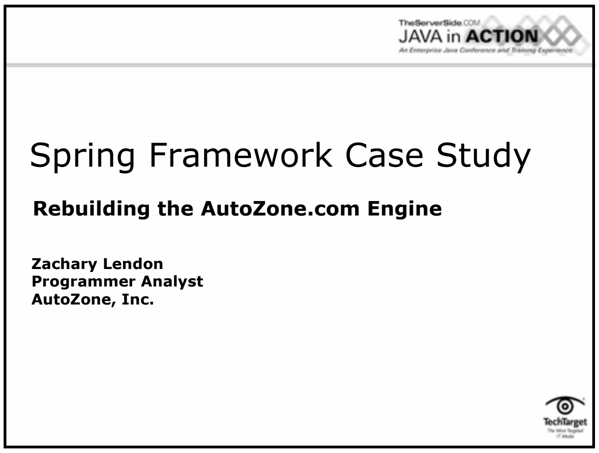
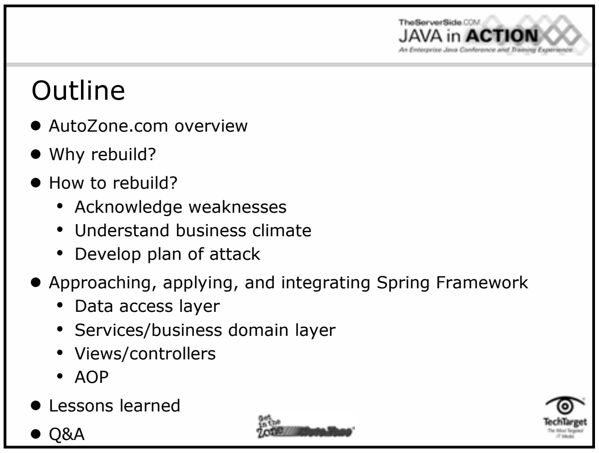


Spring Framework Case Study

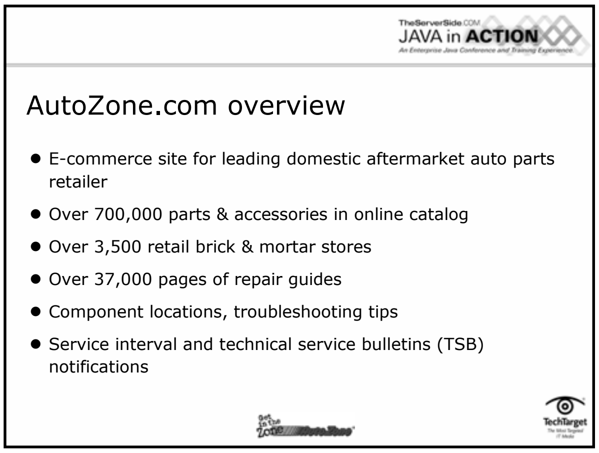
Rebuilding the AutoZone.com Engine

Zachary Lendon
Programmer Analyst
AutoZone, Inc.



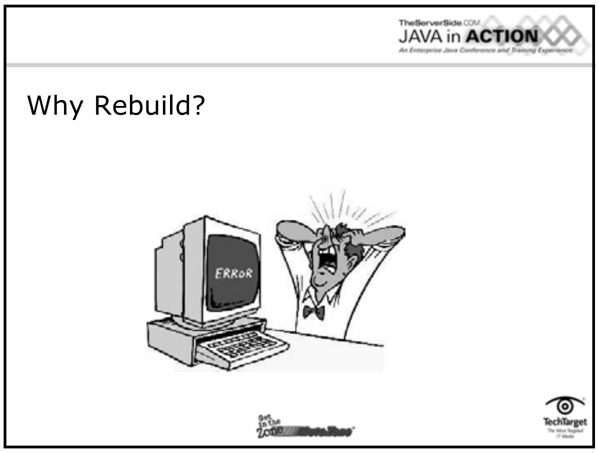
Outline

- AutoZone.com overview
- Why rebuild?
- How to rebuild?
  - Acknowledge weaknesses
  - Understand business climate
  - Develop plan of attack
- Approaching, applying, and integrating Spring Framework
  - Data access layer
  - Services/business domain layer
  - Views/controllers
  - AOP
- Lessons learned
- Q&A

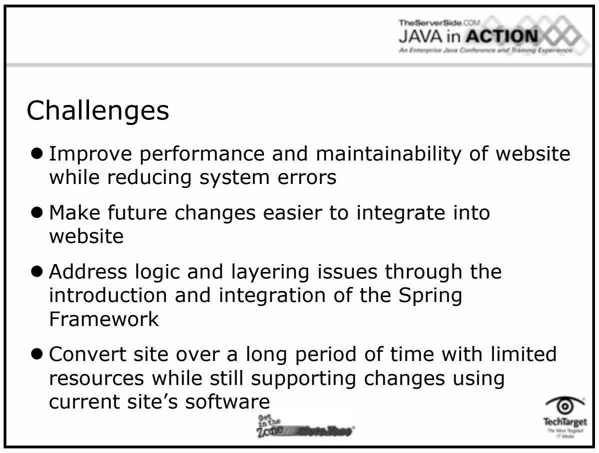


AutoZone.com overview

- E-commerce site for leading domestic aftermarket auto parts retailer
- Over 700,000 parts & accessories in online catalog
- Over 3,500 retail brick & mortar stores
- Over 37,000 pages of repair guides
- Component locations, troubleshooting tips
- Service interval and technical service bulletins (TSB) notifications



Why Rebuild?



Challenges

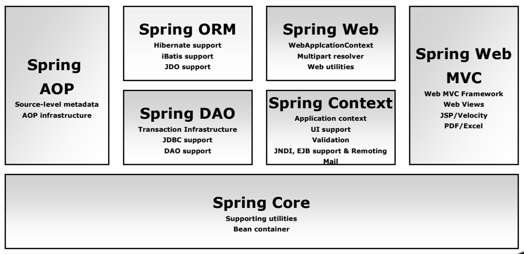
- Improve performance and maintainability of website while reducing system errors
- Make future changes easier to integrate into website
- Address logic and layering issues through the introduction and integration of the Spring Framework
- Convert site over a long period of time with limited resources while still supporting changes using current site's software

Why Rebuild?

- For the customers
  - Improve reliability of site by reducing errors and improving performance
- For the company
  - Save money
    - Reduce additional hardware needs in future
    - Reduce future software development/support costs
  - Make money
    - Improve customer experience -> improve Sales
- For the developers
  - Upgrade technology of website
  - Increase flexibility for future improvements/fixes
  - Ease future integration with other project work
  - Pride factor

Introducing the Spring Framework

•Components we used (in blue)



How to Rebuild?

- Acknowledge your biggest weaknesses/ opportunities
  - AutoZone.com used unreliable, proprietary, and overly-complex data access layer
    - Initially developed to communicate with mainframe DB2 tables, became out-dated several years ago after switch to Informix
    - Effort necessary to develop and support the data access and its resulting code at times was overwhelming
    - Re-inventing the wheel
    - Did not own source
    - Product support: one person
    - Caused issues with garbage collections
    - Added points of failure

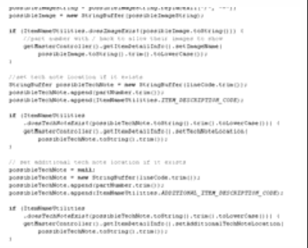
TheServerSide.COM JAVA in ACTION
An Enterprise Java Conference and Training Experience

How to Rebuild?

Acknowledging weaknesses (cont'd).

- Weak MVC implementation
  - Controllers containing state information, instance data, and business domain logic
  - Views containing business logic

Controller example from "CatalogController"



```
possibleImageString = possibleImageString.replaceAll("-", "");
possibleImage = new StringBuffer(possibleImageString);

if (ItemNameUtilities.doesImageExist(possibleImage.toString())) {
    //part number with / back to allow their images to show
    getMasterController().getItemDetailInfo().setImageName(
        possibleImage.toString().trim().toLowerCase());
}

//set tech note location if it exists
StringBuffer possibleTechNote = new StringBuffer(lineCode.trim());
possibleTechNote.append(partNumber.trim());
possibleTechNote.append(ItemNameUtilities.ITEM_DESCRIPTION_CODE);

if (ItemNameUtilities
    .doesTechNoteExist(possibleTechNote.toString().trim().toLowerCase())) {
    getMasterController().getItemDetailInfo().setTechNoteLocation(
        possibleTechNote.toString().trim());
}

// set additional tech note location if it exists
possibleTechNote = null;
possibleTechNote = new StringBuffer(lineCode.trim());
possibleTechNote.append(partNumber.trim());
possibleTechNote.append(ItemNameUtilities.ADDITIONAL_ITEM_DESCRIPTION_CODE);

if (ItemNameUtilities
    .doesTechNoteExist(possibleTechNote.toString().trim().toLowerCase())) {
    getMasterController().getItemDetailInfo().setAdditionalTechNoteLocation(
        possibleTechNote.toString().trim());
}
```

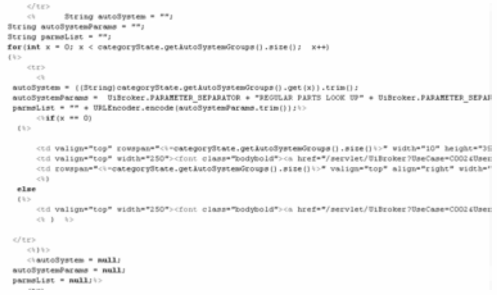
Get in the Zone MotoZone

TechTarget
The Most Targeted IT Media

TheServerSide.COM JAVA in ACTION
An Enterprise Java Conference and Training Experience

View Example:

Catalog page (circa 07/05)



```
    </tr>
    <%      String autoSystem = "";
String autoSystemParams = "";
String parmsList = "";
for(int x = 0; x < categoryState.getAutoSystemGroups().size();  x++)
{%>
    <tr>
      <%
 autoSystem = ((String)categoryState.getAutoSystemGroups().get(x)).trim();
 autoSystemParams =  UiBroker.PARAMETER_SEPARATOR + "REGULAR PARTS LOOK UP" + UiBroker.PARAMETER_SEPAR
 parmsList = "" + URLEncoder.encode(autoSystemParams.trim());%>
      <%if(x == 0)
 {%>

      <td valign="top" rowspan="<%=categoryState.getAutoSystemGroups().size()%>" width="10" height="35
      <td valign="top" width="250"><font class="bodybold"><a href="/servlet/UiBroker?UseCase=C002&User
      <td rowspan="<%=categoryState.getAutoSystemGroups().size()%>" valign="top" align="right" width="
      <%}
 else
 {%>
      <td valign="top" width="250"><font class="bodybold"><a href="/servlet/UiBroker?UseCase=C002&User
      <% }  %>

 </tr>
    <%}%>
    <%autoSystem = null;
 autoSystemParams = null;
 parmsList = null;%>
```

Get in the Zone MotoZone

TechTarget
The Most Targeted IT Media

TheServerSide.COM JAVA in ACTION
An Enterprise Java Conference and Training Experience

How to Rebuild? (cont'd)

- **Understand business climate**
  - Maintain current site functionality
  - Be able to fix critical 'bugs' throughout 'upgrade' life-cycle
  - Limited resources
  - Limited time

Get in the Zone MotoZone

TechTarget
The Most Targeted IT Media

How to Rebuild? (cont.)

- **Develop plan of attack**
  - Methodical, phased back-to-front end 'layered' approach – data access, service, business domain, controllers, view, etc.
  - Once enough back and middle-tier functionalities are defined, slowly introduce front-end change-over; new views/controllers should interact primarily with other new components
  - Extract and re-factor business logic from previous "architecture" into service and business domain layer.
  - After significant milestones are reached, educate developers/business team on various approaches to encourage adoption

Approaching Spring Data Access

- We chose Spring's 'JDBC abstraction layer'
- Important factors for us in defining approach:
  - Legacy Data Model
  - Rich Spring API: exception hierarchy and transaction support
  - Ease of migration for developers familiar with JDBC
  - Less code the better
- Use DAO Interfaces and DAO Implementation classes
- Use service layer as 'wrapper' to DAOs

Applying Spring Data Access

- Laying the groundwork
  - Configure property files

```
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
            <list>
                        <value>/WEB-INF/jdbc.properties</value>
                        <value>/WEB-INF/sql.properties</value>
            </list>
    </property>
</bean>
```

TheServerSide.COM JAVA in ACTION
An Enterprise Java Conference and Training Experience

Define DataSource

```
<bean
id="dataSource"
class="org.apache.commons.dbcp.BasicDataSource">
<property name="url">
<value>${ecom.jdbc.url}</value>
</property>
<property name="driverClassName">
<value>${ecom.jdbc.driver}</value>
</property>
<property name="username">
<value>${ecom.jdbc.username}</value>
</property>
<property name="password">
<value>${ecom.jdbc.password}</value>
</property>
<property name="testOnBorrow">
<value>true</value>
</property>
<property name="validationQuery">
<value>SELECT FIRST 1 [redacted]</value>
</property>
<property name="maxActive">
<value>32</value>
</property>
<property name="maxIdle">
<value>32</value>
</property>
<property name="maxWait">
<value>10000</value>
</property>
</bean>
```

TheServerSide.COM JAVA in ACTION
An Enterprise Java Conference and Training Experience

Applying Spring Data Access (cont).

Our standard DAO implementation:

- Extends org.springframework.jdbc.core.support.JdbcDaoSupport
- Test class extends *AbstractTransactionalDataSourceSpringContextTests*
- DAOs contain inner classes that extend Spring's MappingSqlQuery Class for object mapping (as needed)

TheServerSide.COM JAVA in ACTION
An Enterprise Java Conference and Training Experience

Example: Retrieving Store Information

- DAO Inner Class

```
class StoreQuery extends MappingSqlQuery {
```

- DAO Inner Class Constructor

```
public StoreQuery(final DataSource dataSource, final String sql) {
    setDataSource(dataSource);
    setSql(sql);
    declareParameter(new SqlParameter(Types.INTEGER));
    compile();
}
```

- **Define Query objects (at class level)**

```
private StoreQuery zipToStoreQuery;
private StoreQuery sisterStoreQuery;
private StoreQuery storeNumberInfoQuery;
private StoreHoursQuery storeHoursQuery;
```

- **Initialize them**

```
 */
protected void initDao() throws Exception {
    zipToStoreQuery = new StoreQuery(getDataSource(), ZIP_TO_STORE_SQL);
    sisterStoreQuery = new StoreQuery(getDataSource(), SISTER_STORE_SQL);
    storeNumberInfoQuery =
        new StoreQuery(getDataSource(), STORE_INFO_BY_STORE_NUMBER_SQL);
    storeHoursQuery = new StoreHoursQuery(getDataSource(), STORE_HOURS_SQL);
}
```

- Map ResultSet to Object (in inner class)

```
public Object mapRow(final ResultSet rs, final int rowNum)
    throws SQLException {

    final Store store = new Store();
    final Address address = new Address();
    final Phone phone = new Phone();

    if (StringUtils.equalsIgnoreCase(rs.getString(15), "Y")) {
        store.setMonOpen(OPEN_24);
        store.setTueOpen(OPEN_24);
        store.setWedOpen(OPEN_24);
        store.setThuOpen(OPEN_24);
        store.setFriOpen(OPEN_24);
        store.setSatOpen(OPEN_24);
        store.setSunOpen(OPEN_24);
        store.setMonClose(BLANK);
        store.setTueClose(BLANK);
        store.setWedClose(BLANK);
        store.setThuClose(BLANK);
        store.setFriClose(BLANK);
        store.setSatClose(BLANK);
        store.setSunClose(BLANK);
    } else {
        store.setMonOpen(
            StringUtilities.formatHourSuffix(rs.getString(1)));
        store.setTueOpen(
            StringUtilities.formatHourSuffix(rs.getString(2)));
        store.setWedOpen(
            StringUtilities.formatHourSuffix(rs.getString(3)));
        store.setThuOpen(
            StringUtilities.formatHourSuffix(rs.getString(4)));
        store.setFriOpen(
```

- Finally - Get Data!

```
public Store findStoreInfoByStoreNumber(int storeNumber)
    throws DataAccessException {
    if (logger.isDebugEnabled()) {
        logger.debug(
            "Entering StoreDaoImpl.findStoreInfoByStoreNumber, store number is: "
                + storeNumber);
    }

    Store store = null;

    try {
        store = (Store) storeNumberInfoQuery.findObject(storeNumber);
    } catch (IncorrectResultSizeDataAccessException e) {
        //disregard
    }

    if (logger.isDebugEnabled()) {
        logger.debug("Exiting StoreDaoImpl.findStoreInfoByStoreNumber");
    }

    return store;
}
```

- In retrospect, we could simplify further by…

- Adding Logging Aspect/Interceptor to handle logging
- Removing statements originally written for "clarity" purposes

End up with something like…

```
public Store findStoreInfoByStoreNumber(int storeNumber)
        throws DataAccessException {
  Store store = (Store) storeNumberInfoQuery.findObject(storeNumber);
  return store;
}
```

Integrating Spring – Data Access

- In our initial code-base, only servlet had access to request object
- In init() for main servlet
  - ContextLoaderListener (web.xml) loads beans into ServletContext
  - In main servlet, define public static variable equal to WebApplicationContextUtils.
    getRequiredWebApplicationContext(getServletContext())
- Directly in current "database layer" :

  ```
  ApplicationContext ctx = UiBroker.getApplicationContext();
  CustomerDao dao = (CustomerDao) ctx.getBean("customerDao");
  ```

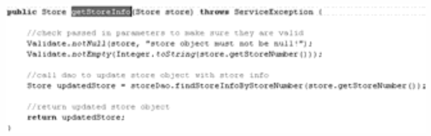
  Where essentially all previous data access logic resided
- Can easily co-exist within or alongside prior database layer
- Rarely call DAO directly – usually access through service or business domain layer.   This allows for phasing out of old database/business layer(s).
- As Spring becomes more widespread in code-base, can re-define how applicationContext is accessed if necessary.

Approaching Service Layer

- Layer between controllers and DAO that exposes business logic.
- Leverages DAOs and business domain-level objects to bundle information for controllers and remote systems.
- Use interfaces!
- Be wary – all the business logic doesn't have to go in the service itself.
- Much of our "mis-layered" code either belongs in this layer or should be accessed using this layer.
- Define transactions at this level.
  - Transaction across DAOs
  - Involve JMS

Applying Spring – Service layer

- Example: Implementation for service to retrieve store information



```
public Store getStoreInfo(Store store) throws ServiceException {

    //check passed in parameters to make sure they are valid
    Validate.notNull(store, "store object must not be null!");
    Validate.notEmpty(Integer.toString(store.getStoreNumber()));

    //call dao to update store object with store info
    Store updatedStore = storeDao.findStoreInfoByStoreNumber(store.getStoreNumber());

    //return updated store object
    return updatedStore;
}
```

- Some key thoughts
  - When layers are properly separated, methods will often look simple…
  - Opportunity for better defining service and business domain layer exists

Applying Spring – Service layer Configuration

- Create Transaction Proxy for Service

```
<bean
id="storeInfoService"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
<property name="target">
<ref local="storeInfoServiceTarget" />
</property>
</bean>
```

- Create Target For Proxy

```
<bean
id="storeInfoServiceTarget"
class="com.autozone.www.service.StoreInfoServiceImpl">
<property name="storeDao">
<ref local="storeDao" />
</property>
</bean>
```

- Transaction Attributes
  - o Default in above example
  - o Isolation levels
  - o Propagation levels
- Pre and post interceptors defined here

Integrating Spring – Service Layer

- Similar to DAO integration
- If used across legacy controller layer (i.e. multiple controllers), define in 'Controller' base class (or some other generically accessible location)

```
public CustomerInfoService getCustomerInfoService() {
   // retrieve Spring application context
   ApplicationContext ctx = UiBroker.getApplicationContext();
   // retrieve customer info service bean from Spring
   CustomerInfoService service =
            (CustomerInfoService) ctx.getBean("customerInfoService");
   return service;
}
```

- Simply call 'getCustomerInfoService().method…'
- Otherwise, above logic in individual controller…

Approaching Spring

- View/Controller Layer:
  - Move site towards Spring Web MVC
  - Why we chose it
    - Promote flexibility
      - Ease of switching between multiple controller and view options
    - Ease of taking advantage of other Spring-based components through Dependency Injection
    - Allows easier integration of future non-AutoZone.com-specific project work (that hopefully will also leverage the Spring Framework)
- Key Motivation:
  - Remove ties between business logic and presentation logic

Approaching Spring MVC

- We chose Struts Tiles (with JSP pages) as primary view technology
- Map virtual URL's to Controllers
  - Mainly use Spring supplied implementations
    - BeanNameUrlHandlerMapping: we use primarily for dynamic pages
    - SimpleUrlHandlerMapping: use primarily for static pages
      - Spring-provided UrlFilenameViewController
  - For our custom controllers
    - MultiActionControllers (MAC)
      - Use MethodNameResolver for defining handler/action to map request to
      - Similar to previous approach (reflection-based) and Struts

Approach Spring MVC with Tiles

- Why Tiles, why not Velocity, SiteMesh, etc?
  - Html in such a state that move to Tiles provides simple means for 'clean up' of view layer.
  - Smallest 'idealogy' change from previous architecture.
  - Build with view layer flexibility in mind.

Applying Spring MVC with Tiles

- Define 'Tiles Configurer' Bean
  - To load tiles definitions
  - 'Configure' Tiles

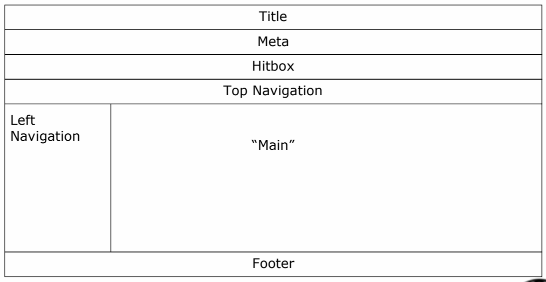
```
<bean id="tilesConfigurer"

class="org.springframework.web.servlet.view.tiles.TilesConfigurer">
    <property name="definitions">
    <list>
    <value>/WEB-INF/tiles-layout.xml</value>
    </list>
    </property>
</bean>
```

Applying Spring MVC and Tiles (cont'd)

- Define 'TilesView' View Resolver
  - Maps view names to view implementations

```
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="viewClass">
<value>org.springframework.web.servlet.view.tiles.TilesView</value>
</property>
</bean>
```

Layout Components of Tiles Page



Define Tiles-Layout Configuration

•Base/Root page

```
<definition name=".root" path="/WEB-INF/views/tiles/root.jsp">
<put name="header" value=".header"/>
<put name="hitbox" value="/WEB-INF/views/tiles/js/hitbox.jsp"/>
<put name="meta" value="/WEB-INF/views/tiles/html/meta.jsp"/>
<put name="left_nav" value="/WEB-
INF/views/tiles/left_nav/left_nav_df.jsp"/>
<put name="footer" value="/WEB-INF/views/tiles/bottom.jsp"/>
</definition>
```

Applying Spring MVC

- **AutoZone.com Home Page**
  - User requests http://www.autozone.com
  - <welcome-file> in web.xml is home.htm
  - Web Server’s docroot has empty file called ‘home.htm’
  - DispatcherServlet mapping is *.htm
  - Request mapped to Spring’s DispatcherServlet
  - Looking for first match for ‘home.htm’

```
<bean name="/home.htm"
class="org.springframework.web.servlet.mvc.ParameterizableViewController">
   <property name="viewName"><value>index</value></property>
</bean>
```

Applying Spring MVC and Tiles

- **Home example (cont).**
  - Searching for viewResolver for index – TilesView is viewClass
  - Find 'index' Tiles definition

```
<!-- Home Page -->
<definition name="index" extends=".root"
controllerClass="com.autozone.www.control.tiles.MainTileController">
<put name="title" value="AutoZone.com - Get in the Zone!"/>
<put name="main" value="/WEB-INF/views/tiles/home.jsp"/>
<put name="pageName" value="Home"/>
</definition>
```

Applying Spring MVC (and AOP)

- **Home Page Example**
  - More going on behind-the-scenes…
  - Interceptor performs 'customer' logic common to nearly all requests

```
<bean id="urlBeanMapping"
 class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
  <property name="interceptors">
    <list>
          <ref local="ecomInterceptor"/>
    </list>
  </property>
</bean>
```

Approaching Interceptors

- 'Interceptors' get at requests before and/or after your handler does
- Our current 'main' interceptor provides conduit to logic previously contained primarily within 'main' servlet
  - Previous servlet became catch-all for quick-fix issues
    - Conditional logic for 2% of cases being evaluated 100% of the time
  - Logic now layered, accessed through interceptor
- Pre-Request: Determine Customer Type
- Post Request:
  - Determine and add data for commonly used 'view' beans if have not already been defined
    - Top Navigation
    - Hitbox
  - Write cookie (if needed)

Pre-Request

```
private MasterService service;

public boolean preHandle(HttpServletRequest request,HttpServletResponse response,Object handler) {

     //get application context; if not defined, define it
     if (UiBroker.context == null) {
               javax.servlet.ServletContext servletContext =
                              request.getSession().getServletContext();
               UiBroker.context = WebApplicationContextUtils.getRequiredWebApplicationContext(
                              servletContext);
     }

     ApplicationContext ctx = UiBroker.context;

     //get MasterController
     MasterController masterController =
               UiBroker.getMasterController(request.getSession());

               //determine customer type and act appropriately
               masterController =
                         service.processCustomerCredentials(request, response, ctx);

     return true;
}  //slightly altered for demonstration purposes
```

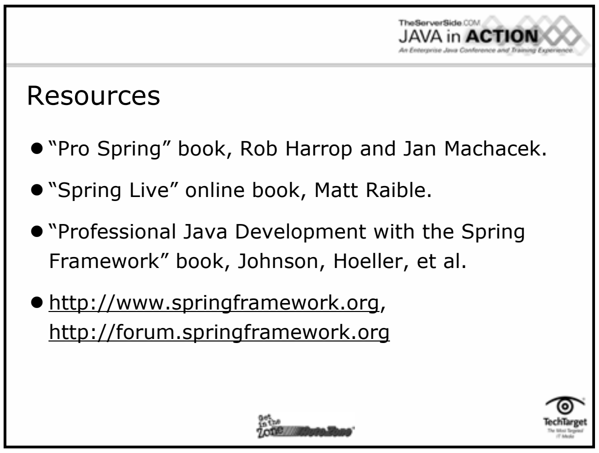
Spring AOP and MVC – Approach Follow-up

- Handler methods should be specific to user request, not be doing housekeeping common to website
- Views should be specific to presentation logic, not contain repetitive logic

  Examples where we leverage Spring AOP:
  - Site Breadcrumbs/Navigation
  - HitBox (customer tracking)

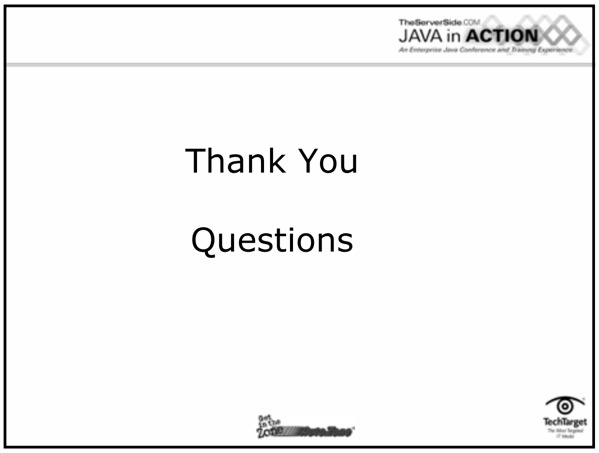
Lessons Learned

- Spring Rocks!
- Spring promotes good coding habits.
- There's seemingly always a better way to do what you just did.
- Mistakes will be made, but they'll be easier to fix when you use the Spring Framework and the design principles it helps developers enforce.



Resources

- “Pro Spring” book, Rob Harrop and Jan Machacek.
- “Spring Live” online book, Matt Raible.
- “Professional Java Development with the Spring Framework” book, Johnson, Hoeller, et al.
- http://www.springframework.org, http://forum.springframework.org



Thank You

Questions