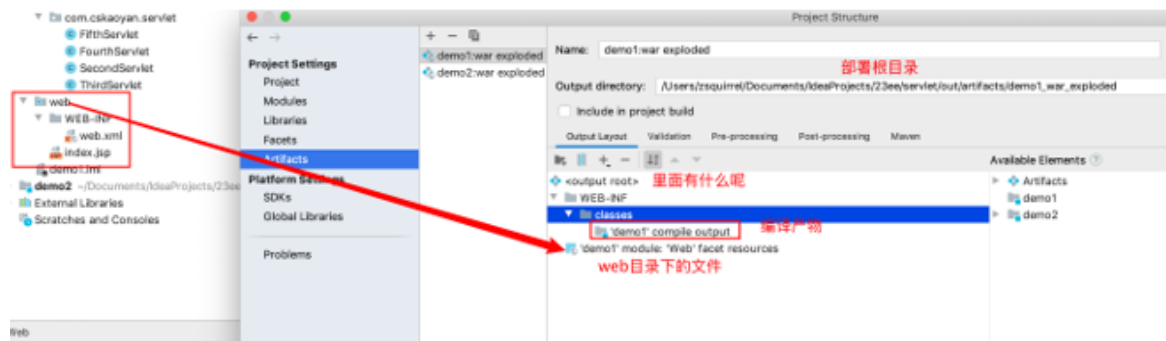
- IDEA复制了tomcat的一些核心配置文件，然后利用这些配置文件重新开启了一个新的tomcat(启动还是在原来的bin目录下)，利用这个tomcat来部署我们的应用（通过CATALINA_BASE可以看到新的tomcat的目录在哪）
- 新的tomcat目录找到应用名.xml文件（虚拟映射），通过docBase找到部署根目录（部署实际运行的应用放在这里）
- 总结：IDEA会复制tomcat的配置文件，然后到某一个目录下利用复制的配置文件重新开启一个新的tomcat来通过虚拟映射部署我们的应用

开发目录与部署目录之间的关系

开发目录和部署目录不是同一个目录，开发目录中存放的是.java文件，编译后会按照**某种规则**，将class文件以及web相关联的文件（web.xml, index.jsp等）全部复制到部署根目录中去



复制规则



常见问题：在开发目录下新建了一个1.html，然后访问显示404，

因为显示的东西是以部署根目录下面的内容为准，很可能是因为IDEA没有将开发目录中的文件同步到部署根目录中

- 重新部署（IDEA通过新的tomcat目录中的配置文件重新部署）
- build - rebuild project（重新编译）->重新部署
- 把部署根目录中的文件全部删了，然后再执行第二个方法

注意：ROOT应用 部署可以直接写 /

servlet生命周期

单例

- `init`：初次访问servlet时会执行，再次访问不会执行

可以通过在注解中设置一个`loadOnStartup`的参数，把它的值设为一个非负数，使得这个方法在应用被加载的时候就执行（如连接数据库，如果用户访问时再连接就比较慢了，所以先连接好）

```
@WebServlet(urlPattern/value = "/life", loadOnStartup = 1)
```

在注解中的servlet节点下设置也可以

```
<servlet>
  <servlet-name>life</servlet-name>
  <servlet-class>com.cskaoan.servlet.lifeCycle.LifeCycleServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

- `service`：每次请求来，都会执行
- `destroy`：应用被卸载(redeploy)或者服务器关闭

url-pattern

细节：

- 一个servlet可以对应多个url-pattern吗：可以的

```
@WebServlet(value = {"/ur11", "/ur12"})
```

- 一个url-pattern可以对应多个servlet吗：不可以

- url-pattern写法有哪些，必须以/开头吗
有两种，第一个种以/开头，第二种*.后缀

url-pattern优先级:

- /* 的优先级始终要高于 *.后缀
- 匹配程度越高，越优先执行

两个特殊的url-pattern

/* 和 /

- 设置了url-pattern为/* 的servlet之后，访问部署根目录下的index.jsp，和1.html都无法访问到
因为*.jsp有自己的servlet负责返回，而其他的静态资源由一个默认的servlet负责返回，自己写了url-pattern为/* 的servlet(而在里面又没有什么实现)，所以系统就不会调用原来负责的servlet返回静态资源（/* 的优先级高于*.jsp）
- 设置了url-pattern为/ 的servlet之后，可以访问部署根目录下的index.jsp（*.jsp对应的servlet生效了），但是仍然访问不到1.html，因为系统不会调用默认servlet帮忙处理了
(默认servlet处理逻辑：如果找到就通过流的形式写出，找不到就404)

结论：平时访问静态资源都是通过url-pattern为/ 的默认servlet，或者是url-pattern为*.jsp的servlet，让它们负责处理请求


默认servlet就是处理那些无家可归的请求

ServletConfig

可以获取servlet的一些初始化的配置参数（在具体的servlet节点下的）（了解即可）

- 配置参数

```
<servlet>
  <servlet-name>config</servlet-name>
  <servlet-class>com.cskaoyan.config.ConfigServlet</servlet-class>
  <init-param>
    <param-name>name</param-name>
    <param-value>zhangsan</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>config</servlet-name>
  <url-pattern>/config</url-pattern>
</servlet-mapping>
```



- 获取参数

```
ServletConfig servletConfig = getServletConfig();
String name = servletConfig.getInitParameter(s: "name");
System.out.println(name);
```

ServletContext

- 获取全局性初始化参数（了解） `getInitParameter()`
 - 直接在web-app节点下设置

```
<context-param>
    <param-name>name</param-name>
    <param-value>utf-8</param-value>
</context-param>
</web-app>
```

- 获取全局性参数（当前context下的任何一个servlet都可以获得该参数）

```
ServletContext servletContext = getServletContext();
String name = servletContext.getInitParameter(s: "name");
System.out.println(name);
```

- 共享数据的场所(`setAttribute()/getAttribute()`)

当前应用（context）下的两个servlet之间想要共享一些数据

- 先在一个servlet中设置数据（key-value）

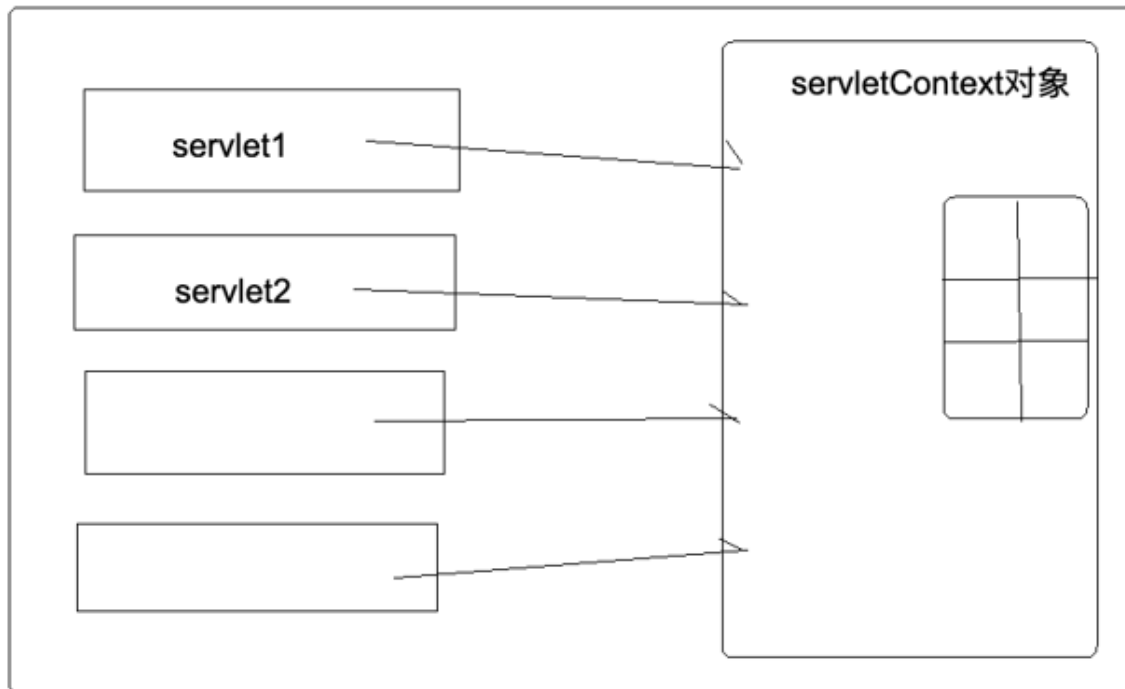
```
ServletContext servletContext = getServletContext();
//JDBC 商场的商品信息 List<Type> 电器 衣服 化妆品
servletContext.setAttribute(s: "name", o: "iphone");
//servletContext.removeAttribute("name");
```

- 其他servlet去获取数据

```
ServletContext servletContext = getServletContext();
String name = (String) servletContext.getAttribute(s: "name");
System.out.println(name);
```

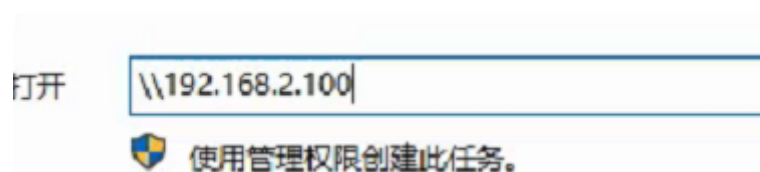
相当于在ServletContext对象中持有一个map

应用 Context



- 获取EE项目的部署路径(`getRealPath()`)
 - 原因：直接在servlet中 `new File("1.html")`，其中的相对路径相对的是tomcat的bin目录，因为EE项目没有main方法，代码是供tomcat调用的，**相对路径相对的目录就是调用VM的那个目录**。而文件都放在bin目录下不方便：可能会存在同名文件，项目打包不方便
 - 所以通过 `getServletContext().getRealPath("")`，就能够获得部署根目录，再加上相对路径就能够获得部署根目录下的文件夹的文件（也可以获取WEB-INF中的文件，因为WEB-INF是不允许浏览器访问，但是允许服务器访问）
- SE（整个tomcat，其中有main方法供VM调用） EE（枝叶）

局域网资源



Week 13

day 1

ServletRequest

接口（规范）：去看文档

获取请求报文中的信息

面向对象：ServletRequest就是对请求报文的封装

请求报文： 请求行 请求头 空格 请求体

请求行： 请求方法 URL 版本

- `getMethod`
- `getRequestURI`：获取标识符，服务器内部路径
`getRequestURL`：
- `getProtocol`

请求头：

- `getHeaderNames()`：返回一个枚举类型，获取所有的请求头的名字，可用来遍历（配合 `getHeader()`）
- `getHeader(String name)`：通过传入的请求头的名字，获取其值

请求体：

- `getInputStream()`：获取一个字节输入流

注意：

post的url（请求行）中也可以显示请求参数（表单数据）

所以get和post只有语义上的区别：获取数据和提交数据

常用API

- `getRemoteAddr()`：获取客户端的IP地址：有可能是IPV6的
- `getRemotePort()`：获取客户端的端口号
- `getLocalAddr()`
- `getLocalPort()`

获取请求参数

请求参数的值就是用户提交的数据

无论请求参数是在url中，还是在请求体中，都可以直接用的API

- `getParameter(String s)`：通过参数名，获取参数的值（字符串） username="daxiao"
- `getParameterValues(String s)`：通过参数名，获取参数的值（以字符串数组形式返回）

进阶查询

循环遍历的方式去处理请求参数

- 通过 `getParameterNames()`，获取一个枚举类（字符串类型）
- 在每一次迭代中，通过 `getParameterValues(String s)`，获取参数的值（通过枚举类的迭代器去做）
- 再根据字符串数组的长度进行进一步处理(==1 or else)

封装到Bean

Dbutils: BeanHandler BeanListHandler

Bean要求封装的类：

- 具有无参构造方法（反射调用，创建对象）
- 成员变量为private（为了封装性）
- 具有public的setter（反射调用，设置成员变量的值）

把请求参数中参数值封装到一个对象中的步骤

- 获取对象的Class对象
- 实例化出一个具体的对象
- 获取set方法对应Method对象
- 调用Method对象的invoke方法，为对象赋值

利用第三方工具类：BeanUtils

EE项目导包：

如何让tomcat在运行时能够找到该context对应的jar包：（tomcat就是一个main方法）

放在开发目录中的web-WEB-INF下的lib文件夹（没有就创建一个），然后add as library

请求参数中文乱码

post

原因：编解码不一致

- 浏览器发过来的数据编码格式：utf-8（html文件中）
- 服务器解码使用的是latin-1，欧洲码表，用一个字符用8bit表示，不支持中文

解决方法：将request对象中的解码格式也设置为utf-8(最好开始时直接设置)

```
void setCharacterEncoding(String env)
```

Overrides the name of the character encoding used **in the body of this request**. This method must be **called prior to** reading request parameters or reading input using `getReader()`.

- 该方法只能改变请求体中的的字符解码格式

- 且必须在读取请求体之前调用才生效

get

tomcat8处理请求行(url)时, 默认的编码格式为utf-8

所以获取get中的请求参数时不设置解码格式, 也不会遇到乱码

注意:

用debug 看乱码, 不要sout (没有参考价值),
光标到这一行的时候, 说明这行代码还没执行

路径写法

原来我们在form表单中写的路径 `http://localhost/app/submit`, 是全路径

全路径写法 (不太好)

背景: 需求会不断变更

企业中开发: 项目经理、产品经理、UI、前端、服务端、测试

开发: 在自己电脑写

测试: 在本地局域网内部署一个应用, 此时就不能用localhost了 (测试的客户机和服务端主机不是同一个机子的情况)

生产: 正式环境 (上线了后也不可能让用户访问用户自己的主机)

但是可以通过配置文件来弥补 (避免硬编码)

相对路径

相对于当前页面

如: 当前页面:

<http://192.168.4.40/app/form.html>

直接在 `action=` 中填“submit3”

提交的页面地址:

<http://localhost/app/submit3>

耦合性太高, 过分依赖于当前页面, 不推荐

以/开头的写法 (推荐, 即服务器内部路径)

/应用名/资源路径

/app/submit3

注意：

- 给浏览器看的要加/应用名
- 给服务器按的不用加应用名

转发和包含

背景：有一个请求需要多个servlet共同参与来处理

转发和包含区别：

相同点：都需要介导多个组件参与到一个请求的处理中（处理的都是响应报文）

不同点：主动权移交差异

- 转发:a 转发给 b
a是源组件，b是目标组件
a转发给b之后，主动权也交到了b的手中
- 包含：a 包含 b
a将b的内容包含到自身，主动权仍然在a手中
- 代码上的体现：
 - 转发：源组件留头不留体（针对响应头和体）(转发是post请求)
登录时：校验页面（处理逻辑）-> 跳转页面
 - 包含：源组件留头也留体
大型的页面：主体部分 main.html<-----下面的友情链接href.html

Dispatcher路径写法

- 全路径 `http://localhost/app/dispatcher2`
不行，前面会自动拼接context的路径（因为转发和包含都是在一个context下面进行的）
- 相对路径 `dispatcher2`
可以
- 以/开头 `/dispatcher2`
可以

口诀：（只适用于/开头的路径写法）

- 路径的执行主体是服务器：不用加应用名（就在一个context下）
两种情况：
- 路径的执行主体是浏览器，要加（浏览器中的form, a, img中的属性）

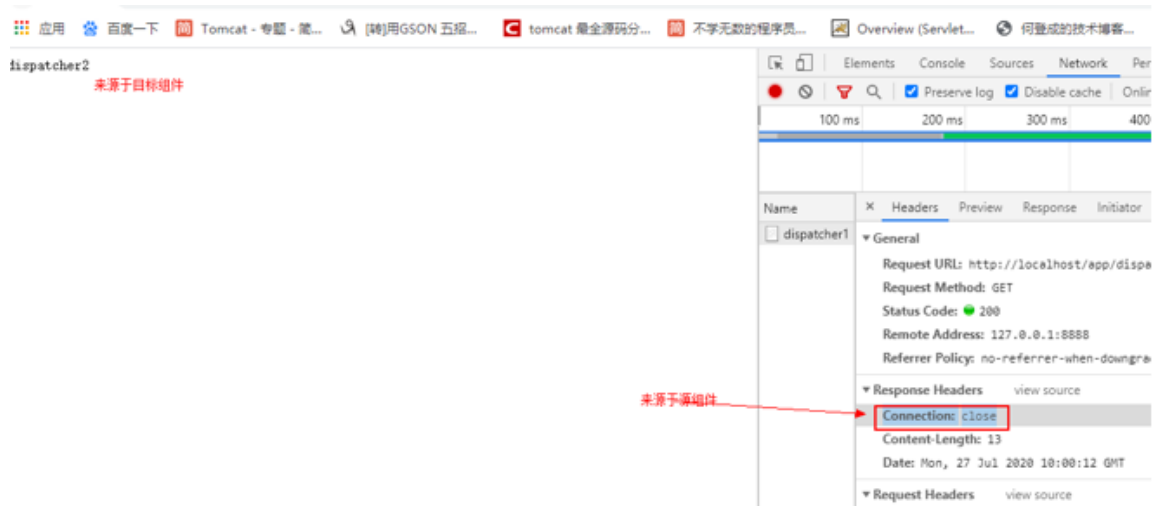
代码体现

转发：

```
// dispatcher1
response.setHeader("Connection", "close");
response.getWriter().println("write in dispatcher1");
RequestDispatcher dispatcher = request.getRequestDispatcher("dispatcher2");
dispatcher.forward(request, response);

// dispatcher2
response.getWriter().println("write in dispatcher2");
```

显示：留头不留体，源组件中：响应头的内容得以保留，响应体中的内容被清空



包含：

```
// dispatcher1
response.setHeader("Connection", "close");
response.getWriter().println("write in dispatcher1");
RequestDispatcher dispatcher = request.getRequestDispatcher("dispatcher2");
dispatcher.include(request, response);

// dispatcher2
response.getWriter().println("write in dispatcher2");
```

显示：留头也留体，源组件中：响应头和响应体中的内容得以保留



Request域

跟context域一样，request对象中也维护了一个map(String -> Object)

Request域：可以拿到同一个request对象的多个组件（filter, servlet），就可以共享request对象中的map

生命周期：

context域的生命周期与context（应用）保持一致(从应用的加载到卸载)

request域的生命周期很短，只存活在一次请求中

范围：转发和包含的源组件和目标组件之间可以共享

问题：地址栏中输入一个地址，反复刷新，这些request对象是同一个吗

不是，每一个http请求对应一个request对象

使用场景：

在一个servlet中通过JDBC查询到了List数据信息，就可以先把数据放入request域，再通过转发传到jsp页面（也是一个servlet）

前后端分离后：传数据给前端就行了（页面跳转给前端做）

day 2

ServletResponse

响应报文的封装：HttpServletResponse（如果发送的是http请求，那么就能在重写父类的service方法中顺利转成）

响应报文：版本 状态码: setStatus 原因短语

响应头： setHeader(key, value)

响应体： getWriter().println()

自己设置404响应报文：

```
response.setStatus(404);
response.getWriter().println("<h1 style=\"text-align: center; color: red\">404</h1>\n" + "<h1 style=\"text-align: center; color: red\">File Not Found</h1>");
```

输出字符数据到客户端

服务端编解码都是用latin-1

输出中文会乱码：

- 服务器编码用的latin-1
- 而浏览器默认的解码格式为操作系统默认的GBK

解决方式：

- 服务器端自己编码时使用一种支持中文的字符集
- 并且让客户端解码时也使用这种字符集

-> 如何把服务器使用的字符集告诉客户端：通过响应头、响应体（html中的head中的charset）

- 乱码解决方案一（在响应头中告知字符集）

第一条语句：①告知客户端响应体中的文件类型，②设置服务器端自己的字符集，③以及让客户端也使用这种字符集

```
response.setHeader("content-type", "text/html; charset=utf-8");
response.getWriter().println("你好")
```

- 乱码解决方案二（在响应头中告知字符集，同一，只是API变了）

```
response.setContentType("text/html; charset=utf-8");
response.getWriter().println("你好");
```

- 乱码解决方案三（在响应体中告知字符集）（jsp原理，在响应体中写html）

```
response.setCharacterEncoding("utf-8");
response.getWriter().println("<!DOCTYPE html>
<html lang='en'>
<head>
  <meta charset='UTF-8'>
  <title>Title</title>
</head>
<body>
你好
</body>
</html>")
```

输出字节数据到客户端

传输二进制文件：音视频、图片、文档

就是普通的字节IO流

页面跳转的另外两种方法

- 定时刷新

利用refresh这个响应头，让浏览器刷新当前页面或者跳转到一个新的页面

```
// 1. 每隔1s刷新一个次
response.getWriter().println(new Date());
response.setHeader("refresh", "1");

// 2. /app 可以用 request.getContextPath() 获得
response.setHeader("refresh", "1;url=/app/1.html")
```

- 重定向

设置状态码：302 307

设置响应头Location

```
// 1.
response.setStatus(302)
response.setHeader("Location", "url")

// 2. 简化版
response.sendRedirect("url")
```

页面跳转比较

转发与定时刷新、重定向

- 联系：都可以用来实现页面跳转
- 区别：
 - 转发时浏览器只发送了一次http请求；其他的是多次请求
 - 转发执行主体是服务器；其他执行主体是浏览器
 - 转发只可以在应用内跳转；其他不受限制
 - 转发是request对象介导；其他的是response介导
 - 转发可以共享request域；其他不可以
 - 重定向状态码302、307；其他都是200（所以不能用404和刷新（200段）或者重定向（300段）搭配）
 - 转发地址栏不发生变化；其他会发生变化

下载

浏览器默认行为：

- 对于可以打开的文件：html, txt, css, js, jpg等，默认会帮你执行**打开**操作
- 对于无法打开的文件：audio, video, exe, zip，默认会帮你执行**下载**操作
(对于浏览器不认识格式的文件也会默认下载)

下载功能：对于那些浏览可以打开的文件，如果不想浏览器默认打开，而是下载，就可以设置下载响应头

```
response.setHeader("Content-Disposition", "attachment;filename=1.jpg");
```

应用场景：管理系统中的统计报表的导出功能，导出就会用到下载响应头

注意：如果text/html类型的文件没有设置编码格式，因为服务器编码默认用的latin-1，

所以浏览器不认识响应体中的内容，无法解析，就只能下载，

所以要先设置编码格式，并且让客户端使用该格式解码

```
// servlet中只要设置这两个地方，就不会出现中文乱码
request.setCharacterEncoding("utf-8");

// 注意如果text/html 写成 test/html 写错的话，浏览器不知道怎么解析这种类型，就会直接下载
// 如果在响应体中写入一个图片，让它以text/html形式解析，就会产生乱码
response.setContentType("text/html;charset=utf-8");
```

FileUpload

文件上传：如何把一个文件上传到服务器

- 将文件装到请求报文的请求体
- 服务器解析请求报文，取出请求体IO流

准备工作：form表单， `input type=file method=post`

```
POST http://localhost/app/upload HTTP/1.1
Host: localhost
Connection: keep-alive
Content-Length: 11
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.89 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/app/upload.html
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=18F279D57BA2DE497F12D45730D4ABAE; Idea-83cde6a5=014bf777-5194-49e4-923c-2f4a24d6bff9
image=1.jpg
```

关注问题

问题一

只会上传文件名，不会上传文件的二进制数据 (img=1.jpg 还是key=value (参数名=参数值) 这种数据结构)

解决方案: form表单添加: `enctype=multipart/form-data`

Form 表单添加 `enctype=multipart/form-data`

```
POST http://localhost/app/upload HTTP/1.1
Host: localhost
Connection: keep-alive
Content-Length: 27485
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://localhost
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryGDK33ErHB1FbvYlK
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/app/upload.html
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=18F279057BA2DE497F12D45730D4ABAE; Idea-83cde6a5=014bf777-5194-49e4-923c-2f4a24d6bf9
-----WebKitFormBoundaryGDK33ErHB1FbvYlK
Content-Disposition: form-data; name="image"; filename="1.jpg"
Content-Type: image/jpeg

00000000JFIF0000000000000000C0000 000 0
```

问题二

通过 `request.getInputStream()` IO流操作后，发现文件以及损坏，图片无法打开

试着上传txt文件

```
-----WebKitFormBoundaryCAPvGbgPj7LG0mcI  
Content-Disposition: form-data; name="image"; filename="hello.txt"  
Content-Type: text/plain  
  
hhhhhhhhhhhhhhhhhhh  
-----WebKitFormBoundaryCAPvGbgPj7LG0mcI--
```

发现会多出一个分隔符，这些分割符放在二进制文件中，就会导致文件损坏

问题三

普通form表单数据和文件同时上传时，通过 `request.getParameter("name")`，发现返回的是 `null`，

即原先可以获取请求参数的API不能再使用了（其他的API也不能用了）

根本原因：添加了 `enctype=multipart/form-data` 之后，请求体中数据的数据结构发生了变化

之前是 key=value

之后是

```
-----WebKitFormBoundarysrZHqtsSnnLGjAI4
Content-Disposition: form-data; name="username"

admin
-----WebKitFormBoundarysrZHqtsSnnLGjAI4
```

处理这个问题

导包，用第三方工具类

commons-fileupload

<http://commons.apache.org/proper/commons-fileupload/>

中文乱码问题：对于两种上传数据类型

- 普通form表单数据中中文乱码问题的解决

```
String value = fileItem.getString("utf-8");
```

- 上传的文件名的中文乱码问题

```
upload.setHeaderEncoding("utf-8")
```

设置上传文件的大小（服务器存储资源很宝贵）

```
// 设置单个上传文件的大小上限，字节为单位  
upload.setFileSizeMax(1024)
```

用户注册：普通form表单数据、头像（在数据库中存储地址即可）

day 3

debug

EE项目：通过访问某个url地址，让代码被执行到，才能到达断点

在程序的入口打断点：service方法，doget，dopost方法

EE关于IO的代码的报错异常都要去部署目录看，不要看开发目录(注意看部署的是哪个项目)

- step over:一行一行地走，不进入方法
- step into:一行一行地走，进入方法
- step out: 从方法中跳出来
- resume program:继续程序，直到下一个断点或者执行完剩下的代码

代码优化

创将一个用来解析request对象，返回一个map对象的工具类的方法（map对象留着给beanutils用）

文件上传同名问题

- 加随机字符

```
String fileName = fileItem.getName();
String randomStr = UUID.randomUUID().toString();
fileName = randomStr + "-" + fileName;
```

- 加时间戳
- 加年月日时间段

解决同一目录下文件数过多的问题

硬盘下某个目录的文件很多加载的时候会很卡

——> 分散文件夹：根据年月日：但是会不均匀

头像尽可能均匀分散：散列，hashCode

根据文件名进行散列：asdasd.jpg -> hashCode(32bit) -> 0x1234abcd

```
// 用stringbuilder拼接字符串，因为string对象拼接时会不断产生垃圾
StringBuilder builder = new StringBuilder();
// 通过string类重写的hashCode方法，获取文件名对应的哈希码值
int hashCode = fileName.hashCode();
// 获取哈希码值对应的16进制字符串对应的字符数组，32bit -> 8个16进制数
char[] chars = Integer.toHexString(hashCode).toCharArray();
// 上传的文件放在部署根目录下的storage文件夹中
builder.append("/storage");
// 拼接目录
for (char c : chars) {
    builder.append("/").append(c);
}
// 拼接文件名，获得相对路径（相对于部署根目录）
String relativePath = builder.append("/").append(fileName).toString();
System.out.println("relativepath: " + relativePath);
// 获取部署目录下的文件路径
String realPath = request.getServletContext().getRealPath(relativePath);
System.out.println("realPath: " + realPath);
File file = new File(realPath);
// 如果文件的父目录不存在，则先创建父目录
if (!file.getParentFile().exists()) {
    // System.out.println("arrive here");
    boolean mkdirs = file.getParentFile().mkdirs();
    System.out.println(mkdirs);
}
// 将item中封装的文件内容，写到部署目录下
item.write(file);
// map中存放文件类型和文件的以/开头的路径（根据需求key也可以改成文件名）
nameToValue.put(fieldName, relativePath);
```

案例

用户注册，然后将信息进行回显在浏览器窗口，图片需要显示出来

upload.html -> servlet (将表单数据和上传的文件路径，封装到bean中) -> JDBC保存到数据库 (保存到某个域中)，跳转到某个页面显示出刚刚的信息

day 4

会话技术

会话能够正常进行的前提：上下文

web访问中：一个标准的http请求是没有状态的，也就是没有上下文

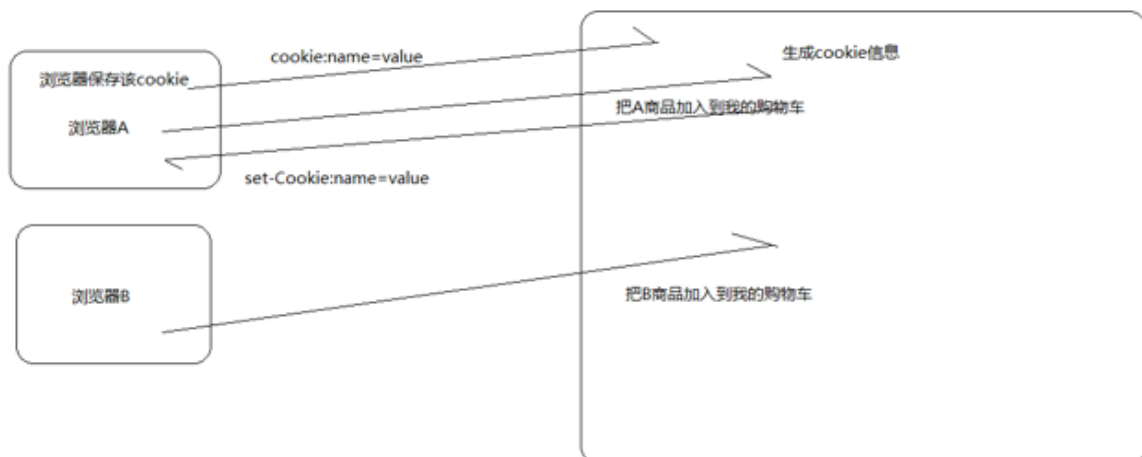
http无状态：http请求在服务器看来是一样的，没有任何区别

因为http无状态，所以它无法进行一个会话的管理

为了让服务器实现会话管理，需要引入一些会话技术：cookie，session

引入会话技术的目的：可以在浏览器存储一些数据：购物车信息、浏览记录、存储登录状态

cookie



服务器端需要做的事情：

- 生成cookie信息 `new Cookie(String name, String value)`
- 将cookie以set-Cookie响应头的形式发送出去 `response.addCookie()`
- 浏览器再次请求时，服务器需要把cookie请求头中的数据取出来 `request.getCookies()` for-each循环

因为cookie的数据存储在客户端，安全性较低，所以不会存储敏感数据

案例：显示上次访问当前页面的时间

```
// cookie中的value值为String类型，所以需要先将其转化为long类型才能调用Date的构造方法
Date date = new Date(Long.parseLong(value));
```

cookie设置

设置保存时间：

```
cookie.setMaxAge()
```

如果没有设置，则默认为负数：表示仅存于浏览器内存中，关闭浏览器则失效

设置正数：表示cookie可以在硬盘中存活多少秒

关闭浏览器后再打开页面，发现请求头cookie中的相关数据，证明的确实存在了硬盘中

删除cookie：

设置cookie的maxage=0，表示删除该cookie

注意：

- 删除cookie时，如果之前没有设置path，那么删除时也不要设置
- 如果之前设置了path，那么删除的时候也要设置才能生效

```
cookie.setMaxAge(0);
cookie.setPath(request.getContextPath() + "/path2")
```

设置path：

场景：希望只访问某些路径时携带cookie，其他路径不携带，这时候可以设置path

是服务器写给浏览器看的，让浏览器只在访问path中的路径时携带cookie请求头

```
cookie.setPath(request.getContext() + "/path2")
```

设置domain：

表示这个cookie归属的域名

```
cookie.setDomain()
```

原则：不可以设置无关的域名，浏览器不允许

比如localhost设置了一个baidu.com的域名，出于安全性考虑，不会设置成功

▼ Ret This Set-Cookie was blocked because its Domain attribute was invalid with regards to the current host url.

跨域：cookie不允许保存和发送

localhost:8080 与 localhost:80 之间也算跨域

父子域名之间存在着一个规则：

Baidu.com-----一级域名

Video.baidu.com---二级域名

Story.video.baidu.com--三级域名

父域名的cookie在子域名中均可以拿到（提高用户体验）

session

属于服务端技术，服务器给用户浏览器开辟一块内存空间存放session对象，那么这个session对象就是专门给某个浏览器服务的。

关联方式：（每个浏览器都对应一个session对象，所以

`request.getSession().getAttribute("username")`）时，每个浏览器得到的都是不同的username

- session对象的id值通过set-Cookie响应头发送给浏览器，浏览器保存JSESSIONID
- 浏览器再次访问时，会携带该ID，服务器取出ID，然后找到这个ID对应的session对象

session的使用：

创建session对象

- `getSession()`：
判断当前请求是否携带JSESSIONID，有就返回ID对应的session对象，没有就创建一个
- `getSession(boolean create)`
判断当前请求是否携带JSESSIONID，有就返回ID对应的session对象，
没有就再判断create是否为true，是就创建，为false就返回null

session的执行流程

第一次访问，遇到了 `request.getSession()`，如何判断有没有对应的session对象：

判断请求头中有没有 cookie: JSESSIONID=...，有就返回，没有就创建一个session对象，同时将该session对象的id值通过 set-Cookie 响应头发送给客户端（只在第一次创建的时候发送，因为之后就存储在浏览器的cookie里面了）

第二次访问，遇到了上面那行代码，客户端此时已经携带了cookie: JSESSIONID=...，所以直接将关联的session对象返回

返回的session对象是给当前浏览器服务的，一个浏览器访问多个servlet，那么就可以在session中进行数据的存取

session域：每一个浏览器有一个

context域：自始至终只有一个

request域：每一个请求就有一个

登录案例：

利用session域，在filter中判断访问某个页面是否需要先登录，

如果需要再进一步从session中获得数据判断用户（浏览器）是否已经登录

问题

- 关闭浏览器，session会销毁吗

不会

还可以取到原来的数据吗：不可以，因为JSESSIONID是存储在cookie中的，关闭浏览器后cookie也随之消失了（cookie默认maxage为负数），就相当于没有钥匙了

如何在浏览器关闭后，还可以拿到原先的数据：

new一个与JSESSIONID同名的cookie，设置它的maxage为正数，将原来的覆盖掉

- 关闭服务器，session会销毁吗

会，session对象肯定会被销毁，但是session对象在应用被停止时被序列化到了硬盘中，再次启动应用时，又会反序列化回来（创建一个新的session对象，但是对象的内容相同）

（序列化文件在catalina base中）

```
org.apache.catalina.session.StandardSessionFacade@665b0e5d ← 打印session的地址
11A082B91186A231DCD5FC0F3B07208C ← 打印session的id值
admin ← 取出session里面的数据
30-Jul-2020 15:00:51.723 唯一标识 [http-nio-80-exec-49] org.apache.tomcat.util.descriptor.web.WebXml.set
30-Jul-2020 15:00:52.118 洪℃佬 [http-nio-80-exec-49] org.apache.jasper.servlet.TldScanner.scanJars At
org.apache.catalina.session.StandardSessionFacade@1cd311bd ← 新的session地址
11A082B91186A231DCD5FC0F3B07208C ← 新的session id
admin ← 新的session数据
```

配置tomcat的manager

本地安装的tomcat里面的webapps目录manager应用不要删除。

Tomcat的conf/tomcat-users.xml文件中配置节点

输出/manager 再输入用户名和密码即可访问

注意：

如果是在IDEA中用redeploy重新启动，则不会反序列化，因为IDEA在每次redeploy的时候，都会到tomcat目录中去复制对应的配置文件，然后把原先的配置文件删除，此时 session.ser 文件被删除了（可以用tomcat部署IDEA应用（虚拟映射））

session的生命周期

session对象的创建：request.getSession()

关闭服务器会销毁session对象，但是里面的数据不会丢失

关闭浏览器，session对象和里面的数据都不会受到影响，只是处于访问不到的状态（需要请求头中携带JSESSIONID）

session可以设置一个有效时间：有效时间内没有被访问，数据自动被销毁

tomcat中默认的时间是30min（每访问一次重新计时）

web.xml中：


```
<!-- ===== Default Session Configuration =====>
<!-- You can set the default session timeout (in minutes) for all newly
<!-- created sessions by modifying the value below.

<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

想主动销毁session中的数据：`session.invalidate()`，只是让session里面的map失效，而非让这个session对象变为null

所以销毁session里面的数据只有两种方式：

- 有效期到
- 主动调用 `invalidate()`（注销账户中会用到）

session依赖于cookie，如果cookie禁用怎么办

发送给客户端的url都重写一遍，这样SESSIONID会当成请求参数附加在url后面

- `response.encodeRedirectURL(java.lang.String url)`
- 用于对sendRedirect方法后的url地址进行重写。
- `response.encodeURL(java.lang.String url)`
- 用于对表单action和超链接的url地址进行重写

day 5

JSP

Java Server Page：动态页面解决方案

java代码必须放在特殊的标签中才行

jsp原理

jsp本质上就是servlet，寻找一下存在的痕迹：catalina base -> word下，找到了index.jsp对应的java和class文件

在地址栏输入xxx.jsp/ index/jsp时，发生了什么事情

- 请求会交给*.jsp 这个servlet处理，这个servlet称为jsp引擎
- jsp引擎会按照一定的语法，将jsp页面转成对应的java文件
- java文件编译成class文件
- 执行service方法(因为是个servlet)

为什么要有servlet和jsp分为两种东西：

servlet即作为逻辑处理，又要显示页面，难以维护，耦合性强

所以servlet专门负责逻辑处理：校验、JDBC、放入request域 转发

jsp专门负责显示页面：request域、session域、context域

jsp语法

jsp表达式:

```
<%= new Date()%> 不能加;
```

-> 在java文件中

```
out.print(new Date()); // 往响应体中写数据
```

jsp脚本片段: 原封不动放到java文件中

```
<% sout("hello jsp"); %>
```

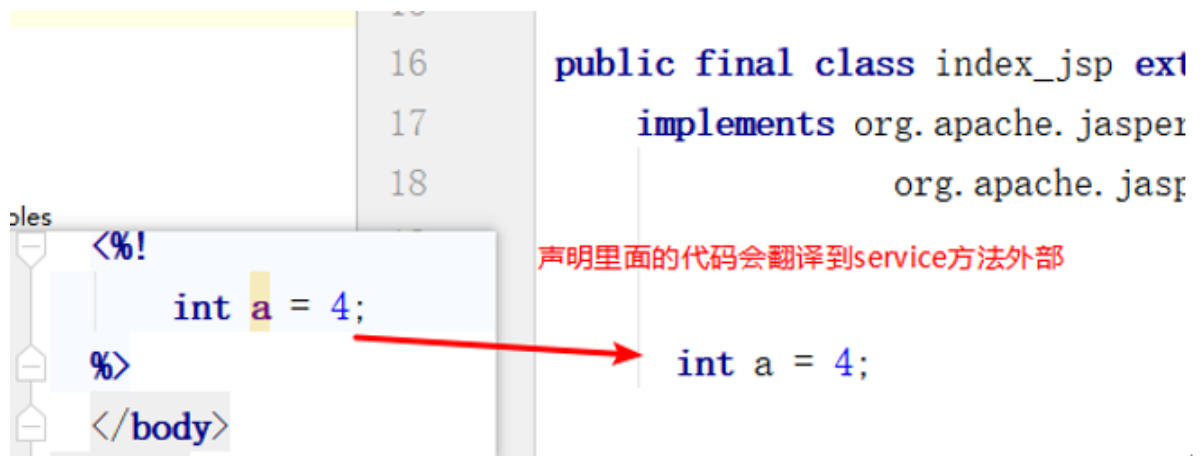
-> 在java文件中

```
sout("hello jsp");
```

注意: 每一个jsp脚本片段可以是不完整的, 但是最终拼接起来需要符合java语法

jsp声明:

放到service方法外 (类似于成员变量, 成员方法)



jsp注释:

jsp注释: `<%-- --%>`, 到java文件就没了

html注释: `<!-- -->`, 一直到html源码都有

jsp九大隐式对象 (在service方法中)

- request
- response
- **pageContext**
- session
- exception
- application
- config
- **out**
- page

pageContext是另外一个域: page域, 仅在当前页面有效

非常核心的一个对象, 可以通过它获取其他八大对象

pageContext

可以给其他域赋值:

```
//pageContext.get  
pageContext.setAttribute("scope", "page");  
pageContext.setAttribute(s: "scope", o: "request", PageContext.REQUEST_SCOPE);  
pageContext.setAttribute("scope", "session", PageContext.SESSION_SCOPE);  
pageContext.setAttribute("scope", "application", PageContext.APPLICATION_SCOPE);
```

findAttribute:

从pageContext -> request -> session -> application

按照这四个域开始依次去找, 找到则结束, 找不到则继续前往下一个域, 直至最后

注意: index.jsp service方法中也会创建一个session

out

是一个带缓冲去的Writer, 缓冲区满了才写入response

可以设置page指令的buffer属性为none关闭该缓冲区

三大Web组件

Servlet, Listener, Filter

Listener

Web开发过程中监听器：

在servletcontext对象创建和销毁时触发监听方法

- 监听对象：ServletContext
- 监听事件：创建和销毁
- 监听者：自己编写的监听器
- 触发事件：触发监听器里面的方法

如何编写一个listener

- 编写一个类实现ServletContextListener接口
- 注册该listener(注解， web.xml)

怎么做到的

接口 + 多态

伪代码：

```
class ServletContextXX {  
    // 将自己编写的listener注入到接口引用中  
    ServletContextListener listener;  
  
    void create() {  
        // context对象创建时，调用持有的该listener对象的该方法  
        listener.contextInitialized();  
    }  
  
    void destory() {  
        // context对象销毁时，调用持有的该listener对象的该方法  
        listener.contextDestroyed();  
    }  
}
```

使用场景：初始化配置， 作为Spring程序执行的入口

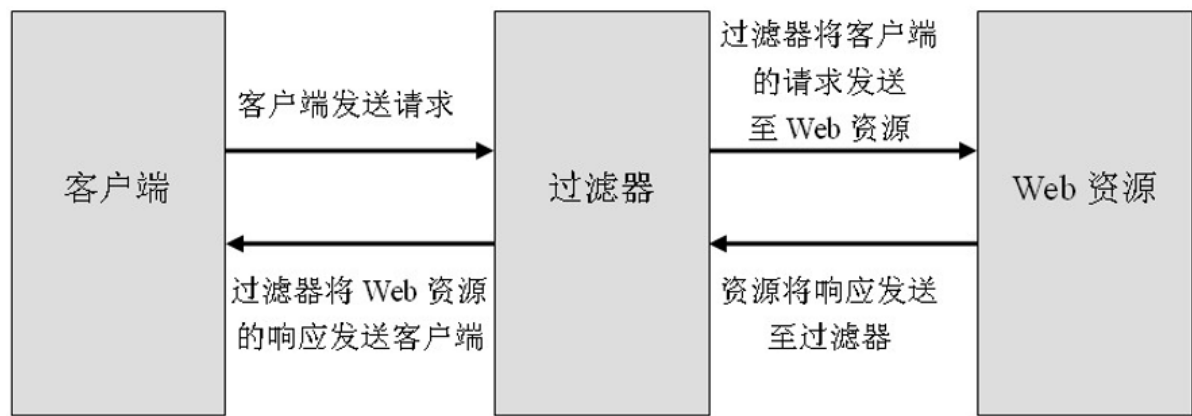
Filter

过滤器

如游戏中屏蔽关键字： ****

也是在tomcat里面

每个http请求都需要经过Filter



如何编写filter

- 编写一个类实现Filter接口
- 声明当前filter(web.xml, 注释):

```
<filter>
  <filter-name>first</filter-name>
  <filter-class>com.daxiao.FirstFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>first</filter-name>
  <url-pattern>/filter</url-pattern>
</filter-mapping>
```

```
@WebFilter("/filter")
```

如何让Filter和Servlet产生关联

通过url-pattern, 将servlet的url-pattern赋值给filter

默认情况下filter执行的是拦截操作, 如果想要filter放行, 那么需要

```
filterChain.doFilter(req, resp)
```

*filter的url-pattern可不可以设置/*呢*

可以设置为/*, 可以用来做设置编码格式和登录拦截 (用session)

一个url-pattern可以对应多个Filter吗

可以

一个url-pattern不能对应多个servlet (如果可以, 没法确定调用哪个)

一个url-pattern可以对应多个filter

从功能上分析:

- servlet: 开发动态web资源, 处理请求作出相应
- filter: 过滤器, 执行过滤任务的, 如果一个url-pattern对应多个filter, 先后调用即可

一个url-pattern对应多个Filter, 先后执行顺序如何确定

对于web.xml来说，以mapping声明的先后顺序为准

如果是注解，按类名首字母的ASCII先后顺序为准

如果web.xml和注解同时都配置，先xml后注解

整个请求的执行流程

到达应用context之前的步骤都不会变化，区别在于到达应用之后

- 浏览器地址栏输入 `http://localhost:8080/app/servlet`，浏览器构建请求报文
- 到达目的主机后8080端口接收，处理请求报文，生成request对象和一个response对象
- 两个对象传递给engine，挑选host继续处理
- 继续传递对象给context，根据配置的filter的url-pattern，如果有匹配的filter，就将其加入到队列中，如果有多个filter匹配到，就按规则确定执行顺序
- 查找匹配的servlet，如果有也加入到队列中（没有就执行默认的servlet）
- 依次调用队列中的组件，然后依次传递request, response对象给组件
- 依次执行完毕，一级一级地返回
- connector读取response里面的数据，生成响应报文，返回

Filter案例

JDBC原始的执行过程

```
class.forName(com.mysql.jdbc.Driver);
connection conn = DriverManager.getConnection(url,user,password)
prepareStatement psmt = Conn.prepareStatement(sql);
psmt.setInt(1,xxx);
psmt.setString(2,xxx);
ResultSet rs = Psmt.executeQuery();
while(rs.next()){
    Rs.getString(columnLabel);
    Rs.getInt(columnLabel);
}
conn.close();
psmt.close();
rs.close();
```

为什么要引入数据库连接池

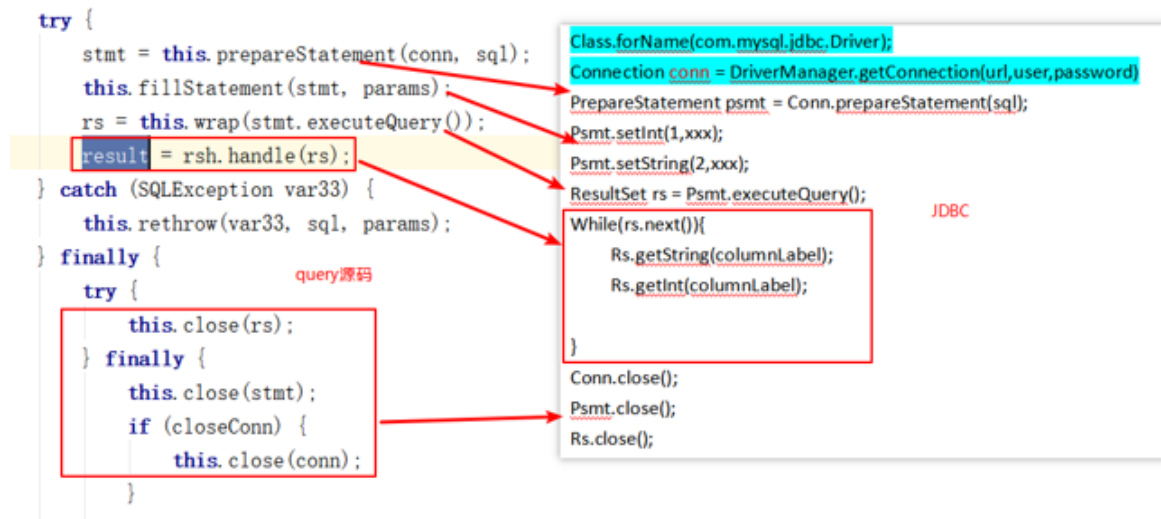
跟引入线程池的原因一样，因为创建连接的过程是十分消耗系统性能的，频繁创建和销毁连接，极易容易引起系统宕机。所以先预先创建好，需要的时候就取出一个来用，用来不关闭，而是直接放回连接池

`DataSource` 数据源， `getConnection()`

包装设计模式：重写close方法，让它把连接放回到连接池，而非直接关闭

```
// Dbutils: JDBC的简单封装
QueryRunner runner = new QueryRunner(ds);
BeanListHandler
List<Bean> beans = runner.query(sql, rsh, params);
```

query方法源码



rsh:

- BeanListHandler: 将结果封装为list对象形式
- BeanHandler: 将结果的第一行数据封装为一个对象
- ScalarHandler: 用来查询只有一列的数据 (聚合函数 `count max min avg`)

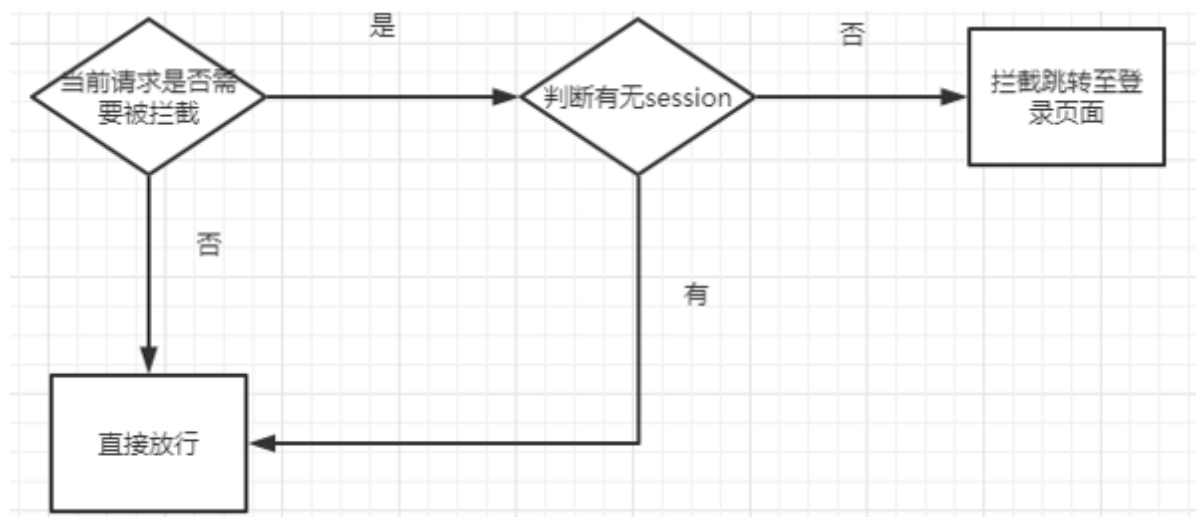
(scalar:标量)

配置文件的存放位置

EE项目配置文件放在哪里?

与最外层package目录同级通过类加载器获取 `inputstream`

`aclass.class.getClassLoader().getResourceAsStream("filename")` 获取src目录下的文件



Week 14

day 1

MVC

JSON

前后端交流：json

{ }：表示一个对象

[]：表示一个集合、数组

js里面的json对象表示方法

在数据传输中一般用的是json字符串：就是Java中的 Map<String, Object>

json对象和json字符串的区别在于属性值加上引号

json对象：

```
var country = {name:"中国", province:[{name:"黑龙江", cities:["哈尔滨", "大庆"]},  
                                         {name:"广东", cities:["深圳", "广州"]},  
                                         {name:"台湾", cities:["台北", "高雄"]},  
                                         {name:"新疆", cities:["乌鲁木齐"]}]}
```

json字符串：

```
var pro = {  
  "name": "中国",  
  "province": [{"name": "黑龙江", "cities": ["哈尔滨", "大庆"]},  
               {"name": "广东", "cities": ["广州", "深圳", "珠海"]},  
               {"name": "台湾", "cities": ["台北", "高雄"]},  
               {"name": "新疆", "cities": ["乌鲁木齐"]}]  
}
```

Java语言操纵json字符串

- json字符串转成Java对象

场景：前端传输过来一个商品的信息，需要将这些数据保存到数据库

```
String user =  
String users =  
Gson gson = new Gson();  
// 字符串中只有单个对象：可以利用try-catch来看String -> Integer有没有失败，以检测传入  
的参数  
User u = gson.fromJson(user, User.class);  
  
// 字符串中是一个对象数组（集合）
```



```

JsonElement jsonElement = new JsonParser().parse(users);
JsonArray jsonArray = jsonElement.getAsJsonArray();
List<User> list = new ArrayList<>();
for (JsonElement element : jsonArray) {
    User u = gson.fromJson(element, User.class);
    list.add(u);
}

```

- Java对象转成json字符串

场景：查询某个商品，返回给前端的商品信息。Java对象转成响应的json字符串

工具包：google: Gson, alibaba: fastjson, SpringMVC: jackson

```

String json = gson.toJson(user);
String jsons = gson.toJson(users);

```

bejson.com：判断是否是json格式的网站

mvnrepository.com：找包的网站

add as library相当于加入到classpath

MVC

当需求变更时，从一种实现方式变更为另外一种实现方式，如果需要改动的代码很多，那么就是有问题的（耦合性太高了）

原来的代码：注册案例，存储到json中

```

String jsonStr = outputStream.toString(Charset.forName("UTF-8"));
List<User> users = new ArrayList<>();
Gson gson = new Gson();
if(!StringUtil.isEmpty(jsonStr)) {      model的内容
    //之前有人注册过 List<User>
    JsonElement jsonElement = new JsonParser().parse(jsonStr);
    JsonArray jsonArray = jsonElement.getAsJsonArray();
    for (JsonElement element : jsonArray) {
        User u = gson.fromJson(element, User.class);
        if(u.getUsername().equals(username)) {      view的内容
            response.getWriter().println("当前用户名已经存在，请更换用户名");
            return;
        }
    }
    users.add(u);
}

```

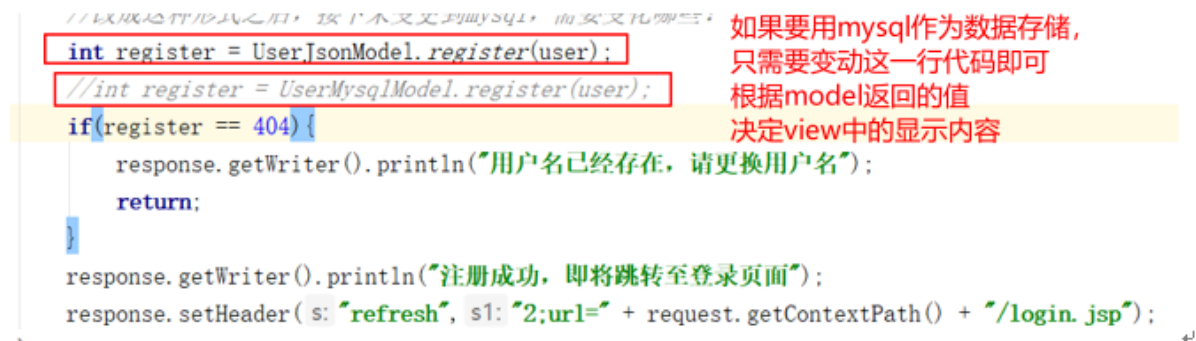
MVC：三个模块

Model：bean，对于bean的操作，与数据库相关的：用户注册，登录，查询，新增，修改等

View：视图，jsp, response的输出

Controller：控制器，servlet。逻辑，参数校验，调用model代码，根据结果再返回给view

相当于用controller把view和model隔开了



仍然存在的问题：现在是在注册模块中，如果在登录、查询、新增这些模块中，需求发生变更，这些地方也要全部变更 ——> 如何让代码变更的地方尽可能的少

- 方法签名相同
- 返回值相同
- UserJsonModel -> UserMysqlModel (希望不变化)

就是想做一个规范 -> 接口天生就是用来做规范的

所以可以将json和mysql这两个model的实现抽成一个接口，然后它们分别有自己的实现

新名词：UserJsonModel -> UserDao (Data Access Object) 专门处理数据的

三层架构

还是出于解耦的需求

接口 + 多态 + 组合 (唯一的耦合性存在于组合中，通过Spring解决)

- 展示层: view, controller
 - controller: 负责校验和逻辑判断 (看有没有进入service的资格), 调用service, 将返回的结果发送给view (实现的是共性)
 - doGet doPost中只负责分发到具体的方法
- 业务层: service
 - 和某个功能具体相关的代码应当写在这 (实现的是个性)
 - 如分页查询: dao查询得到的结果, 如果和前端需要的相差很大, 需要转换和适配数据结构
- 数据层: dao (一次查询对应dao中的一个方法)
 - 里面就是一个sql, 不要在一个方法中写很多sql, 会导致复用性很差

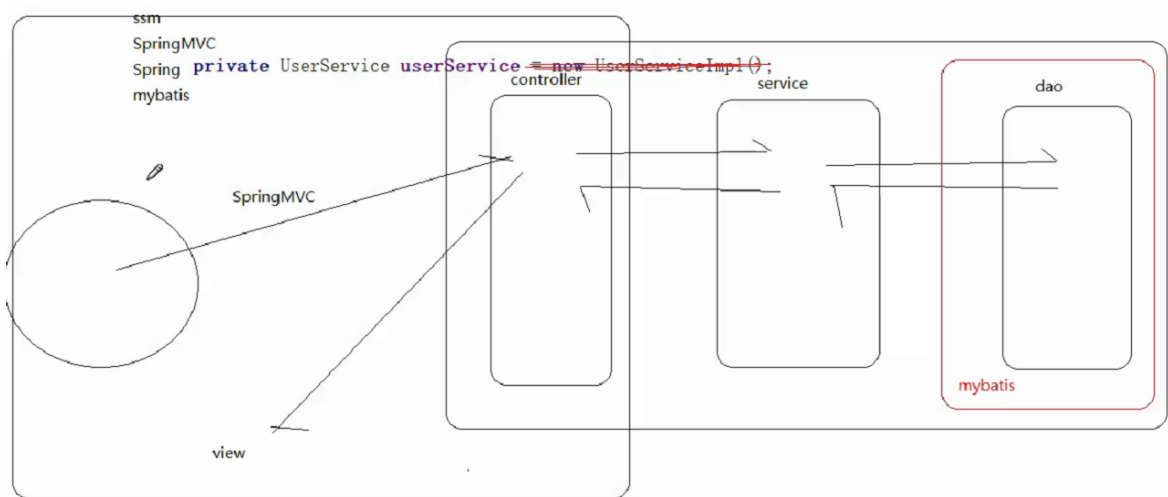
比较复杂的业务: 分页查询, 就需要业务层的参与

节点:	包括子节点 <input checked="" type="checkbox"/> 机房: ===请选择=== 设备名称:	IP:	查询	
节点	机房	设备名称	IP	接入用户
东新11C数据中心	东新3楼机房	东新10C-2924-A	192.168.100.11	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-B	192.168.100.12	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-D	192.168.100.14	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-A	192.168.100.11	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-B	192.168.100.12	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-D	192.168.100.14	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-A	192.168.100.11	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-B	192.168.100.12	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-D	192.168.100.14	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-A	192.168.100.11	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-B	192.168.100.12	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-D	192.168.100.14	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-EU1000-B	192.168.100.11	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-B	192.168.100.12	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2940L3-1	192.168.100.14	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-29500-B15-A	192.168.100.14	[接入用户]
东新11C数据中心	东新3楼机房	东新10C-2924-A	192.168.100.11	[接入用户]
共找到 92 记录, 每页 15 条记录 上一页 1 2 3 4 5 6 7 下一页				

分页查询返回的结果:

- 一共有多少条记录: count(*)
- 一共多少页: 计算
- 当前页应该显示的具体条目

```
select * from table limit pagesize offset (currentPage - 1) * pagesize
```



day 2

Maven

apache.org: java的开源组织

核心功能: 项目构建 + 依赖管理

项目构建：

从编写代码开始，然后到项目正式部署上线这一整套流程：

- 编译：编写的java文件编译成class文件
- 测试：测试写的代码功能上有没有问题（编写一系列的测试用例，但是需要自己一个个去运行）
- 打包：EE项目，通常需要打war包
- 部署：直接部署、虚拟映射

maven：输入一个指令，就完成打包部署以及之前的全部流程

依赖管理：

管理项目依赖的jar包。java项目的一个痛点，导入很多的jar包

减轻项目大小，编译时可以省去导入jar包的过程（在pom.xml的dependencies中添加节点即可）

避免冲突（jar包版本之间的冲突）

-classpath作用：定位到jar包的地址，将jar包加载到内存中

classpath：JVM在类加载时用于定位class文件而使用的一个环境变量

Maven安装

压缩包直接解压安装到盘符根目录，目录不要太深，不要安装在中文目录下

设置环境变量即可 `MAVEN_HOME %MAVEN_HOME%\bin`

用java写的，所以它自身的运行也需要依赖一些jar包

设置本地仓库：

本地仓库是用来存放jar包的，但是下载下来的maven并没有本地仓库

当需要引入一个依赖的时候，需要在maven中做一个配置，让maven先去本地仓库查看有没有对应的jar包

```
<!-- localRepository
| The path to the local repository maven will use to store artifacts.
|
| Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
```

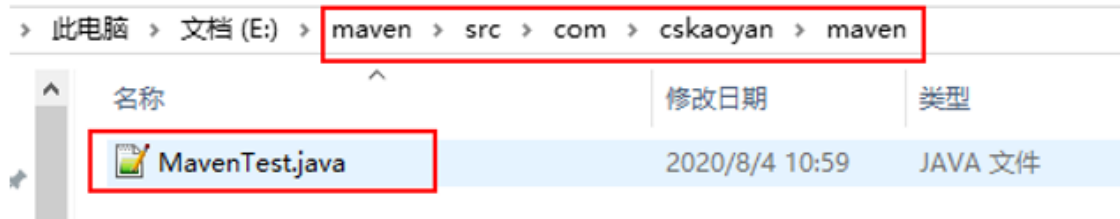
没有的话，会直接到中央仓库去下载对应的jar包（如果没有设置镜像的话），同时缓存在本地仓库，下次就可以直接使用了

设置国内镜像：

```
<mirror>
  <id>nexus-aliyun</id>
  <mirrorOf>central</mirrorOf>
  <name>Nexus aliyun</name>
<url>http://maven.aliyun.com/nexus/content/groups/public</url>
</mirror>
```

项目构建

新建一个maven项目，在maven目录下唤出cmd



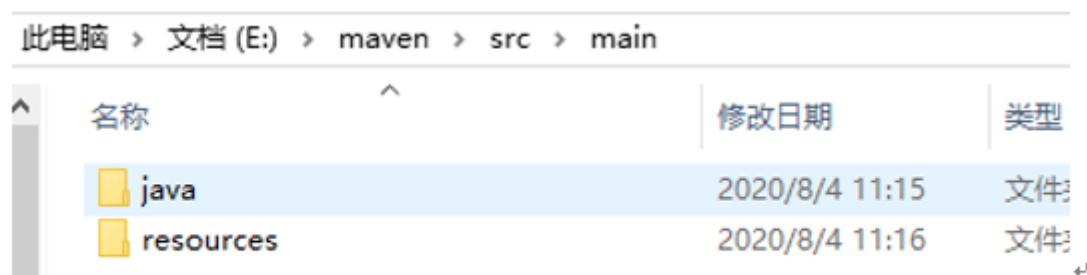
编译

mvn compile 指令，不行，缺少了 pom.xml 文件

需要先配置pom.xml，配置之后编译成功但是没有对应的class文件生成

约定大于配置：没有主动的规定，但是大家一般会遵守（如bean类里面的setter就要写成setName）

- java文件必须要放在 src/main/java 目录下
- 配置文件必须要放在 src/main/resources 目录下
druid.properties在maven项目中放在resources下
- 测试文件必须要放在 src/test/java 目录下
- 测试所需的配置文件放在 src/test/resources 目录下



接下来设置约定的目录之后，再次编译↵

编译成功后会生成target文件夹

测试

mvn test

只需编写好测试用例即可，maven会自动帮你执行测试用例

mvn clean：清理编译产物

测试类名称要写成：xxxTest（约定）

打包

```
mvn package
```

将项目打成jar包或者war包

依赖管理

坐标

- groupId：组织名称
- artifactId：项目名称
- version：版本号

直接去官网查即可：<https://mvnrepository.com/>

scope依赖范围

provided必须要写，否则可能会因为版本冲突问题报错

依赖范围 (Scope)	对于主代码 classpath有效	对于测试代码 classpath有效	被打包，对于 运行时 classpath有效	例子
compile	Y	Y	Y	log4j
test	-	Y	-	junit
provided	Y	Y	-	servlet-api
runtime	-	-	Y	JDBC Driver Implementation

compile: 默认**编译依赖范围**。对于编译，测试，运行三种 classpath 都有效

test: 测试依赖范围。只对于测试 classpath 有效

provided: 已提供依赖范围。对于编译，测试的 classpath 都有效，但对于运行无效。因为由容器已经提供，例如 servlet-api

runtime:运行时提供。例如:jdbc 驱动



Administrator
因为tomcat里面就有

1.2.2 依赖传递

依赖传递

项目2依赖项目1，项目1依赖fileupload，那么项目2也可以拿到fileupload

但是有的jar包无法传递，如scope为 test, provided 的，

发现依赖传递不过来，自己手动再引入依赖即可

依赖冲突

项目引入：

spring-context-----spring-beans

spring-jdbc-----spring-beans

如果两个jar包依赖的beans版本不同时，就会产生依赖冲突

解决冲突的方式：

- 第一声明者优先

取在 pom.xml 中先定义的依赖的版本

- 路径近者优先

即依赖层级比较浅

- 依赖排除

- 提取常量 (推荐)

```
<properties>
<spring.version>5.1.8.RELEASE</spring.version>
</properties>
```



```
<dependency>
```

```
  <groupId>org.springframework</groupId>
```

```
  <artifactId>spring-context</artifactId>
```

```
  <version>${spring.version}</version>
```

```
</dependency>
```

```
<dependency>
```

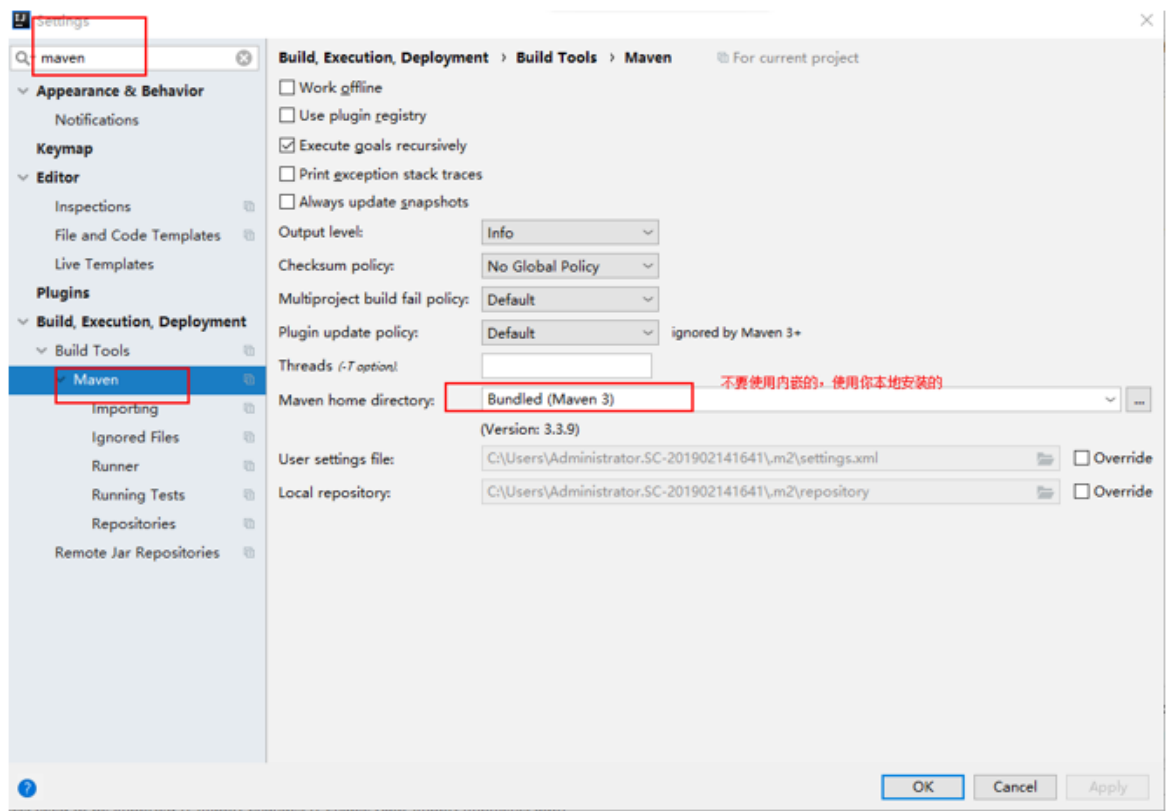
```
  <groupId>org.springframework</groupId>
```

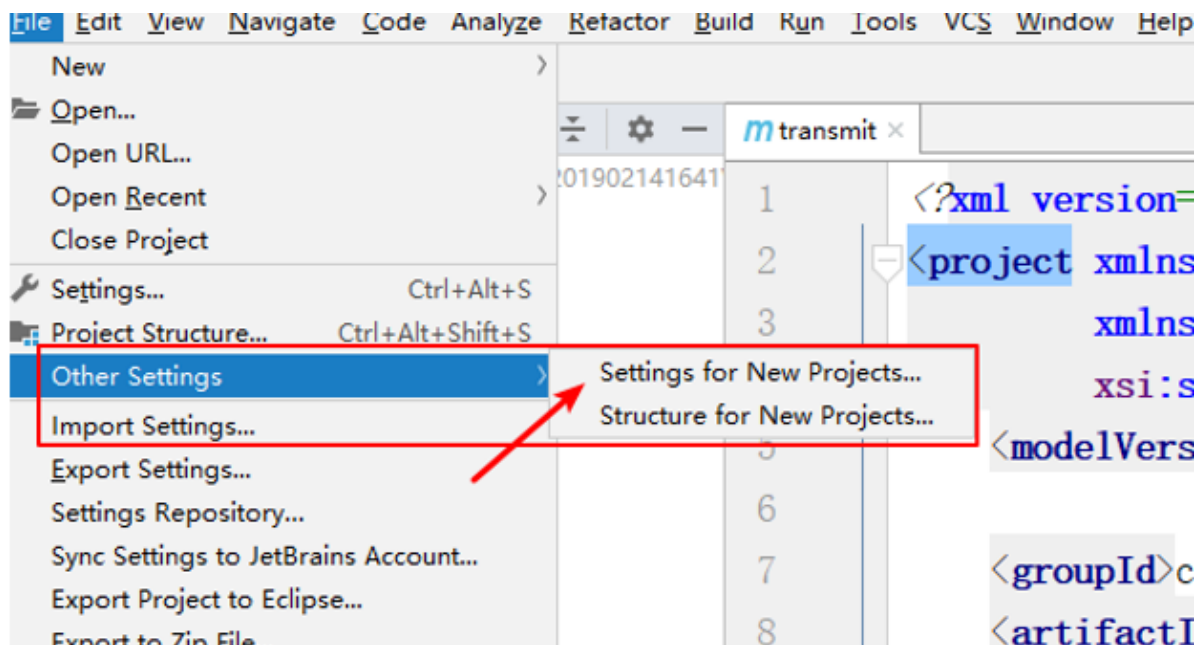
```
  <artifactId>spring-jdbc</artifactId>
```

```
  <version>${spring.version}</version>
```

```
</dependency>
```

idea开发maven项目





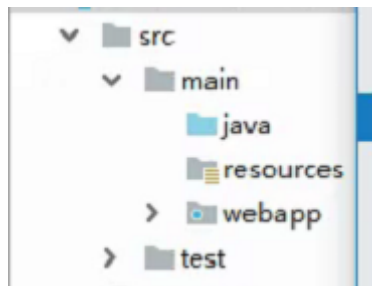
生命周期

指令可以组合使用 mvn clean package

install: 把项目放到本地仓库

如何使得jar包放到部署目录的lib规则:

- 自己制定映射规则: 在module setting的facets中设置web项目的配置文件目录和根目录



- 再在 pom.xml 中设置

```
<packaging>war</packaging>
```

- 然后选择war exploded的artifact (记住不要自己点create artifact)

注意事项: mvn编译时默认用的JDK1.5

所以需要到maven的 conf/setting.xml 中去设置

```
<profile>
  <id>jdk-1.8</id>
  <activation>
    <activeByDefault>true</activeByDefault>
    <jdk>1.8</jdk> </activation>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
  </properties>
</profile>
```

day 3

项目一

管理员账号模块

前后端分离概念

最开始的时候，前端地位比较低（html, css, js）

页面由jsp显示

页面和数据是从一个服务器中获取的

前后端分离后，页面和数据从两个不同的服务器获取

即先获取页面，浏览器再根据页面中的url请求后端服务器的数据

vue项目

先需要做的事

- vue目录下：cnpm install 根据package.json中的dependencies 导入包
- npm run dev：以开发者模式运行前端服务器

api：请求

admin.js：后台管理系统中的所有请求

client.js：前台系统里面的所有请求

设置后端服务器的地址在 conf/axios-admin.js 中

接口文档：

功能上类似于java中的接口

java中的接口：调用某个接口方法，会返回一个结果，但是不知道内部是怎么实现的

项目中的接口：向某个地址发起请求，接下来会返回一个对应的结果：响应报文，前端也不知道它是如何实现

接口文档：里面写的就是一个一个的请求，为了规范和统一，权责分明

根据js文件和官网：确定返回值、数据结构、内容

登录接口

请求失败，fail

因为跨域：自己在一个主机的8085端口，然后又发送请求到这个主机的8084端口，就会产生跨域，浏览器会屏蔽这个行为

在每次请求前都会先发送一个options试探请求，去试探服务器究竟支不支持，如果支持（响应报文中那几个响应头），才会真正发送请求

- 如果获取请求参数，请求体中不是key-value的数据结构
- 如何处理请求体里面的数据：`getInputStream()`
- 获取请求体之后怎么办：`ByteArrayOutputStream -> toString("utf-8")`

gson: jsonStr -> java object (object class需要和json中的数据格式对上)

- 处理
- 响应（在响应体中写入json字符串）

根据code的值，前端的js会读result对象中的不同的属性（看js文件中的处理来确定自己传的json的数据结构）

- 写完一个接口，及时写接口文档

```
// Result就是整个传回前端的json字符串的数据类型
public class Result {
    // code为0读取message，非0处理data
    private Integer code;
    // 报错信息
    private String message;
    // data中是传入的Object， Map<String, Object>或者它要求的数据类型对应的Class
    private Object data;
    // ...
}
```

DEBUG 无法登陆

- 查看请求报文，如果是failed
请求没有成功
- 确保8084的服务器启动成功

- 看响应报文的响应头（跨域有没有处理）
- 看有没有进入servlet

servlet多给了就会有问题 provided 会有版本冲突

一个接口中一般会对三个对象：bo vo bean

- **bo**: business object
用于处理客户端输入的
- **vo**: view object
用于展示输出的
一个视图可能是由多个model组合而成（从多个数据模块获取数据）
里面的数据可能是多个bean杂糅而成，向客户端输出
- **bean**
和数据库对应的，一般和数据库表字段一一对应

要实现的功能在注释写TODO

day 4

简单多条件查询

动态sql：根据传入的条件来动态拼接形成SQL语句

```
String baseSql = "select * from admin where 1 = 1 ";
// 1 = 1 就是 true 在解析的时候就会去掉
// query第三个参数是Object... 实际上就是Object[]
// 所以最好传Object[]进去，要不然会出各种问题
// 比如传入List<Object>里面有几个对象，但是Object...只会认为这是一个参数，就会导致参数
// 个数不匹配
List<Object> params = new ArrayList<>();
// 参数非空就连接到SQL语句，且增加参数到params中
if(!StringUtils.isEmpty(searchAdminBO.getEmail())){
    baseSql = baseSql + " and email like ? ";
    params.add("%" + searchAdminBO.getEmail() + "%");
}
if(!StringUtils.isEmpty(searchAdminBO.getNickname())){
    baseSql = baseSql + " and nickname like ? ";
    params.add("%" + searchAdminBO.getNickname() + "%");
}
// sql里面的问号一定要和参数的个数保持一致
// 否则sql执行会报wrong number of arguments 参数个数不匹配
List<Admin> admins = null;
try {
    admins = runner.query(baseSql, new BeanListHandler<Admin>(Admin.class),
        params.toArray());
}
```

```
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
    return admins;
```

官网

115.29.141.32:8085

day 5

商品管理模块

商品表中的

一个类目对应多个商品，一个商品对应多个规格

- price:取所有规格中的最小值
- stockNum:取所有规格中的和

在插入和修改商品的时候，再根据规格表更新商品表中的这两个值即可

这样查询的时候就不用计算，只是插入和修改的时候需要计算（查询次数多于插入和修改）

图片上传

表单中只有二进制文件，返回一个全路径的url即可

因为开发、测试和生产环境用的域名不一样：

所以路径不能写死，需要从配置文件中读取出域名+ 端口号拼接出全路径（使全路径灵活起来）

在Listener中读入properties，然后放入context域即可

删除商品：删除商品及其规格

删除类目：删除类目，及其商品，及其商品的所有规格

TODO: 事务：保证都成功或者都失败（保证插入商品和其规格是一个原子性过程）

在mysql中获取**当前线程**中最后一次插入的id的值：

```
select last_insert_id(); # 返回值在Java中的对应的数据类型为BigInteger  
select count(*) from users; # 返回值在Java中对应的数据类型为Long  
# 都是用 new ScalarHandler<>()处理
```

day 6

订单管理模块

MySQL与Java数据类型的对应：

基本上都是对应Java中的包装类，在model中成员变量用包装类

MySQL	Java
decimal	BigDecimal
date	java.sql.Date
bit	Boolean

因为js对象的属性部分就是一个Map<String, Object>，

所以可以用Map<String, Object>对应传回的json字符串中的data

count(*) 与 count(id)的区别

- `count(*)`：统计所有的记录数，包括为null的记录
- `count(column name)`：统计该列非null的记录
- `count(1)`：跟 `count(*)` 一样

反射的利用：

BeanHandler会根据传入的class对象，通过反射创建对象，

- 先通过一个无参的构造函数 `clazz.newInstance()`，创建该对象
- 然后再通过 `setColumn`，即数据库表中各个字段的方法尝试给该对象的成员变量赋值，所以可以利用这一点，让BeanHandler帮我们完成一些数据的封装工作
- 如：

```
// stateId是数据库表中的字段
// 所以setStateId方法一定会被尝试调用，且会传值进来（如果没有也不会报错，只是赋值不成功）
// 所以可以根据传入的值来写控制逻辑设置state的值
public void setStateId(Integer stateId) {
    this.stateId = stateId;
    if (stateId == -1) {
        setState("全部");
    } else if (stateId == 0) {
        setState("未付款");
    } else if (stateId == 1) {
        setState("未发货");
    } else if (stateId == 2) {
        setState("已发货");
    } else if (stateId == 3) {
        setState("已到货");
    }
}
```

- 又如：

```
// 数据库表中有这几个字段，所以这几个set方法一定会被调用
// 在其中给user设置初值，就让BeanHandler帮我们封装好了user对象
private UserVO user = new UserVO();
public void setNickname(String nickname) {
    user.setNickname(nickname);
}

public void setName(String name) {
    user.setName(name);
}

public void setAddress(String address) {
    user.setAddress(address);
}

public void setPhone(String phone) {
    user.setPhone(phone);
}
```

在model类中要写清楚 该model的作用

Week 15

day 1

权限验证

后台管理系统除了登录和注销的接口，其他的接口都需要拦截：用白名单

前台用户系统只需拦截用户个人中心、订单和购买等行为：用黑名单

拦截过程：

- 判断该请求方法
- 判断接口是否需要拦截
- 再根据session域中是否有相关信息判断是否被拦截
(登录时在session域中放入一个浏览器对应的信息)

· session的id一直在变化：

因为cookie请求头中没有有效的JSESSIONID，

导致服务端的 `getSession()` 一直在生成不同的session对象

没有的原因是因为跨域

解决不携带cookie请求头的方法

- 前端代码：
 axios.defaults.withCredentials = true
 两个文件 axios-admin axios-client
- 服务端代码：
 access-control-allow-origin 填写上前端对应的域名

设置之后，get 和 post 请求每次都会携带SESSIONID，但是options请求仍然不会携带（不影响，因为它不进入我们的处理逻辑）

filter 和controller需要两份
 service和dao, model都可以复用

day 2

正则表达式

概念

正确的规则表达式，regular expression, regex

作用

用来判断、检索、替换那些符合某个模式的文本

正则基本规则

次数限定符

作用于前面单个字符或者是用括号括起来的表达式

字符	描述
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于 {0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的 "do"。? 等价于 {0,1}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,}	n 是一个非负整数。至少匹配 n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "fooooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{n,m}	m 和 n 均为非负整数，其中 n <= m。最少匹配 n 次且最多匹配 m 次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。

字符集

只匹配单个字符

<code>x y</code>	匹配 <i>x</i> 或 <i>y</i> 。例如， <code>'z food'</code> 匹配 “z” 或 “food”。 <code>'(z f)ood'</code> 匹配 “zood” 或 “food”。
<code>[xyz]</code>	字符集。匹配包含的任一字符。例如， <code>"[abc]"</code> 匹配 “plain” 中的 “a”。
<code>[^xyz]</code>	反向字符集。匹配未包含的任何字符。例如， <code>"[^abc]"</code> 匹配 “plain” 中的 “p”。
<code>[a-z]</code>	字符范围。匹配指定范围内的任何字符。例如， <code>"[a-z]"</code> 匹配 “a” 到 “z” 范围内的任何小写字母。
<code>[^a-z]</code>	反向范围字符。匹配不在指定的范围内的任何字符。例如， <code>"[^a-z]"</code> 匹配任何不在 “a” 到 “z” 范围内的任何字符。

特殊字符

<code>^</code>	匹配输入字符串开始的位置。如果设置了 <code>RegExp</code> 对象的 <code>Multiline</code> 属性， <code>^</code> 还会与 <code>\n</code> 或 <code>\r</code> 之后的位置匹配。
<code>\$</code>	匹配输入字符串结尾的位置。如果设置了 <code>RegExp</code> 对象的 <code>Multiline</code> 属性， <code>\$</code> 还会与 <code>\n</code> 或 <code>\r</code> 之前的位置匹配。
<code>\d</code>	数字字符匹配。等效于 <code>[0-9]</code> 。
<code>\D</code>	非数字字符匹配。等效于 <code>[^0-9]</code> 。
<code>\w</code>	匹配任何字类字符，包括下划线。与 <code>"[A-Za-z0-9_]"</code> 等效。
<code>\W</code>	与任何非单词字符匹配。与 <code>"[^A-Za-z0-9_]"</code> 等效。

其他

<code>\</code>	将下一个字符标记为一个特殊字符、或一个原义字符、或一个 向后引用、或一个八进制转义符。例如， <code>'n'</code> 匹配字符 “n”。 <code>'\n'</code> 匹配一个换行符。序列 <code>'\\'</code> 匹配 “\” 而 <code>'\('</code> 则匹配 “(”。
<code>.</code>	匹配除换行符 <code>\n</code> 之外的任何单字符。要匹配 <code>.</code> ，请使用 <code>\.</code> 。

day 3

事务

一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生（转账）

ACID：

- 原子性
- 一致性：事务中的操作必须满足一定的约束（字段、外键的约束）
- 隔离性
- 持久性

事务的前提，必须是在相同的连接中

```
connection.setAutoCommit(false);
// ...
connection.commit(); // connection.rollback()
```

queryrunner.update()源码

```
// 如果调用该方法时没有指定connection, 则每次调用update都会从传入的数据库连接池中获取
public int update(String sql, Object param) throws SQLException {
    Connection conn = this.prepareConnection();
    return this.update(conn, true, sql, param);
}
```

问题: 如何保证service中的里面所有调用的dao方法全部用一个connection

- 方法一: 再写一套API, 调用dao方法时传入connection
- 方法二: 利用线程对应的thread对象中的map来存取该线程唯一对应的一个connection
因为一个http请求对应的controller, service, dao都是在同一个线程中

利用threadLocal在当前线程中的map中设置 threadLocal -> connection

ThreadLocal set/get 源码

```
public void set(T value) {
    // 获取当前线程对象
    Thread t = Thread.currentThread();
    // 获取当前线程对象中的map
    ThreadLocalMap map = getMap(t);
    // map不为空, 则将一个entry: threadLocal -> value放入map中
    if (map != null)
        map.set(this, value);
    else
        // map为空则先创建map。再放入键值对
        createMap(t, value);
}
```

```
public T get() {
    // 获取当前线程对象
    Thread t = Thread.currentThread();
    // 获取当前线程对象中的map
    ThreadLocalMap map = getMap(t);
    // 如果map不为空
    if (map != null) {
        // 从map中取出当前threadLocal对应的entry
        ThreadLocalMap.Entry e = map.getEntry(this);
        // 返回其value
        if (e != null) {
            @SuppressWarnings("unchecked")
            T result = (T)e.value;
            return result;
        }
    }
    // 如果map为空, 则创建map, 然后返回null
    return setInitialValue();
}
```

问题：如何保证service与dao中能拿到同一个threadLocal对象

工具类，设置静态变量，保证threadlocal的唯一性

```
public class DruidUtils {

    private static DataSource dataSource;
    // 保证service层和dao层用的是同一个threadLocal，因为thread中的map是threadlocal ->
    object
    private static ThreadLocal<Connection> threadLocal = new ThreadLocal<>();

    static {
        String path = FileUtils.getPath("druid.properties");
        FileInputStream in = null;
        try {
            in = new FileInputStream(path);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        Properties properties = new Properties();
        try {
            properties.load(in);
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            dataSource = DruidDataSourceFactory.createDataSource(properties);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static DataSource getDataSource() {
        return dataSource;
    }

    /**
     *
     * @param transactional true表示开启事务
     * @return 同一个事务中所用的connection
     * @throws SQLException
     */
    public static Connection getConnection(boolean transactional) throws
    SQLException {
        // 如果是事务性的连接
        if (transactional) {
            // 从当前线程中获取该threadLocal唯一对应的connection
            Connection connection = threadLocal.get();
            // 如果是第一次尝试获取
            if (connection == null) {
                // 从数据库连接池中随便获取一个connection
                connection = getConnection();
                // 将entry:threadlocal -> connection 放到当前线程的map中去
                threadLocal.set(connection);
            }
            // 返回当前线程中thread的map中唯一的connection
            return connection;
        }
    }
}
```

```
    }  
    return getConnection();  
}  
  
/**  
 * 因为整个web项目用的是线程池，当前线程可能会在将来被唤醒使用  
 * 那么thread里面的map的中的数据也将失去线程隔离性  
 * 所以在用完thread中的map后，将threadLocal-> null  
 */  
public static void releaseConnection() {  
    threadLocal.set(null);  
}  
  
public static Connection getConnection() throws SQLException {  
    return dataSource.getConnection();  
}  
}
```