第一天 项目设计

- 数据库设计 - **PowerDesigner**

业务管理　　系统管理

核准件		
组织机构代码	VARCHAR2(32)	
备注	VARCHAR2(128)	
联系人	VARCHAR2(32)	
联系电话	VARCHAR2(32)	
验资文件	VARCHAR2(32)	
文件备注	VARCHAR2(128)	
核准件编号	VARCHAR2(32)	<pk>
经办人	VARCHAR2(32)	
反馈状态	CHAR(1)	

企业		
组织机构代码	VARCHAR2(32)	<pk>
外汇登记证号	VARCHAR2(32)	
企业中文名称	VARCHAR2(64)	
企业英文名称	VARCHAR2(64)	
联系人	VARCHAR2(32)	
联系电话	VARCHAR2(32)	
注册资本	NUMBER(9)	
外方注册资本	NUMBER(9)	
注册币种	CHAR(3)	
经办人	VARCHAR2(32)	
登记日期	CHAR(10)	

投资人		
投资人登记编号	VARCHAR2(32)	<pk>
投资人名称	VARCHAR2(32)	
国别	CHAR(3)	
组织机构代码	VARCHAR2(32)	
联系人	VARCHAR2(32)	
联系电话	VARCHAR2(32)	
电子信箱	VARCHAR2(32)	
备注	VARCHAR2(128)	
经办人	VARCHAR2(32)	
登记日期	CHAR(10)	

用户		
用户代码	VARCHAR2(32)	<pk>
用户姓名	VARCHAR2(32)	
机构类型	CHAR(1)	
用户密码	VARCHAR2(32)	
登记日期	CHAR(19)	

企业投资人		
组织机构代码	VARCHAR2(32)	<pk>
投资人登记编号	VARCHAR2(32)	<pk>
注册资本出资额	NUMBER(9)	
利润分配比例	NUMBER(3)	

  - 数据库的表一定要有主键
    1. 表中数据的唯一性标示，通过可以快速访问数据
    2. 多张表进行数据关联时，可以采用主键和外键关联的方式
  - 在实际工作中，日期类型一般很少使用 date 类型，因为不同的数据库处理方式不一样，所以一般采用 char 类型声明。
  - 表关系中如果是一对多的关系（Emp，dept）,采用外键关联就可以了，但是如果是双向一对多（企业,投资人），一般采用第三张表实现。
  - 表设计不采用外键的原因：效率

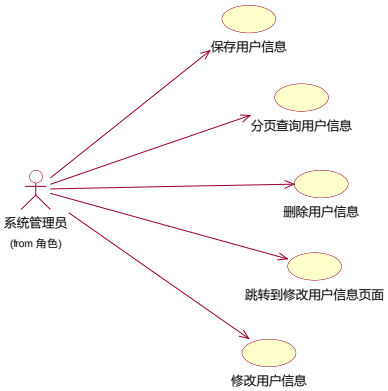
    外键的作用是数据的完整性约束.,但是这种校验，在数据量大的情况下，会导致性能降低，所以在一些对性能要求比较高的系统中，一般不使用外键。约束操作一般采用程序或页面业务功能来实现。

- 业务流程设计 – **Rational Rose**

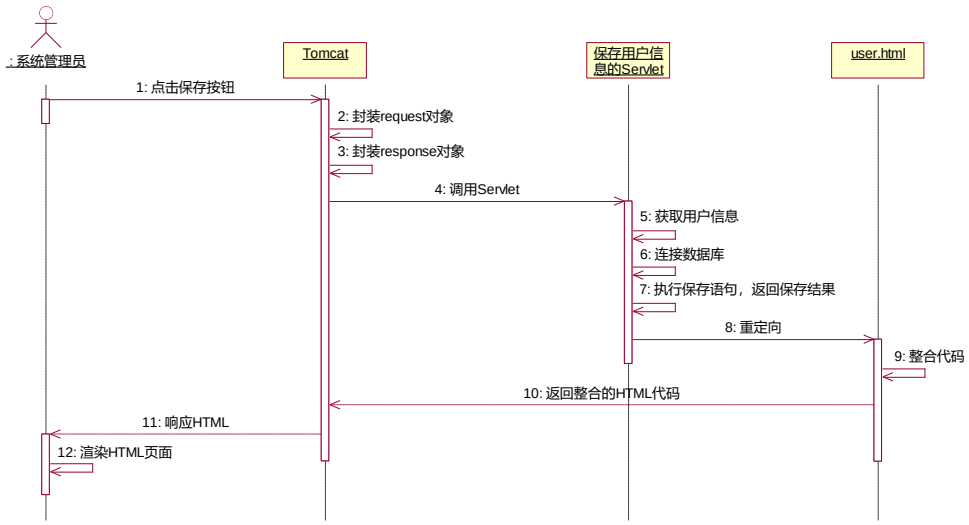
  UML （统一建模语言）

  运用统一的标准化的图形或标记来表示系统当中的业务，并且可以对软件系统进行面向对象的建模操作。

  - 使用案例图、用例图（**以用户的角度描述系统的功能**）

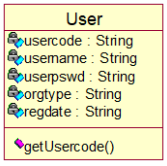


  - 序列图，时序图（**业务流程图，描述程序的执行流程**）



1. 小人：表示角色，其实也是业务流程的发起者
2. 黄色的长方形：表示在业务流程中参与的对象
3. 实线：表示信息或数据的传递。在代码实现上其实就是方法的调用。
4. 折线：表示对象自己调用自己
5. 虚线：表示信息或数据的返回。在代码实现上表示方法结束，返回。
6. 长条形：表示在整个业务流程中，对象的生存时间。

- 类图（**类的图形化表现方式**）



- 搭建项目的开发环境
  - 搭建的开发环境应该可以模拟用户的真实访问环境。可以避免一些不必要出现的问题。
  - 搭建项目开发环境的步骤
    1. 创建一个 Web 项目，取名为 PrjEGov
    2. 将原型界面中所有资源拷贝到项目的 **WebRoot** 文件夹中
    3. 开发工具会对静态资源进行语法校验，但是这个校验和我们项目开发关系不大，所以可以屏蔽。点击菜单 window ➔ Pref…➔ MyEclipse ➔ Validation 去掉所有的勾，保留 JSP 的校验。
    4. 修改 web.xml 文件，将 **login.html** 文件设置为欢迎页面
    5. 为了让用户访问时，不需要指定端口号，可以修改配置文件（**TOMCAT_HOME/conf/server.xml**），将端口号由 8080 改为 80
    6. 将 Web 项目发布到服务器的 webapps/**ROOT** 文件夹中，那么这样，用户访问时不需要指定项目的名称，直接访问服务器的根就可以访问我们的项目了。
    7. 将 IP 地址使用域名进行代替（修改 **C:\WINDOWS\system32\drivers\etc\hosts**），那么这样，用户可以直接通过域名访问项目。添加内容 ： 127.0.0.1 www.egov.com

第二天 用户管理

- HTML 中使用图片来代替按钮
  - 设定图片的样式， 模拟按钮的点击效果
  - click 称之为事件，当点击图片的**时候**执行 JS 脚本代码。（onclick 是事件句柄，注册在事件句柄后边的代码在事件发生的时候自动执行）
  - document.location = “URL”；修改当前浏览器地址栏的 URL。
- 验证表单数据
  - 获取页面元素的常用方式
    1. document.getElement**ById**(id) – 使用元素的 ID 属性访问，并且只会获取一个;
    2. document.getElements**ByName**(name) – 使用元素的 NAME 属性访问，可以获取同名的多条数据;
    3. document.getElements**ByTagName**(tagName) – 使用标签的名称访问，可以获取相同名称的多条数据;
  - JS 中是弱类型语言，没有类型的概念，所以任何的数据类型之间都可以互相转换。
  - 判断页面元素的取值是否为空，主要是判断表单元素对象的 value 属性是否为空（空字符串不是 null，空字符串是””）。
- 提交表单
  - submit 按钮
  - JS 方法获取表单对象，调用 submit 方法。
- 提交表单的服务器路径
  - **在 JSP、HTML 页面中编写的提交路径不需要加“项目名”了，因为这个项目已经部署到 Tomcat 服务器根路径下了，这种方式只适合开发这个项目中使用。如果大家开发的项目还是部署到 webapps 下，路径还和以前相同。**
    以前：都以 “/” 开始，/项目名/servletPath
    现在：都以“/” 开始，但是 /项目名 省略了。直接编写 /servletPath
    注意：我们现在写的路径没有涉及到相对路径，一直使用的还是绝对路径。
- Servlet 中，获取用户提交的数据
  - request.getParameter(name) – 获取单一的表单数据;
  - request.getParameterValues(name) – 获取相同名称的多条表单数据;
  - request.getParameterMap() – 在不知道名称的情况下，获取所有的表单数据集合;
  - request.getParameterNames() – 获取所有的表单数据传递的名称;
- 时间格式化
  - 格式化字符串的书写方式。
  - MM：月份 ； mm ： 分钟
- 数据库的操作
  - DML（insert, update, delete）： pstat.**executeUpdate()**;
    1. executeUpdate 方法的返回值为 int 类型，表示 sql 语句成功执行后，影响数据库的数据条数
  - DQL（select）: pstat.**executeQuery()**;
    1. executeQuery 方法的返回值为 ResultSet 对象，包含查询结果集合。
- 页面跳转的方式
  - 转发
    1. 可以共享请求对象中保存的数据
    2. 如果路径使用斜杠开头，表示从 web 应用的根开始查找
  - 重定向
    1. 无法共享请求对象中保存的数据。
    2. 如果路径使用斜杠开头，表示从服务器的根开始查找。

- 工具类的封装
  - 将重复性的代码进行封装，在后续的开发中可以直接调用，非常方便，可以提高开发效率
  - 工具类中一般的方法都是静态的，可以通过类名直接访问，方便使用。
  - 维护代码时，只要修改一个地方，所有调用的地方全部都修改了，维护性比较好
  - 日期工具类
  - 数据库工具类
- 数据传递过程中，出现乱码
  - 表单中如果存在中文，在浏览器将数据传递到服务器时，会自动将中文转换成特殊编码（ISO8859-1）的内容，然后传递。因为网络中无法传递中文
  - 将乱码的字符串转换成中文
    - 将乱码字符串按照指定的编码方式恢复成字节码序列
    - 将字节码序列按照正确的编码进行转换
  - 设置请求对象的字符编码可以解决乱码问题。
    - 设置字符编码必须在获取任意数据之前，否则不起作用。
    - 设置请求对象的字符编码只对请求体的内容起作用。只支持 POST 方式提交数据，不支持 GET 方式提交数据，因为 get 方式将数据放置在请求头部中传递给服务器
  - GET 方式传递数据中如果存在中文乱码可以修改服务器配置文件解决
    - TOMCAT_HOME/conf/server.xml

      <Connector port="80" protocol="HTTP/1.1"

      connectionTimeout="20000"

      redirectPort="8443" **URIEncoding="GB18030"** />
    - 服务器的配置根据不同的服务器，配置方式不一样。
- 数据库保存数据时候也可能会出现乱码
  - 安装数据库时，一定要指明编码方式。
  - Oracle 数据库可以根据系统语言自动选择编码。
  - MySQL 数据库默认的编码方式为 ISO8859-1，安装是一定要修改字符编码。
- JSP 显示数据时，也可能出现乱码
  - 通过 page 指令，设置属性来控制编码
- Servlet 输出内容，也可能出现乱码
  - 设定输出内容类型的字符编码 response.setContentType(“text/html;charset=GB18030”);
- HTML 显示数据时，也可能出现乱码
  - 设置 meta 标签，控制浏览器窗口的编码方式

第三天 用户管理

- 分页查询
  - 分页查询分为两种方式
    - 逻辑分页
      - 从数据库中获取所有的数据，放置在内存当中，然后通过逻辑运算将数据进行筛选，然后获取当前页面中需要显示的数据。
      - 优点：不需要频繁地访问数据库，只需要从内存筛选数据就可以了。
      - 缺点：如果数据量大的场合，会占用大量的服务器内存空间，导致服务器性能降低。
      - 逻辑分页可能查询的结果和数据库的数据不一致，所以**不推荐使用**这种分页方式。

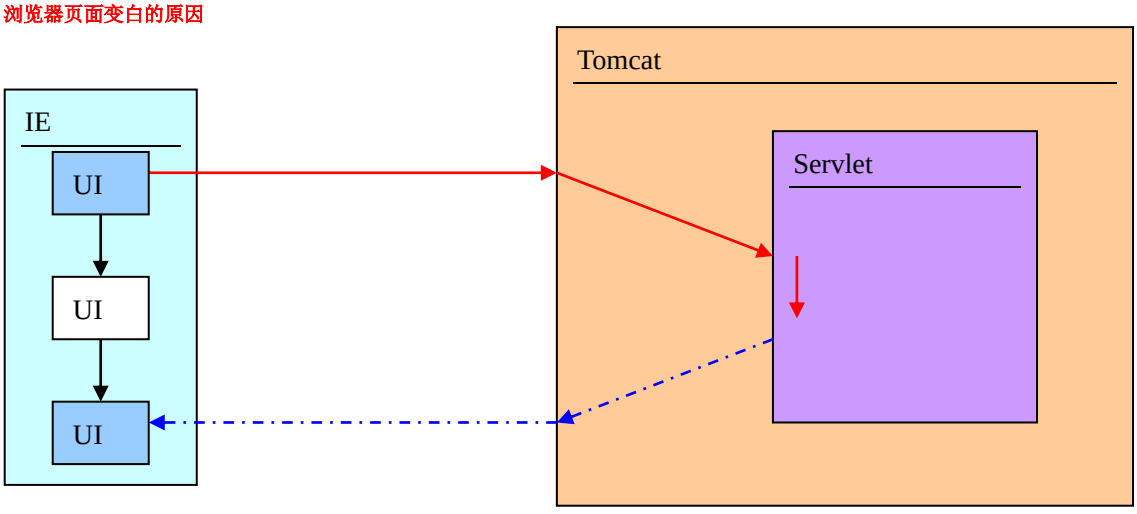
- 物理分页
  - 由数据库本身的分页查询机制进行查询操作，获取的数据不会很多，不会占用大量的内存。并且采用数据库内部查询机制，查询效率非常高。**推荐使用**
  - 优点：从数据库获取的数据量不是很多，占用内存的空间不会很多，那么这样可以提升服务器的性能。
  - 缺点 1：进行翻页时，需要重新访问数据库，而数据库的访问会导致会使用大量重量级对象，导致性能降低。
  - 缺点 2：因为 SQL 文采用数据库特有的操作，所以会导致程序不具备移植性。

- Oracle 中的分页查询
  - 采用 rownum 的三层查询嵌套完成
    - Rownum 不能和 order by 语句联合使用

Oracle 本身不支持 rownum 关键字的大于或大于等于的操作。

第四天 用户管理

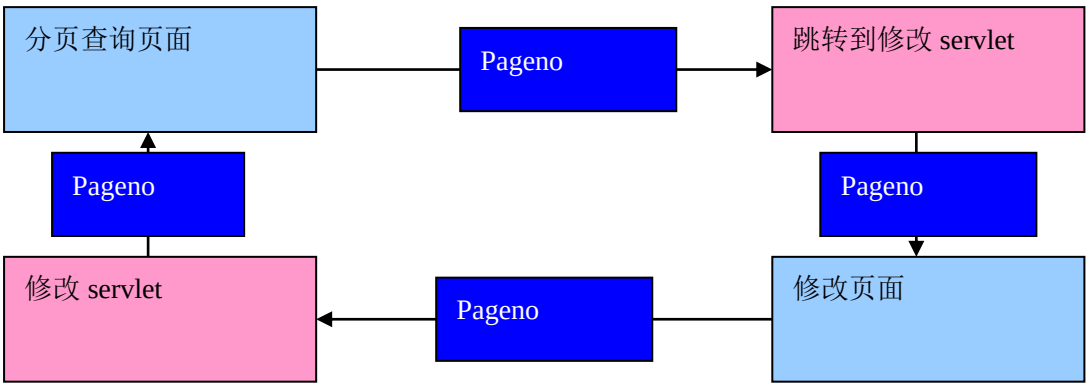
- 表单元素的 value 属性
  - Value 属性直接在标签中声明，表示设置表单元素的初始值。
  - Value 属性只对 input 标签起作用。其他标签的表单元素<select><textarea>不起作用。
- 浏览器解析静态代码的方式
  - 采用逐行解析的方式解析静态代码
- Body 标签的 onload 属性
  - 当页面内容全部加载完成后执行 JS 脚本代码
- **浏览器页面变白的原因**



IE 浏览器显示页面的内容是使用 **UI(User Interface)线程**进行显示的，当页面点击超级链接或进行表单数据提交时，UI 线程会将页面的内容清空，所以页面才会变白，然后如果服务器发送响应内容，浏览器 UI 线程解析内容后，会重新显示页面。所以浏览器的右下角会存在一个进度条，用于显示加载页面的过程，将这个过程我们称之为页面刷新。

- 浏览器变成白页面的原因：
  - 服务器没有发送任何的响应内容
  - 服务器发生了错误，影响了正常的响应逻辑，导致没有响应内容。

- 在指定的页面修改数据后回到指定的页面

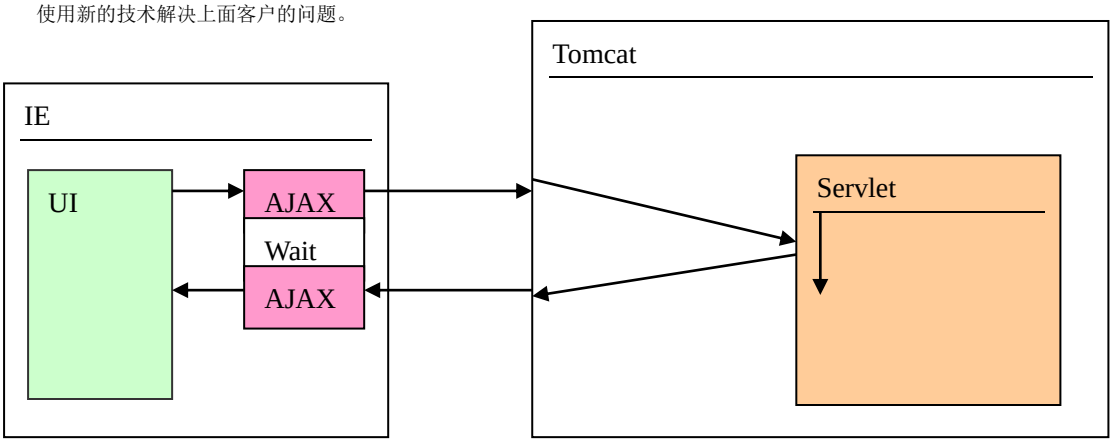


在流程当中的所有环节里，页码参数不能丢失，如果丢失，无法实现用户的需求

- confirm 方法和 alert 方法
  - 这两个方法都是 JS 当中 window 对象的内置方法。Window 对象可以省略，所以这两个方法访问时可以直接使用。
  - 这两个方法都可以暂停 UI 线程的解析操作。
- 数据库操作的事务处理
  - 将多个数据库的操作当成一个整体，要么完全成功，要么完全失败。
  - 事务的使用流程
    - 开启事务
    - 提交事务
    - 回滚事务
    - 结束事务
    -

第五天 系统扩展&AJAX

- 将分页功能使用分页对象进行封装。
- 为了避免主键重复的问题，在保存之前应该对主键进行校验。
- 客户提出新的要求
  - 验证用户代码时，不应该将其他表单元素的值清空
  - 验证用户代码时，页面不应该刷新。
- 使用新的技术解决上面客户的问题。



- AJAX 的实现原理
  - 通过 JS 在浏览器中创建一个新的线程
  - UI 线程不再将数据提交给服务器，而是将数据通过浏览器的内部操作传给 **AJAX 线程**。
  - AJAX 线程获取数据后将数据提交给服务器。
  - AJAX 线程提交数据后，需要等待服务器的响应。在这个过程中，不做任何操作。
  - AJAX 线程获取服务器的响应之后，会通过浏览器内部操作将结果反馈给 UI 线程
  - UI 线程就可以更新页面的内容，实现用户的要求。
  - 在整个过程中，UI 线程没有提交数据，那么页面不会刷新，并且页面元素的值不会被清空。
- **AJAX**
  - JS 在浏览器中创建的一个新的线程
  - 可以采用异步地方式和服务器进行数据交互
  - 新的数据提交方式
- **Ajax 操作的步骤（对应 AJAX 的 5 个状态）**
  1. 创建 Ajax 对象
     - 需要考虑浏览器软件的不同
  2. Ajax 对象建立服务器的连接(open)
     - 提交数据的方式
     - 服务器的连接地址
     - 线程是否异步操作
  3. Ajax 对象发送数据（send）
     - 请求体的内容可能为 null
  4. Ajax 对象**正在**获取数据
     - 预先定义方法用于获取服务器的响应
  5. Ajax 对象**获取完**数据
     - 预先定义方法用于获取服务器的响应
  6. 更新 UI 界面的内容
- AJAX 操作时的 IE 缓存问题
  - 当 **IE 浏览器**使用 **GET 方式**提交请求时，浏览器会将请求地址和服务器的响应结果放置在缓存当中，如果下一次，用户访问**相同路径**的资源时，不会再访问服务器，而是从缓存中获取响应，那么这样，就会出现缓存问题。但是这种方式可以提高访问效率。
  - 如何解决缓存问题
    1. 让用户每一次访问的路径不相同。在请求地址的后面增加时间戳，因为可能会影响客户使用浏览器的效率，所以谨慎使用。

```
url = url + "&timer=" + new Date().getTime();
```

    2. 换浏览器，不能保证用户肯定可以换浏览器，所以这种不推荐使用
    3. 让用户每一次访问都访问服务器，而不是从缓存中获取内容，设置请求头部信息

```
xmlHttp.setRequestHeader("If-Modified-Since", "0");
```

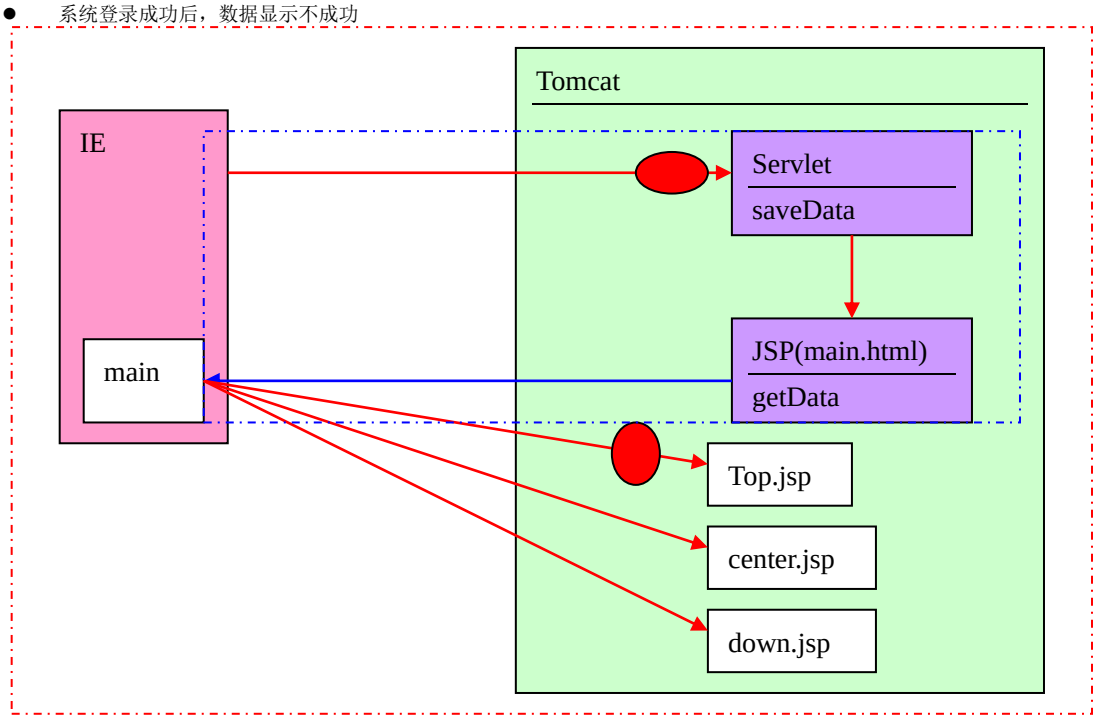
    4. **使用 POST 方式也可以解决缓存问题。**
- AJAX 的应用
  - 表单数据验证
  - 下拉框的联动效果
- 用户登录功能的实现

- Base 标签的作用
  - 用于改变默认相对路径的指向位置，默认情况下，相对路径是以当前资源的**访问路径**为基准，查找资源，但是如果增加 base 标签之后，从 base 标签所指向的位置开始查找资源。

Base 标签是 HTML 标签，也就意味着对 JS 不起作用

第六天 登录&退出&投资人管理

- 系统登录成功后，数据显示不成功



因为在引入 top.jsp 时，等同于发送了新的请求，所以登录 servlet 中保存数据无法共享。

上面图形中的蓝色的虚线框表示请求范围

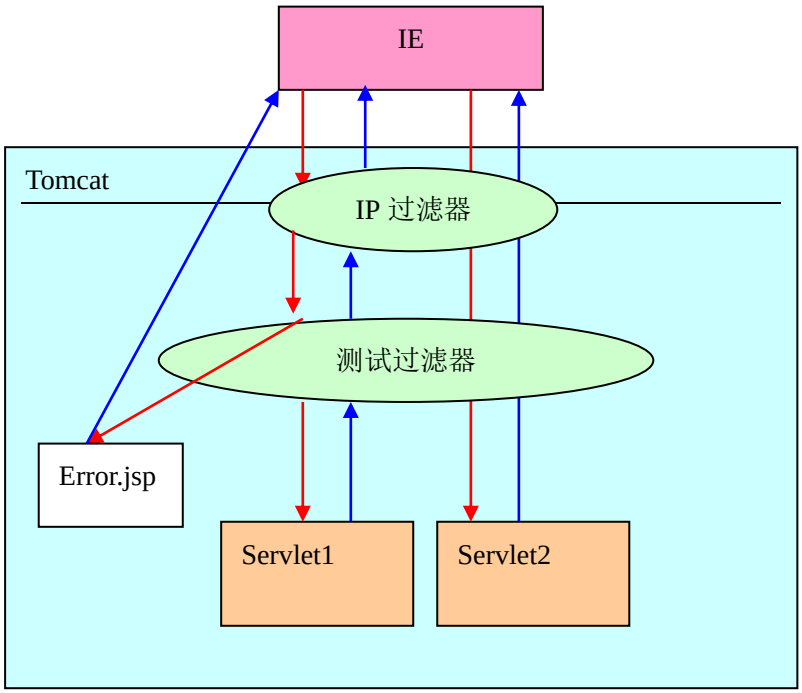
上面图形中的红色的虚线框表示会话范围

- 分页查询中的动态参数查询
  - 参数不是固定不变的，可能存在也可能不存在。这样的参数查询非常难于处理。处理起来不太容易，因为条件之间的关系非常紧密。
  - 如果工作中出现这样的查询，一般采用技术框架解决（Ibatis， Hibernate）
- Oracle 数据库中的序列
  - 可以自动生成序号，不会重复生成，也就不会出现主键重复。
  - 序列是 Oracle 数据库特有的，如果在 SQL 语句中用，那么 SQL 语句不具备移植性
  - 当数据库操作失败时，序列号依然会自动增加，但是数据却没有保存成功，所以下一回保存数据，会发现序号存在跳号的可能。所以序列生成的序号有可能不连续。如果系统开发时，要求序号必须连续的场合，不能使用序列.
- JDBC 操作异常：无效的列索引
  - 当 JDBC 操作时，参数传递的索引值和？的位置不一致会导致这个错误。
- Oracle 操作异常：未明确定义列
  - 多个表进行关联时，存在相同字段名称的列，而对这个列操作时没有增加别名，会导致这个错误。

第七天 系统扩展&反射&Filter

- **将请求中的参数封装为指定对象**
  - 使用反射将逻辑上重复的代码进行封装，可以简化代码开发，提高开发效率。
  - 反射的使用，可以使程序具备更好的通用性，所以基本上所有的技术框架的底层都在使用反射，所以学好反射，对于学习技术框架是非常有帮助的。
- **Filter 过滤器**

1. Filter 技术在 servlet2.3 规范中加入的，大部分的 jsp/servlet 容器都支持



2. **第一个 Filter 程序**
   a) 创建一个 JAVA 类，实现特定的接口（**javax.servlet.Filter**）
   b) 重写 doFilter 方法，完成自定义的过滤逻辑
   c) 修改 web.xml,增加对过滤器的支持

```
<!-- IP 访问过滤器 -->
<filter>
    <!-- 过滤器的引用名称 -->
    <filter-name>ipAccessFilter</filter-name>
    <!-- 过滤器的完整类名 -->
    <filter-class>com.bjpowernode.egov.web.IpAccessFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>ipAccessFilter</filter-name>
```

```
        <url-pattern>/servlet/pageQueryInvestServlet</url-pattern>
    </filter-mapping>
```

3. **Filter 的生命周期**

   a) 过滤器的创建是在服务器启动时完成的

   b) 过滤器创建之后马上进行初始化操作，调用 init 方法

   c) 当用户访问指定路径时，会执行过滤器的 doFilter 方法

   d) 当服务器停止时，会执行销毁方法用于回收资源

4. **Filter 和 servlet 之间的关系**

   a) 两者都是服务器提供的服务资源。只不过提供服务的方式不一样，servlet 主要用于业务逻辑处理，而过滤主要用于对用户的请求进行过滤和筛选，并且过滤器在 servlet 之前执行。

   b) 两者的生命周期非常的像，所以参考的来学习。

   c) 两者的配置方式非常的像，所以可以参考使用

   d) 过滤器和 servlet 都是多线程单实例的，如果多线程同时修改了过滤器中成员属性的话，会出现线程安全问题。

5. **多个 Filter 之间的关系**

   a) 多个过滤器谁先执行，谁后执行？

      1. 多个过滤器执行的顺序取决于**<filter-mapping>**标签的顺序，如果标签声明在前，先执行，但是后执行完毕，声明在后，后执行，但是先执行完毕。

   b) 一个过滤器执行完成后，怎么跳转到另外一个过滤器的。

      1. 通过特定的对象完成请求的传递：**chain.doFilter(request, response);**
      2. **调用方法之后，如果有下一个过滤器，那么会执行过滤器的 doFilter 方法，如果没有过滤器，那么会直接调用最终的资源(Servlet)**

6. **如何给 Filter 传递参数**

   a) 修改 web.xml， 增加初始化参数

```
    <!-- IP 访问过滤器 -->
    <filter>
        <!-- 过滤器的引用名称 -->
        <filter-name>ipAccessFilter</filter-name>
        <!-- 过滤器的完整类名 -->
        <filter-class>com.bjpowernode.egov.web.IpAccessFilter</filter-class>
        <!-- 初始化参数 -->
        <init-param>
            <param-name>ipaddresses</param-name>
            <param-value>127.0.0.1</param-value>
        </init-param>
    </filter>
```

   b) 在初始化方法中获取初始化参数

```
String ipaddresses = filterConfig.getInitParameter("ipaddresses");
```

7. **Filter 的路径匹配模式**

   a) **精确匹配**，不用任何修饰符，如：

```
<filter-mapping>
<filter-name>CharsetEncodingFilter</filter-name>
<url-pattern>/servlet/TestServlet</url-pattern>
</filter-mapping>
```

b) **扩展匹配**，由星号“*”和扩展名组成，如:

```
<filter-mapping>
<filter-name>CharsetEncodingFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

c) **路径前缀匹配**，包含一个目录和一个/*

```
<filter-mapping>
<filter-name>CharsetEncodingFilter</filter-name>
    <url-pattern>/servlet/*</url-pattern>
</filter-mapping>
```

d) **全部匹配**，一般使用/*

```
<filter-mapping>
<filter-name>CharsetEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Servlet 规范中匹配关系的描述：

- 以斜杠开头并且以斜杠星结尾的路径用于**路径前缀匹配**
- 以**星点**开头的字符串用于**扩展名**的映射
- 其他的请求字符串全部都用于**精确匹配**

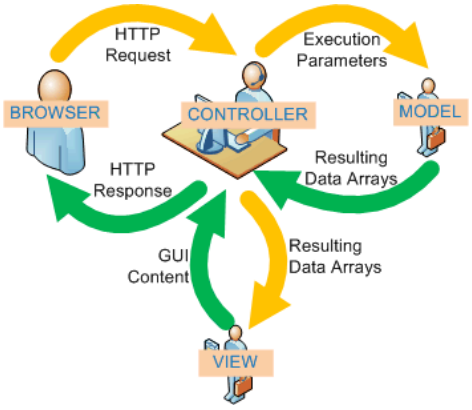
第八天 企业信息登记

- 父子窗口的数据交互
  - 获取父窗口对象 ： var parentWindow = window.opener
  - 获取子窗口对象 ： var childWindow = window.open(…)
- 表格的动态增加和删除
  - <table>→ Table
  - <tr> → TableRow
  - <td> →TableCell
- JS 中字符串的引号嵌套问题。
  - JS 字符串中双引号和单引号可以嵌套使用，双引号和双引号，单引号和单引号不能嵌套使用。
  - “xxx’yyyyy’xx” 或 ‘xxx”yyyyy”xxx’
  - 如果必须在字符串中有相同引号的嵌套，可以采用转义字符.

第九天 系统扩展&MVC&XML

- **MVC 架构模式**
  - MVC 是三个单词的缩写,分别为： 模型(Model),视图(View)和控制（Controller)。

    MVC 模式的目的就是实现 **Web 系统的职能分工**,具体如下图：



  - MVC(Model View Controller)模型－视图－控制器

    MVC 是一个**架构模式**，它强制性的使应用程序的输入、处理和输出分开。使用 MVC 应用程序被分成三个核心部件：模型、视图、控制器。它们各自处理自己的任务

    视图

    视图是用户看到并与之交互的界面。

    模型

    模型表示企业数据（数据模型：dao）和业务规则及操作（业务模型：service）

    控制器(调度中心)

    控制器接受用户的输入并**调用**模型和视图去完成用户的需求。

    常见的 MVC 组件：Struts，Spring MVC，JSF

  - MVC 的优点

    低耦合性：因为模型与控制器和视图相分离，所以很容易改变应用程序的数据层和业务规则。

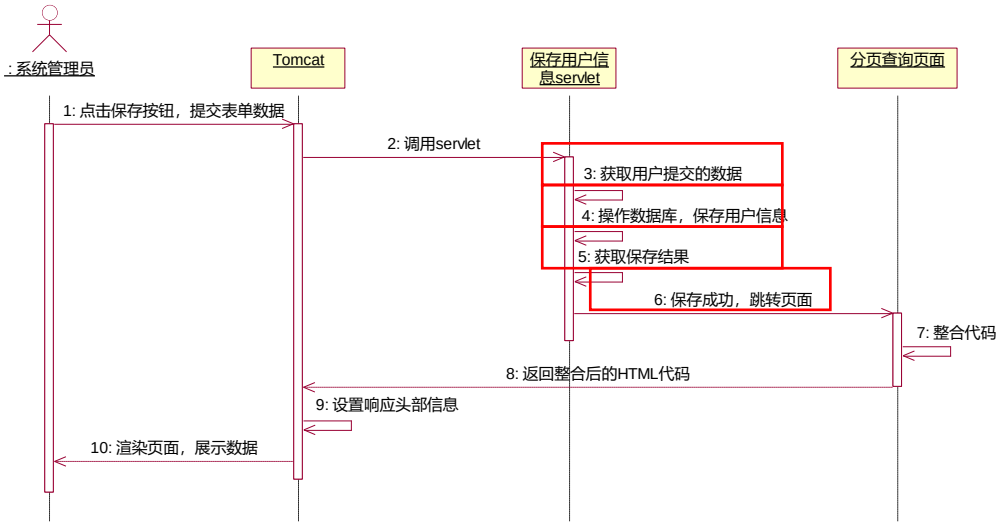
    高重用性和可适用性:其中的一部分更改不会导致整个整体修改，例如，很多数据可能用 HTML TABLE 来表示，但是也有可能用 HTML DIV 来表示，而这些表示所需要的仅仅是改变视图层的实现方式，而控制层和模型层无需做任何改变。

- 将系统架构改为 MVC 模式，增加 Service，Dao 层

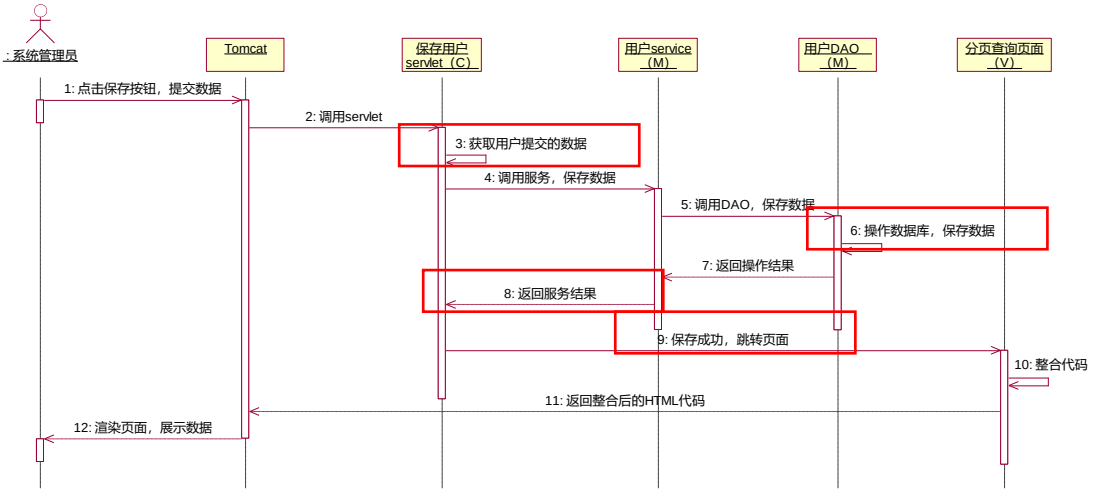
  DAO：（Data Access Object，数据访问对象）将低级别的数据访问逻辑与高级别的业务逻辑分离

- **架构模式**
  - 通过类和对象组合在一起形成一个特定结构完成业务的开发方式
  - MVC 架构模式并没有增加业务的流程，只不过将固定的流程分解到不同的对象中去完成，这样，降低了功能和功能之间的关联性，也同时降低了业务和业务之间的关联性，这样可以更加容易地扩展系统的功能

- 未使用 MVC 架构的流程：



- 使用 MVC 架构的流程：



- 让程序具备更好的通用性以及扩展性的方式
  - 过滤器（AOP）
  - MVC 架构模式
  - 工具类
  - XML、Properties 配置文件
- XML 文件和 HTML 文件的主要区别：
  - XML 是区分大小写，HTML 不区分大小写
  - XML 必须含有结尾标识
</BR>，HTML 可以没有结尾标识
  - 使用单一元素时，XML 必须使用/结尾
，HTML 中可以没有
  - XML 的属性值必须包含在引号中 name="xxx",HTML 中属性可以不用引号包含 name=123
  - HTML 中可以含有不带值的属性 **readonly**, XML 不行
  - **XML 主要是保存数据的一种文件格式，HTML 主要是显示数据的一种文件格式**

- XML：

  可扩展标记语言.

  重点掌握读写的方法

  - **DOM** 方式：

  是面向模型的，一次将 XML 文档加入内存，编程简单，适合小型文件的解析

  当文件大的场合，读取后会将文件的内容全部加载到内存中，那么会增加内存负担，影响程序执行效率，但是正因为文档结构全部加载到内存中，所以查找元素快

  - **SAX** 方式 – **dom4j**：

  是面向事件的，读一部分解析一部分，编程复杂一些，适合大型文件的解析.

  触发事件的场合才会去读取文件中的部分，所以内存的使用不会很大，但是因为是现查现找，所以查找效率上相对来说低一些。

- 使用第三方组件 dom4j 完成 XML 配置文件的解析
  - 引入组件相关的 JAR 包。
    - dom4j-1.6.1.jar
    - jaxen-1.1-beta-6.jar
  - 生成和读取 XML 文件。
- 使用 XPATH 快速定位 XML 文件内容

第十天 系统扩展&JSTL

- 程序的开发效率和运行效率
  - SQL
  - 增强 for 循环
  - 工具类
  - 数据库连接池
  - JSTL（标签库）
- 提高 JSP 的开发效率
  - 简化 JSP 中的 JAVA 代码
  - JSTL 技术可以使用标签的方式来代替 JSP 中 JAVA 代码。
  - 使用 JSTL 标签库和 EL 表达式联合使用，简化 JSP 中的 JAVA 代码
- 在 JSP 中简化 JAVA 代码
  - EL 表达式
    - EL 表达式中含有内置对象，通过内置对象可以访问服务器中保存的数据以及表单中传递的数据

pageContext(JSP)	**➔ pageScope(EL)**
request(JSP)	**➔ requestScope(EL)**
session(JSP)	**➔ sessionScope(EL)**
application(JSP)	**➔ applicationScope(EL)**
request.getParameter (Java) ➔ param(EL)	
request.getParameterValues (Java) ➔ paramValues(EL)	

    - EL 表达式可以访问对象及属性，且访问时，不需要指定范围

${ 对象.属性 } ➔ ${user.username}

- EL 表达式如果没有指定访问数据的范围，那么默认从最小的范围中获取，获取不到，自动扩大范围递归查找

  pageContext ➔ request -➔ session ➔ application

- EL 表达式访问对象的属性，其实访问的是对象属性的 get 方法。属性不是必须的

  ${user.username} ➔ user.getUsername()

- EL 表达式无法完成逻辑操作，如循环，条件判断，必须和 JSTL 标签库联合使用。
- EL 表达式中的特定的表达式语法规则
  - 字符串可以使用单引号

    ${'abc'}

  - 双等号比较字符串是比较内容，可以使用 eq 代替， 不等使用 ne 代替

    ${"abc" == "abc"} or ${"abc" eq "abc"} or ${"abc" ne "def"}

  - Empty 为空判断

    ${empty user} or ${not empty user}

- JSTL 标签库
  - JSTL 标签库可以使用标签的方式来简化 JSP 中的 JAVA 代码
  - 使用 JSTL 标签库时，需要引入相关的 JAR 包并导入标签库的指令信息，否则无法使用
  - JSP 解析标签时，会直接执行标签所对应的 JAVA 程序，将程序的执行结果输出到浏览器中。
  - 常用的标签
    1. **<c:if>**
    2. <c:out>
    3. **<c:choose><c:when><c:otherwise>**
    4. **<c:forEach>**

第十一天 FushionChartsFree&Fileupload

- 自定义函数库
  1. 自己函数库的配置文件 egov.tld, 将文件保存在 WEB-INF 目录下
  2. 文件内容参考 JSTL 中自带的函数库文件 fn.tld
  3. 在 JSP 文件中导入自己的函数库
  4. 在 EL 表达式中使用函数 ：**${函数库的引用名称：函数的名称（函数的参数...）}**
- 将数据以 flash 方式展现(**FusionChartsFree**)
  - JS +Flash+XML
- 使用第三方组件（JFreeChart）将数据以图片的方式展现
  - 柱状图
  - 饼状图
  - 在浏览器中显示图片
- 将数据以 PDF 方式展现(**iReport**)

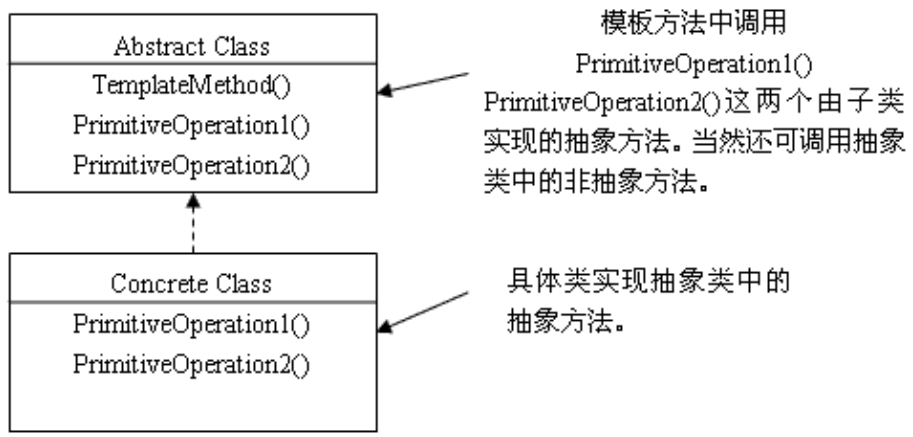
- 由工具生成 PDF 模板文件，然后由 java 程序调用模板文件并设置数据，生成 PDF 文件

- 使用第三方组件完成文件上传功能(**commons-fileupload**)
  - 文件上传 ：将本地的文件通过网络拷贝到服务器中。
  - 文件下载：将服务器的文件通过网络拷贝到本地
  - **文件上传时，表单标签中应该增加 enctype 属性。**
  - **文件上传时，表单元素的类型为 file**
  - **因为文件上传是将文件内容放置在请求体中传递到服务器，可是 get 方式提交数据，根本没有请求体，所以文件上传不支持 GET 方式提交数据。**
  - 文件上传时，设置请求对象的字符编码不起作用，会出现中文乱码，如果想要获取正确的中文，需要使用组件的特殊方法

第十二天 核准件反馈

- 使用**模板方法**设计模式改善控制器的实现

  这是一个很简单的模式，却被非常广泛的使用。之所以简单是因为在这个模式中仅仅使用到了**继承**关系，模板方法（Template Method）模式定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。这里的算法的结构，可以理解为你根据需求设计出来的业务流程。特定的步骤就是指那些可能在内容上存在变数的



- 模式中涉及到的细节：
  - 类之间一定要有继承关系，
  - 父类中应该有一个方法来定义算法的骨架。
  - 然后在方法中调用抽象方法(业务方法)，
  - 而抽象方法的具体实现由子类完成。

  HttpServlet 中 service 方法体现了模板方法设计模式

  模板方法设计模式和 java 动态绑定机制相关联(必须有多态机制)

- 项目相关的内容介绍
  - 人员组织结构 ：21
    1. 1 个项目经理(PM)
    2. 2 个设计人员
    3. 3 个开发小组,每个小组 6 个人 (**境内业务**， 境外业务， IC 卡业务)

- 1 个小组长(TL, GL)
- **1 个高级软件工程师 （SE）**
- 4 个开发人员（PG：ProGramer）

➢ 项目开发周期：6 个月

1. 2 个月， 客户需求分析和设计
2. 1.5 个月，开发业务功能 coding
3. 2.5 个月，测试系统（单元测试， 功能测试， 性能测试， 压力测试。）

➢ 项目开发技术 ：Struts2 + Spring + Hibernate + AJAX（JQuery）

➢ 项目中学习到的内容： 团队合作

- 软件开发原则
  - **开闭原则（OCP）**
    1. **开发程序时一定要考虑程序的扩展性，系统一定要对扩展开放， 对修改关闭。**
  - 里氏代换原则（LSP）
    1. 任何类出现的地方，子类一定可以出现（多态）
  - **依赖倒转原则（DIP）**
    1. **尽量依赖于抽象的对象，而不要依赖于具体的对象（面向接口编程）**
  - 接口隔离原则（ISP）
    1. 尽量采用小的，独立的接口访问对象，而不要采用大的接口进行数据的通信，使通信尽可能的窄。
  - 合成复用原则
    1. 尽量采用合成和聚合的方式来完成功能的复用，而不要采用继承的方式。（Service + Dao）
  - **迪米特法则（松耦合）**

一个软件实体（对象）尽可能少的和其他实体（对象）发生作用。

第十三天 系统扩展

- **监听器（监听者、被监听者（事件源，事件的发起人）、特殊的事件）**
  - 生活中，监听器是在特定场合对特定的事件进行监听的设备或仪器
  - Web 系统中，监听器可以监听服务器对象状态的变化以及属性的操作。只要监听到，可以执行相应的操作
  - 实现监听器
    1. 创建监听器类，实现特定的接口(javax.servlet.ServletContextListener)

javax.servlet.ServletContextListener ➔ Web 应用对象监听器

javax.servlet.ServletContextAttributeListener ➔ Web 应用对象属性监听器

javax.servlet.http.HttpSessionListener ➔ 会话对象监听器

javax.servlet.http.HttpSessionAttributeListener ➔ 会话对象属性监听器

javax.servlet.ServletRequestListener ➔ 请求对象监听器

javax.servlet.ServletRequestAttributeListener ➔ 请求对象属性监听器

    2. 修改 web.xml，增加对过滤器的支持

```
<!-- 监听器 -->
```

```
<listener>
  <listener-class>com.bjpowernode.egov.web.ServerStartupListener</listener-class>
</listener>
```

- **使用 Tomcat 自带的连接池操作（tomcat-dbcp）（tomcat-Data Base Connection Pool）**
  - 在 CATALINA_HOME/webapps/webapp/META-INF/ 目录下新建 context.xml 文件，文件内容如下：

```
<?xml version="1.0" encoding="GBK"?>
<Context reloadable="true">
    <Resource
            name="jdbc/egov"
            auth="Container"
            type="javax.sql.DataSource"
            maxActive="100"
            maxIdle="30"
            maxWait="10000"
            username="egov"
            password="bjpowernode"
            driverClassName="oracle.jdbc.driver.OracleDriver"
            url="jdbc:oracle:thin:@192.168.1.xxxx:1521:bjpowernode" />
</Context>
```

  - 将连接数据库的驱动 ojdbc14.jar 拷贝到 CATALINA_HOME/webapps/webapp/WEB-INF/lib/ 目录下。
  - 在需要获取数据库连接对象的位置编写如下 java 代码：

```
Context context = new InitialContext();
DataSource ds = (DataSource)context.lookup("java:/comp/env/jdbc/egov");
Connection conn = ds.getConnection();
```

- 注意事项：当我们修改 context.xml 文件的时候，必须将 C:\apache-tomcat-6.0.32\conf\Catalina\localhost**webappname.xml** 删除，重新启动 Tomcat 服务器 context.xml 文件才算是彻底修改。
- **JNDI （Java Naming and Directory Interface） java 命名和目录服务接口**
  - 降低程序和程序之间的耦合性
  - 降低设备和设备的耦合性
  - Tomcat 服务器实现了 JNDI 服务，启动 Tomcat 服务器，等同于启动 JNDI 服务器
  - JNDI 是 SUN 制定的一套规范，这套规范可以和 JDBC 进行类比，JDBC 也是一套规范，我们面向 JDBC 接口调用就可以获取数据库中的数据。同样，我们面向 JNDI 接口调用，可以获取“资源”，这个资源不是在底层数据库中。而是在 JNDI 上下文当中。
  - JNDI 提供了一种服务，这个服务可以将“名称”和“资源”绑定在一起，然后程序员通过面向 JNDI 接口调用方法 lookup 可以查找到相关的资源。

- **GOF95 设计模式**
  - 什么是设计模式? **可重复利用的解决方案**
  - 1995 有 GOF（4 人组）提出，GOF95
  - 设计模式从大体上分为三类
    - 创建型模式：应用在系统创建对象时，动态的决定怎么创建对象
    - 结构型模式：将类和对象组合在一起，形成更大的结构
    - 行为型模式：在不同的类之间划分职责和对算法抽象

我们学习过的设计模式：

单例模式（创建型设计模式）

装饰者模式（结构型设计模式）

模板方法模式（行为型设计模式）

- **工厂模式（创建型设计模式）（不属于 23 种设计模式之一）**
  - 生活中，工厂可以创建产品的。
  - 工厂中创建的产品应该具有相同的特征（颜色， 型号，品牌， 功能）
  - 工厂中创建的产品不应该是一模一样的，应该具有细微的区别，可以根据一些属性进行区分。

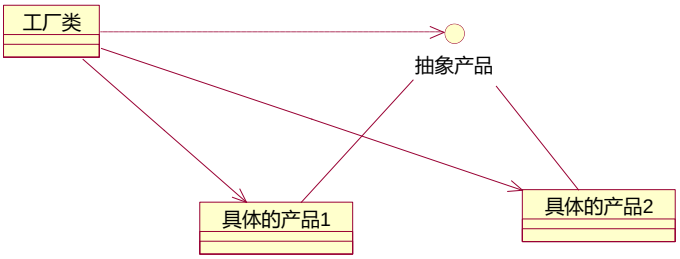
  使用代码实现：
  - 代码实现中应该有一个可以创建对象的类，将这个类称之为**工厂类**，创建对象的方法就类似于工厂中流水线，所以方法也称之为**工厂方法**。
  - 工厂类中创建的多个产品（对象）具有相同的特征。可以采用继承相同的类或实现相同的接口来实现。
  - 通过工厂方法创建对象时，可以传递不同的参数来进行区分。获取不同的对象。

- 工厂模式的模拟应用场景：
  - 小卖店提供饮料，可以作为工厂
    1. 工厂类：Shop
    2. 工厂方法：getDrink()
  - 将产品的特征抽象出来
    1. Interface Drink
  - 具体的多个产品，产品应该符合特征
    1. Kele implements Drink
    2. MD implements Drink
  - 工厂方法应该具有参数，获取不同的对象，获取的对象应该具有相同特征。
    1. Drink getDrink(String drinkType);

  早期的工厂模式的代码实现采用逻辑判断来创建不同的对象，但是这种方式违背了 OCP 开闭原则。

  现在的工厂模式创建对象时一般采用反射创建，这种方式可以遵循 OCP 开闭原则。
- 工厂模式的优点：
  - 隐藏了产品的创建细节。降低了用户和产品创建之间的耦合性
  - 创建产品的方法使用反射实现，遵循 OCP 开闭原则，有利于扩展。

- 创建工厂模式需要注意，
  - 一定要有工厂类和工厂方法
  - 工厂方法一定要返回抽象的对象。

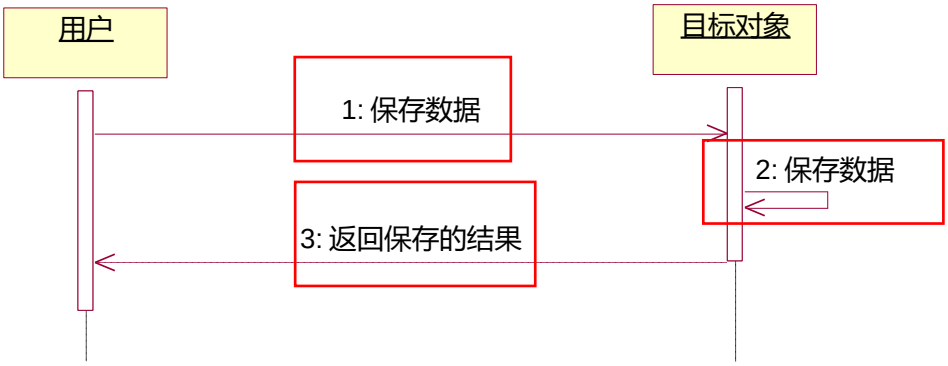


工厂设计模式
1) 隐藏产品的创建细节
2) 遵循OCP的设计原则
3)在当前的项目中，一般创建产品为使用反射完成
4)如果创建一系列的产品，不遵循OCP原则。

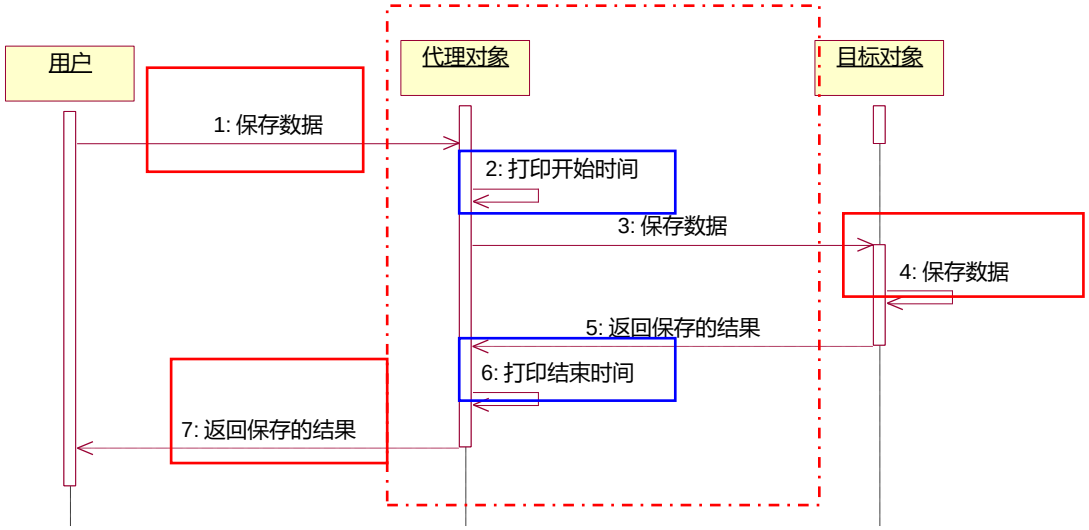
实现模式的过程
1）工厂类，创建对象的类
2）工厂类应该提供一个可以创建抽象产品的方法
3）工厂类创建的产品一定要具有相同的特征

- **代理模式（结构型设计模式）**
  - 目标对象和代理对象
  - 目标对象表示实现要完成业务操作的对象(租户)
  - 代理对象表示代替目标对象执行业务操作的对象（房屋中介）
  - 实际的业务操作一定由目标对象来完成。（代理对象应该包含一个目标对象的引用。）
  - 目标对象和代理对象应该具有相同的行为。（继承相同的类或实现相同的接口）
  - 代理模式从实现的方式来讲，分成**静态代理模式**和**动态代理模式**。

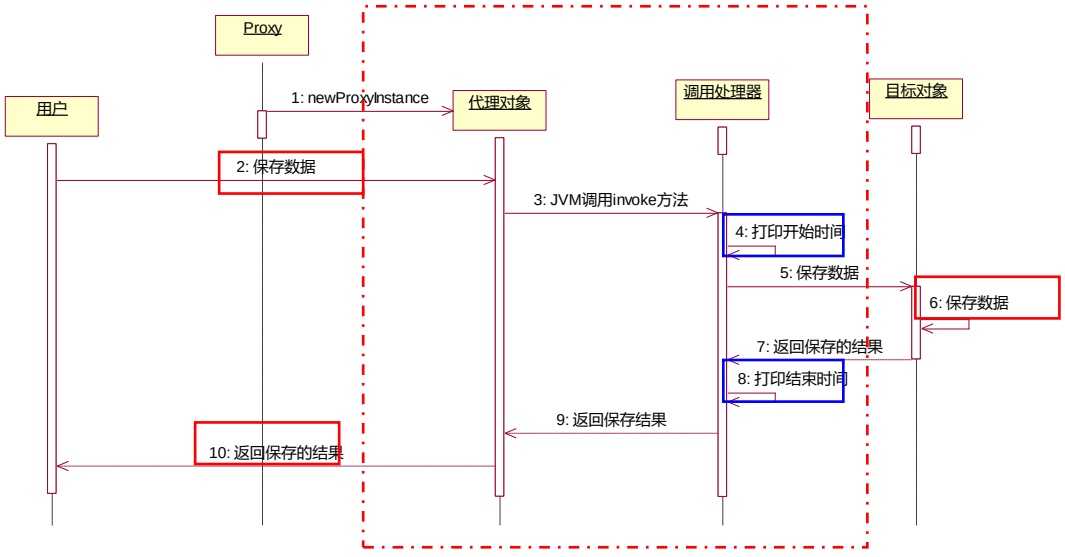
普通的流程:



静态代理的流程


用户
代理对象
目标对象
1: 保存数据
2: 打印开始时间
3: 保存数据
4: 保存数据
5: 返回保存的结果
6: 打印结束时间
7: 返回保存的结果

动态代理模式


Proxy
用户
1: newProxyInstance
代理对象
调用处理器
目标对象
2: 保存数据
3: JVM调用invoke方法
4: 打印开始时间
5: 保存数据
6: 保存数据
7: 返回保存的结果
8: 打印结束时间
9: 返回保存结果
10: 返回保存的结果

第十四天 系统扩展

- 项目中存在的问题
  - 控制器继承了 HttpServlet，就和 web 系统紧密耦合在一起，违背了迪米特法则（Struts2 解决）
  - 控制器存在线程安全问题。在实际的开发过程不容易发现问题。（Struts2 中的 Action 多例的）
  - SQL 文写死在程序当中，不利于扩展，违背 OCP 开发原则.（ iBatis 框架解决 SQL 语句配置问题）
- 将之前学习过的内容融合在一起解决项目中存在的问题
  - XML
  - XPATH
  - DOM4J
  - 工厂模式
  - 动态代理模式
  - ThreadLocal
  - Clone
  - 软件开发原则
  - 监听器