Servlet:

Overview of `FindWeapons`

1. **init()**:
   - Initializes the `WeaponDao` instance for database operations.
2. **doGet() and doPost()**:
   - Process user input from HTTP GET and POST requests.
   - Fetch weapons based on the input name parameter.
   - Sort the fetched weapons based on various attributes if a sort key is provided.
   - Forward the data to "FindWeapons.jsp" for display.
3. **extractNumberPart(String name)**:
   - Extracts and pads numeric parts from weapon names to support alphanumeric sorting.
4. **WEAPON_NAME_COMPARATOR**:
   - A comparator for sorting weapons by name, considering numeric values in names for a natural sort order.

Conclusion

The `FindWeapons` servlet is designed to manage and display weapon data based on user queries. It supports filtering by name and sorting by multiple attributes, effectively utilizing database interactions and Java Servlet technology to facilitate user interaction in a web application.

Overview of `WeaponCreateServlet`

The `WeaponCreateServlet` is designed to handle web requests for creating new weapon entries in the database using a web form. It is part of a Java web application that interacts with a data access layer to perform its operations.

Key Functions

1. **init()**:
   - Initializes the `WeaponDao` instance which is used to interact with the database.
2. **doGet(HttpServletRequest req, HttpServletResponse resp)**:
   - Prepares and forwards the request to the "WeaponCreate.jsp" page. This method mainly handles displaying the form for creating a new weapon.
3. **doPost(HttpServletRequest req, HttpServletResponse resp)**:
   - Handles the submission of the weapon creation form.
   - Retrieves form data from the request, including weapon attributes like name, stack size, price, and various statistics such as item level, damage, and more.
   - Creates a new `Weapon` object and attempts to save it to the database using the `WeaponDao`.
   - Catches and handles SQL exceptions, and throws an `IOException` if an error occurs.

  - Sets a success message if the weapon is created successfully and forwards the request back to the "WeaponCreate.jsp" page with the result.

Conclusion

The `WeaponCreateServlet` primarily facilitates the creation of new weapons in the database through a user interface. It efficiently handles both displaying the form for weapon entry and processing the form data to create new weapon records, integrating web technology with backend database operations.

Overview of `WeaponDeleteServlet`

The `WeaponDeleteServlet` is designed to handle web requests for deleting existing weapon entries in the database. It is part of a Java web application that interfaces with a data access layer to facilitate database operations.

Key Functions

1. **init()**:
   - Initializes the `WeaponDao` instance which is responsible for database interactions related to weapon data.
2. **doGet(HttpServletRequest req, HttpServletResponse resp)**:
   - Sets up initial messages for the user interface and forwards the request to the "WeaponDelete.jsp" page. This method is primarily responsible for displaying the deletion form.
3. **doPost(HttpServletRequest req, HttpServletResponse resp)**:
   - Processes the deletion request submitted via the web form.
   - Retrieves the weapon ID from the request parameters.
   - Validates the provided weapon ID and handles scenarios where the ID might be empty or invalid by setting appropriate messages.
   - Attempts to delete the weapon from the database using the `WeaponDao`.
   - Catches and processes exceptions, including an `IllegalArgumentException` if attempting to delete a restricted weapon, and a `SQLException` for database errors.
   - Updates the user interface with messages about the deletion status (success or failure) and controls the form’s submit functionality based on the operation's outcome.
   - Forwards the request back to "WeaponDelete.jsp" with updated messages and status.

Conclusion

The `WeaponDeleteServlet` efficiently manages the deletion of weapons in the database, providing robust error handling and user feedback. It ensures that operations are secure by handling exceptional cases (like restricted deletions) and communicates effectively with the user through status messages, making it integral to maintaining the integrity and management of weapon data in the application.

Overview of `WeaponDetailServlet`

The `WeaponDetailServlet` is a Java servlet within a web application that is focused on retrieving and displaying detailed information about a specific weapon from a database. It leverages multiple data access objects (DAOs) to gather comprehensive data about the weapon, its bonuses, and the jobs allowed to use it.

Key Components and Functions

1. **Initialization (`init()` method)**:
   - Initializes instances of `WeaponDao`, `WeaponBonusDao`, and `WeaponAllowedJobDao`. These DAOs are responsible for accessing data related to weapons, their bonuses, and the jobs that can use them, respectively.
2. **Handling GET Requests (`doGet()` method)**:
   - **Parameter Retrieval**: Retrieves the `weaponId` from the HTTP request.
   - **Fetching Weapon Data**:
     - Uses `weaponDao` to fetch the main details of the weapon specified by `weaponId`.
     - Checks if the weapon exists; if not, sets a message indicating the weapon does not exist.
   - **Fetching Weapon Bonus**:
     - Retrieves specific bonus attributes for the weapon (like attack power) using `weaponBonusDao`.
   - **Fetching Allowed Jobs**:
     - Gathers information on which jobs can use the weapon through `weaponAllowedJobDao`.
     - Iterates over all jobs (fetched using `JobDao`) to determine if they are allowed to use the specified weapon.
   - **Error Handling**:
     - Catches and logs `SQLExceptions`, converting them to `IOExceptions` for proper servlet error handling.
   - **Setting Attributes and Forwarding**:
     - Sets the retrieved weapon, its bonuses, and allowed job details as request attributes.
     - Forwards the request to "WeaponDetail.jsp" to display the gathered information.

Conclusion

The `WeaponDetailServlet` serves as a critical component for providing detailed information about weapons within the application. It efficiently aggregates data from multiple sources (weapon details, bonuses, and job permissions) to provide a comprehensive view. The servlet ensures robust error handling and uses effective data management practices to facilitate user interactions with weapon data, enhancing the user experience by providing detailed insights into each weapon's attributes and its compatibility with different jobs in the game.

Overview of `WeaponUpdateServlet`

The `WeaponUpdateServlet` is a Java servlet tasked with handling the update operations for weapon details in a database-driven web application. It interfaces with a data access object (`WeaponDao`) to fetch and update weapon records based on user input.

Key Components and Functions

1. **Initialization (`init()` method)**:
    - Initializes the `WeaponDao` instance, which provides methods to retrieve and update weapon details in the database.
2. **Handling GET Requests (`doGet()` method)**:
    - **Parameter Validation**: Extracts the `weaponId` from the request and checks if it's valid.
    - **Data Retrieval**:
        - If `weaponId` is valid, uses `weaponDao` to fetch the weapon details.
        - Sets a message if the weapon does not exist.
    - **Error Handling**: Catches SQL exceptions, logs them, and throws an `IOException`.
    - **Response Preparation**:
        - Sets weapon details and messages as request attributes.
        - Forwards the request to "WeaponUpdate.jsp" to allow the user to update weapon details.
3. **Handling POST Requests (`doPost()` method)**:
    - **Parameter Retrieval**: Extracts `weaponId` and `newRequiredLevel` from the request.
    - **Validation and Processing**:
        - Validates `weaponId` and ensures the new required level is provided.
        - Fetches the current weapon details; if the weapon exists, updates its required level based on user input using `weaponDao`.
        - Sets success or error messages based on the operation's outcome.
    - **Error Handling**: Similar to `doGet`, it manages SQL exceptions.
    - **Response Preparation**:
        - Sets updated weapon details and messages as request attributes.
        - Forwards the request back to "WeaponUpdate.jsp" with the results of the update operation.

Conclusion

The `WeaponUpdateServlet` plays a vital role in the management of weapon data within the application, allowing users to update specific attributes of a weapon, such as its required level. It ensures robust validation and error handling to manage database operations effectively and provides a user-friendly interface for updating weapon details. This servlet enhances the functionality and interactivity of the web application by facilitating the dynamic updating of database records.

XML File

The provided XML configuration snippet is from a `web.xml` file, used to configure servlet behavior in a Java web application that follows the Java Servlet specification. This configuration is specific to the deployment of a servlet-based application managed by a servlet container (like Apache Tomcat).

Breakdown of the XML Elements

1. **Root Element `<web-app>`**:
    - **Attributes**:
        - `xmlns`: Declares the XML namespace for a JavaEE servlet, which is "http://xmlns.jcp.org/xml/ns/javaee" for JavaEE 7 and later.
        - `xsi:schemaLocation`: Specifies the XML Schema instance namespace and the schema document location that validates the XML document. For JavaEE version 4.0, the schema is located at "http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd".
        - `version`: Specifies the version of the Servlet specification the application uses, in this case, "4.0".
        - `id`: Provides an identifier for the web application, here "WebApp_ID".
2. **Element `<display-name>`**:
    - Contains a human-readable name of the web application, "CS5200Project", which is generally used for administrative purposes.
3. **Element `<welcome-file-list>`**:
    - Specifies a list of files that the web server can serve as the default page when the application is accessed without specifying a particular resource.
    - `<welcome-file>`: Specifies "FindWeapons.jsp" as the default document for the application. This means when the application URL is accessed without any specific path, the "FindWeapons.jsp" file is served.
4. **Commented Out Servlet Configuration**:
    - The XML also contains commented-out sections that define and map a servlet named `FindWeapons`.
        - `<servlet>`: Defines a servlet named "FindWeapons" and specifies its implementing class as `game.servlet.FindWeapons`.
        - `<servlet-mapping>`: Maps the servlet named "FindWeapons" to the URL pattern `/findweapons`. This means any requests to `http://yourserver/findweapons` will be handled by the `FindWeapons` servlet.

Conclusion

This configuration file is central to setting up the basic behaviors of a web application, including defining which JSP or HTML file to show by default when the application is accessed and potentially mapping servlets to URL patterns (though the latter is commented out in this snippet). The XML structure adheres to the Java Servlet specification and ensures that the application conforms to the expected standard of behavior for servlets and web applications in a JavaEE environment.

JSP File

FindWeapons.JSP

The JSP file you provided is designed for managing weapon details in a web application. It allows users to:

1. **Search for Weapons**: Users can enter a weapon name and choose a sorting criterion through a form, which posts the data to be processed.
2. **Display Weapon Information**: It lists weapons with details like name, item level, and damage types, each with links for more detailed views, updates, and deletion options.
3. **Create, Update, and Delete Operations**: Provides links to create new weapons or update and delete existing ones.

This page effectively combines form handling, dynamic data display, and navigation within the application, making it central to weapon management functionalities.

WeaponCreate.JSP

The provided JSP file is structured to facilitate the creation of a new weapon in a web application. It features a form where users can input various attributes of a weapon, such as name, max stack size, price, item level, physical damage, magic damage, auto attack rate, delay, and required level. Here are the main functionalities:

1. **Form Submission**: The form is set to post data to the "weaponcreate" action, which presumably handles the insertion of the new weapon data into a database.
2. **Input Fields**: Users fill out multiple fields to define the weapon's characteristics. Each field has an associated label for clarity.
3. **Feedback Message**: After submission, feedback (such as success messages) is displayed to the user, providing confirmation or error information based on the operation's outcome.

This page serves as a straightforward interface for adding new weapons to the game's database, ensuring that all necessary weapon details are collected and processed appropriately.

WeaponDelete.JSP

The JSP file provided is designed to facilitate the deletion of a weapon from a web application's database. Here's how it functions:

Structure and Elements

1. **Title Display**: Uses the `${messages.title}` to display the appropriate message at the top of the page, which could indicate the purpose of the form or the result of a previous action.
2. **Form for Deletion**:

   - **Action**: The form submits data to the "weapondelete" action, which handles the deletion process on the server side.
   - **Conditional Display**:
     - The input field for the "Weapon ID" is shown or hidden based on the condition `${messages.disableSubmit}`. If the condition is true, indicating a situation like a successful deletion or an invalid operation, the input field is hidden.
     - Similarly, the submit button is conditionally displayed using the same variable. This prevents further submissions when it's not appropriate, such as after a successful deletion or if an error has occurred that needs to be addressed first.
3. **Security and Usability**:
   - **Input Sanitization**: Uses `fn:escapeXml(param.weaponId)` to safely encode the weapon ID from the URL parameter to prevent Cross-Site Scripting (XSS) attacks.
   - **User Feedback**: Utilizes conditional JSTL tags to manage the display of form elements, enhancing user experience by dynamically adjusting the interface based on the context of the interaction.

Summary

This JSP page serves as a straightforward and user-focused interface for deleting weapons. It employs conditional logic to adapt the UI based on the operation's context and ensures security through proper handling of user inputs. This approach not only helps in preventing accidental deletions but also maintains a clean and effective user experience during sensitive operations like deletions.

WeaponDetail.JSP

The provided JSP file serves as a detailed view page for individual weapons within a web application. Here's how it functions and is structured:

Key Features and Structure

1. **Header and Metadata**:
   - Sets up basic HTML and page metadata including the title "Weapon Detail".
2. **Display Weapon Attributes**:
   - Presents various attributes of the weapon such as its name, item level, physical damage, and magic damage using the `c:out` tag to safely render text, preventing HTML or script injection.
3. **Conditional Display for Bonus**:
   - Uses `c:choose`, `c:when`, and `c:otherwise` to display the weapon's bonus if it exists. If no bonus is associated with the weapon, it displays "No bonus available".
4. **List of Allowed Jobs**:
   - A list (`<ul>`) of jobs that are permitted to use the weapon, populated through a `c:forEach` loop iterating over `allowedJobs`. Each job's name is displayed in a list item (`<li>`).
5. **Action Links**:

- Provides hyperlinks to delete or update the weapon. These actions are linked with the weapon's ID, which is included in the URL to ensure the correct weapon is targeted.

Summary

This JSP page effectively uses JSTL to display detailed information about a weapon while ensuring the content is securely rendered. The page is user-focused, providing not only detailed attributes of the weapon but also interactive links that allow users to easily delete or update the weapon directly from the page. This makes it a comprehensive tool for managing weapon details within the application.

WeaponList.JSP

The JSP file you provided is designed for displaying a list of weapons based on item level criteria, within a web application. It includes both a form for submitting the search criteria and a table for presenting the results. Here's a breakdown of its main features:

Key Features and Structure

1. **Form for Filtering Weapons**:
    - The form allows users to input an item level and submit it to filter weapons. This is handled through a POST request to the `weaponlist` endpoint.
    - The input field for the item level uses `fn:escapeXml` to ensure any user input is safely encoded, preventing Cross-Site Scripting (XSS) attacks.
2. **Success Message Display**:
    - After submitting the form, any success messages (e.g., results of the search or an error message) are displayed using `${messages.success}`.
3. **Table of Weapon Data**:
    - Displays a list of weapons in a tabular format, including columns for name, item level, physical damage, and magic damage.
    - Each weapon name in the table is a hyperlink that leads to a detailed view (`weapondetail`), allowing users to see more detailed information about each weapon.
    - The table is dynamically populated using a `c:forEach` loop that iterates over the `weapons` collection provided by the server.

Summary

This JSP page serves as an interactive interface for querying and viewing weapons based on their item level. It efficiently combines form handling and dynamic data display, providing users with a straightforward way to access detailed information about each weapon. The inclusion of security measures like `fn:escapeXml` enhances the safety of the application by preventing common web vulnerabilities such as XSS. This page is essential for users interested in exploring weapons based on specific criteria within the application.

WeaponUpdate.JSP

The JSP file you've provided is designed for updating specific attributes of a weapon within a web application. The main functionality revolves around submitting a form that enables users to update the required level of a weapon identified by its ID. Here's a breakdown of its main components:

Key Features and Structure

1. **Header and Setup**:
   - Sets up the page with basic HTML and metadata, including setting the character encoding and specifying the title "Update a Weapon".
2. **Form for Updating Weapon Details**:
   - The form submits data to the `weaponupdate` action using the POST method. This approach helps ensure data security by not exposing sensitive information in the URL.
   - **Weapon ID Input**: Users must enter the ID of the weapon they wish to update. This field utilizes the `fn:escapeXml` function to safely encode the input, preventing Cross-Site Scripting (XSS) attacks.
   - **Required Level Input**: Provides a field for users to enter a new required level for the weapon, allowing updates to this specific attribute.
3. **Submission Button**:
   - A simple submit button allows the form data to be sent to the server for processing.
4. **Success Message Display**:
   - Displays feedback from the server using `${messages.success}`, which could inform the user of the success or failure of the update operation.

Summary

This JSP page provides a focused and secure interface for updating the required level of weapons in the application's database. It combines form handling and user feedback to ensure a smooth and safe user experience. The use of JSTL functions like `fn:escapeXml` for input handling further enhances security by preventing common web vulnerabilities. This page is essential for administrative tasks within the application, allowing for easy adjustments to weapon properties as needed.

Added Method in DAL Class

```
    public List<Weapon> getWeaponsByName(String name) throws SQLException {
        List<Weapon> weapons = new ArrayList<Weapon>();
        String selectWeapons = "SELECT Weapon.itemID, Item.name, 
Item.maxStackSize, Item.price, Item.itemLevel,\n"
                + "Weapon.physicalDamage, Weapon.magicDamage, Weapon.autoAttack, 
Weapon.delay, Weapon.requiredLevel\n"
                + "FROM Weapon\n"
                + "INNER JOIN Item ON Weapon.itemID = Item.itemID\n"
                + "WHERE Item.name LIKE ?;\n"
                + "";
        Connection connection = null;
        PreparedStatement selectStmt = null;
        ResultSet results = null;
        try {
            connection = connectionManager.getConnection();
            selectStmt = connection.prepareStatement(selectWeapons);
            selectStmt.setString(1, "%" + name + "%");
            results = selectStmt.executeQuery();
            while (results.next()) {
                int itemID = results.getInt("itemID");
                String resultName = results.getString("name");
                int maxStackSize = results.getInt("maxStackSize");
                double price = results.getDouble("price");
                int itemLevel = results.getInt("itemLevel");
                int physicalDamage = results.getInt("physicalDamage");
                int magicDamage = results.getInt("magicDamage");
                int autoAttack = results.getInt("autoAttack");
                int delay = results.getInt("delay");
                int requiredLevel = results.getInt("requiredLevel");
                Weapon weapon = new Weapon(itemID, resultName, maxStackSize,
price, itemLevel, physicalDamage, magicDamage, autoAttack, delay, requiredLevel);
                weapons.add(weapon);
            }
        } catch (SQLException e) {
            e.printStackTrace();
            throw e;
        } finally {
            if (connection != null) {
                connection.close();
            }
            if (selectStmt != null) {
                selectStmt.close();
            }
            if (results != null) {
                results.close();
            }
        }
        return weapons;
    }
```

The `getWeaponsByName` method in the Java code is designed to retrieve a list of weapons from a database where the weapon names match a specified search term. This method constructs a SQL query that performs an inner join between `Weapon` and `Item` tables, searches for weapons whose names contain the provided string, and returns a list of `Weapon` objects with complete details extracted from the database.