DOWNLOAD PDF JSP AND A JDBC CONNECTION POOL BEAN

Chapter 1 : java - Setting up Tomcat JDBC connection pool with the Spring JDBCTemplate - Stack Overflo

Re: connection pool @ JSP Bean Feb 25, AM (in response to) Tomcat comes with a facility to pool datasources, you might want to look at that as it doesn't require you to initialize the pool in your webapp.

Tweet Connection pooling is a mechanism to create and maintain a collection of JDBC connection objects. The primary objective of maintaining the pool of connection object is to leverage re-usability. A new connection object is created only when there are no connection objects available to reuse. This technique can improve overall performance of the application. This article will try to show how this pooling mechanism can be applied to a Java application. Establishing a database connection is a very resource-intensive process and involves a lot of overhead. Moreover, in a multi-threaded environment, opening and closing a connection can worsen the situation greatly. To get a glimpse of what actually may happen with each request for creating new database connection, consider the following points. Database connections are established using either DriverManager of DataSource objects. An application invokes the getConnection method. JVM has to ensure that the call does not violate security aspects as the case may be with applets. The invocation may have to percolate through a firewall before getting into the network cloud. On reaching the host, the server processes the connection request. The database server initializes the connection object and returns back to the JDBC client again, going through the same process. And, finally, we get a connection object. This is just an overview of what actually goes on behind the scenes. Rest assured, the actual process is more complicated and elaborate than this. In a single-threaded controlled environment, database transactions are mostly linear, like opening a connection, doing database transaction, and closing the connection when done. Real-life applications are more complex; the mechanism of connection pooling can add to the performance although there are many other properties that are critical to overall performance of the application. The complexity of the concept of connection pooling gets nastier as we dive deep into it. But, thanks go to the people who work to produce libraries specifically for the cause of connection pooling. What Actually Happens with Connection Pooling?

Chapter 2 : java - Approach to JDBC connections in a Servlet/JSP application - Stack Overflow

Or course each JSP requires different Resultsets, so some cases 2 or 3 resultsets, so I am sending each query string to the bean which returns the resultset back ok, but now i have to close the connection(ie return the connection in the bean back to the pool).

Start coding something amazing with our library of open source Cloud code patterns. Content provided by IBM. It is further assumed that you have replaced the default GlassFish server instance in NetBeans with a new instance with its own folder for the domain. This is required for all platforms. As I went through the steps I realized that I needed to record the steps for my students to reference. Here then are these steps. Steps 1 through 4 are required, Step 5 is optional. I recommend downloading the Platform Independent version. You are looking for the driver file named mysql-connector-java Copy the driver file to the lib folder in the directory where you placed your domain. On my system the folder is located at C: If GlassFish is already running then you will have to restart it so that it picks up the new library. On my Windows 8. Click on Properties and the Servers dialog will appear. By default it is checked. NetBeans determines the driver to copy to GlassFish from the file glassfish-resources. Without this file and if you have not copied the driver into GlassFish manually then GlassFish will not be able to connect to the database. Any code in your web application will not work and all you will likely see are blank pages. Step 1a or Step 1b? I recommend Step 1a and manually add the driver. The reason I prefer this approach is that I can be certain that the most recent driver is in use. As of this writing NetBeans contains version 5. If you copy a driver into the lib folder then NetBeans will not replace it with an older driver even if the check box on the Server dialog is checked. NetBeans does not replace a driver if one is already in place. If you need a driver that NetBeans does have a copy of then Step 1b is your only choice. All that is required is that you create a connection to the database. I assume that the database you wish to connect to already exists. Go to the Services tab and right mouse click on New Connection. In the next dialog you must choose the database driver you wish to use. It defaults to Java DB Embedded. Pull down the combobox labeled Driver: Click on Next and you will now see the Customize Connection dialog. Here you can enter the details of the connection. On my system the server is localhost and the database name is Aquarium. Here is what my dialog looks like. Notice the Test Connection button. I have clicked on mine and so I have the message Connection Succeeded. There is nothing to do on this dialog so click on Next. On this last dialog you have the option of assigning a name to the connection. By default it uses the URL but I prefer a more meaningful name. Click on Finish and the connection will appear under Databases. Verify that the database is running and is accessible. If it is then delete the connection and start over. Having a connection to the database in NetBeans is invaluable. You can interact with the database directly and issue SQL commands. In this dialog you can give the project a name and a location in your file system. The final dialog lets you select the application server that your application will use and the version of Java EE that your code must be compliant with. Here is my project ready for the next step. Right mouse click on the project name and select New and then Other â€¦ Scroll down the Categories list and select GlassFish. The next dialog is the General Attributes. Do not prefix the name with java: An upcoming article will explain why. There is nothing for us to enter on the Properties dialog. On the Choose Database Connection dialog we will give our connection pool a name and select the database connection we created in Step 2. Notice that in the list of available connections you are shown the connection URL and not the name you assigned to it back in Step 2. We do need to make one change. The resource type shows javax. DataSource and we must change it to javax. A new folder has appeared in the Projects view named Other Sources. It contains a sub folder named setup. In this folder is the file glassfish-resources. I have reformatted the file for easier viewing. Configure GlassFish with glassfish-resources. When the application is un-deployed the resource and pool are removed. If you want to set up the resource and pool permanently in GlassFish then follow these steps. Go to the Services tab and select Servers and then right mouse click on GlassFish. If GlassFish is not running then click on Start. With the server started click on View Domain Admin Console. Your web browser will now open and show you the GlassFish console. If you assigned a user name and password to the server you will have to enter this

information before you see the console. In the Common Tasks tree select Resources. You should now see in the panel adjacent to the tree the following: Click on Add Resources. You should now see: In the Location click on Choose File and locate your glassfish-resources. Mine is found at D: If everything has gone well you should see: The final task in this step is to test if the connection works. Go to Step 1a and manually add the driver. The resources are now visible in NetBeans. Having the resource and pool add to GlassFish permanently will allow other applications to share this same resource and pool. You are now ready to code! Code something amazing with the IBM library of open source blockchain patterns. Read More From DZone.

Chapter 3 : JSP, Java Beans, Database and Jigsaw

The Tomcat JDBC Connection Pool blog.quintoapp.com is described as a replacement or alternative to the older commons-dbcp connection pool. It can be used either through a JDNI resource or standalone as instantiated bean outside of the container. Tomcat JDBC (blog.quintoapp.com) is part of the.

Why Use Enterprise Jav aBeans? EJBs have many uses in business applications, including the use of session beans for server-side business logic and entity beans to manage data persistence. EJB technology provides a more robust infrastructure than JSP or servlet technology, for use in secure, transactional, server-side processing. A typical application design often uses a servlet as a front-end controller to process HTTP requests, with EJBs being called to access or update a database, and finally another servlet or JSP page being used to display data for the requester. There are three categories of EJBs: Container Managed Persistence entity beans, in particular, are well-suited to manage persistent data, because they make it unnecessary to use the JDBC API directly when accessing a database. Instead, you can let the EJB container handle database operations for you transparently. Session beans are useful to model business logic and may be either stateless or stateful, with stateful beans typically being used where transaction state must be maintained across method calls or servlet requests. Stateless beans contain individual business logic methods that are independent of application state. Within the entity bean implementation, a basic persistence manager supports both simple mapping and complex mapping, supporting one-to-one, one-to-many, many-to-one, and many-to-many object-relational mappings. It also automatically maps fields of an entity bean to a corresponding database table. To facilitate application maintenance and deployment, Oracle Application Server provides a number of enhancements, including dynamic EJB stub generation. The s ervlet calls an EJB that is co-located, meaning it is in the same application and on the same host, running in the same JVM. For this, use EJB local interfaces. Remote lookup within the same application: The servlet calls an EJB that is in the same application, but on a different host, where the application is deployed to both hosts. This requires EJB remote interfaces. This would be the case for a multitier application where the servlet and EJB are in the same application, but on different tiers. Remote lookup outside the application: The servlet calls an EJB that is not in the same application. In versions of EJB before 3. They are then used to create EJBs for use by the application. J2EE components can use the default no-args constructor to look up objects within the same application. This setup is typically in the servlet code, but for a lookup in the same application it can be in the rmi. Remote lookup within the same application on different hosts also requires proper configuration of the OC4J EJB remote flag for your application, on each host. EJBObject interface, and a home interface extending the javax. In this model, all EJBs are defined as remote objects, adding unnecessary overhead to EJB calls in situations where the servlet or other calling module is co-located with the EJB. In this case, the EJB has a local interface that extends the javax. EJBLocalObject interface, in contrast to having a remote interface. In addition, a local home interface that extends the javax. EJBLocalHome interface is specified, in contrast to having a home interface. RMI and other overhead are eliminated when you use local interfaces. An EJB can have both local and remote interfaces. The term local lookup in this document refers to a co-located lookup, in the same JVM. Do not confuse "local lookup" with "local interfaces". Although local interfaces are typically used in any local lookup, there may be situations in which remote interfaces are used instead. When this flag is set to "true" on a server, beans will be looked up on a remote server instead of the EJB service being used on the local server. Update the file to set this flag to "true", as follows: You can deploy the application EAR file to both servers with a remote flag value of "false", then set it to "true" on server 1, the servlet tier. This is illustrated in Figure Use the default administrative user name for the remote host, and the administrative password set up on the remote host. This avoids possible JAZN configuration issues.

Chapter 4 : How to setup JNDI Database Connection pool in Tomcat - Spring Tutorial Example

I develop a website with JSP pages and some beans under Apache+Tomcat. Tomcat is configured to destroy my bean instances every 2 hours. But, if my objects are killed, the JBDC connection stays alive indefinitely.

This is by convention but is not required. You can specify any desired prefix in your taglib directive. See "Tag Syntax Symbology and Notes" for general information about tag syntax conventions in this manual. Do this by specifying a data source location, in which case connection caches are supported, or by specifying the user, password, and URL individually. The implementation uses oracle. See "ConnBean for a Database Connection" for more information. The ConnBean object for the connection is created in an instance of the tag-extra-info class of the dbOpen tag. You must set either the dataSource attribute or the user, password, and URL attributes. Optionally, you can use a data source to specify a URL, then use the dbOpen tag user and password attributes separately. When a data source is used, and is for a cache of connections, the first use of the cache initializes it. If you specify the user and password through the dbOpen tag user and password attributes, that will initialize the cache for that user and password. Subsequent uses of the cache are for the same user and password. Optionally use this to specify an ID name for the connection. You can then reference this ID in subsequent tags such as dbQuery or dbExecute. Alternatively, you can nest dbQuery and dbExecute tags inside the dbOpen tag. You can also reference the connection ID in a dbClose tag when you want to close the connection. In this case, the connection will be found through the connection ID. With the scope attribute, it is possible to have multiple connections using the same connection ID but different scopes. If you specify a connection ID, then the connection is not closed until you close it explicitly with a dbClose tag. Without a connection ID, the connection is closed automatically when the dbOpen end-tag is encountered. Use this to specify the desired scope of the connection instance. The default is page scope. If you specify a scope setting in a dbOpen tag, then you must specify the same scope setting in any other tagâ€”dbQuery, dbExecute, or dbCloseâ€”that uses the same connection ID. Optionally use this to specify the JNDI name of a data source for database connections. First set up the data source in the OC4J data-sources. You can use the dbOpen tag user and password attributes instead. This attribute is supported only in OC4J environments. This is the user name for a database connection. If a user name is specified through both a data source and the user attribute, the user attribute takes precedence. It is advisable to avoid such duplication, because conflicts could arise if the data source is a pooled connection with existing logical connections using a different user name. This is the user password for a database connection. Note that you do not have to hardcode a password into the JSP page, which would be an obvious security concern. Instead, you can get the password and other parameters from the request object, as follows: URL required if no data source is specified: This is the URL for a database connection. Set this to "true" for an automatic SQL commit when the connection is closed or goes out of scope. The default "false" setting results in an automatic SQL rollback. As a convenience, if you want to specify application-wide automatic commit or rollback behavior, set the parameter name commit-on-close in the application web. In previous releases, the behavior is always to commit automatically when the connection is closed. The commitOnClose attribute offers backward compatibility to simplify migration. If connId is not used in the dbOpen tag, then the connection is closed automatically when the dbOpen end-tag is reached; a dbClose tag is not required. This is the ID for the connection being closed, specified in the dbOpen tag that opened the connection. This is the scope of the connection instance. The default is "page", but if the dbOpen tag specified a scope other than page, you must specify that same scope in the dbClose tag. This tag uses an oracle. CursorBean object for the cursor, so you can set properties such as the result set type, result set concurrency, batch size, and prefetch size, if desired. Also see "Example Using the transform and dbQuery Tags". This currently results in a syntax error. The dbQuery tag does not currently support LOB columns. You can use this to specify an ID name for the cursor. This is required if you want to process the results using a dbNextRow tag. If the queryId parameter is present, then the cursor is not closed until you close it explicitly with a dbCloseQuery tag. Without a query ID, the cursor is closed automatically when the dbQuery end-tag is encountered. This is not a request-time attribute, meaning it cannot take a JSP expression value. This is the ID

for a database connection, according to the connId setting in the dbOpen tag that opened the connection. If you do not specify connId in a dbQuery tag, then the tag must be nested within the body of a dbOpen tag and will use the connection opened in the dbOpen tag. This is not a request-time attribute. The default is "page", but if the associated dbOpen tag specified a scope other than page, you must specify that same scope in the dbQuery tag. This is the desired output format, one of the following. This is the maximum number of rows of data to display. The default is all rows. This is the number of data rows to skip in the query results before displaying results. The default is 0. Use this to bind a parameter into the query. The following example is from an application that prompts the user to enter an employee number, using bindParams to bind the specified value into the empno field of the query: To use this, you must also set output to "XML". If queryId is not specified in the dbQuery tag, then the cursor is closed automatically when the dbQuery end-tag is reached; a dbCloseQuery tag is not required. The ID for the cursor to be closed, specified in the dbQuery tag that opened the cursor. Place the processing code in the tag body, between the dbNextRow start-tag and end-tag. The body is executed for each row of the result set. The result set object is created in an instance of the tag-extra-info class of the dbQuery tag. This is the ID of the cursor containing the results to be processed, specified in the dbQuery tag that opened the cursor. Place the statement in the tag body, between the dbExecute start-tag and end-tag. CursorBean object for the cursor. The dbExecute tag does not currently support LOB columns. This is the ID of a database connection, according to the connId setting in the dbOpen tag that opened the connection. If you do not specify connId in a dbExecute tag, then the tag must be nested within the body of a dbOpen tag and will use the connection opened in the dbOpen tag. The default is "page", but if the dbOpen tag specified a scope other than page, you must specify that same scope in the dbExecute tag. For DDL statements, the statement execution status will be printed. The default is "no". Use this to bind a parameter into the SQL statement. The following example is from an application that prompts the user to enter an employee number, using bindParams to bind the specified value into the empno field of the DELETE statement: For applications using the data-access tags, consider using the dbSetParam tag to supply only parameter values rather than textual completion of the SQL statement itself. This avoids the possibility of what is referred to as "SQL poisoning", where users might enter more SQL code in addition to the expected value. This is the name of the parameter to set. This is the desired value of the parameter. This is the scope of the bind parameter. Example The following example uses a dbSetParam tag to set the value of a parameter named id2. This value is then bound into the SQL statement in the dbExecute tag. The dbSetCookie tag wraps functionality of the standard javax. This is the name of the cookie. This is the desired value of the cookie. Because it is permissible to have a null-value cookie, this attribute is not required. This is the domain name for the cookie. The form of the domain name is according to the RFC specification. This is for a comment describing the purpose of the cookie. This is the maximum allowable age of the cookie, in seconds. Use a setting of "-1" for the cookie to persist until the browser is shut down. This is the version of the HTTP protocol that the cookie complies with.

Chapter 5 : java - Connection pooling in spring jdbc - Stack Overflow

When you say 'some class which runs in container', does a bean called from a jsp The following is a sample blog.quintoapp.com that adds a Oracle jdbc connection pool to.

You must show a list of products to user filtered by criteria from database. The client selects some of these products and updates the minimum stock to get an alert and restock them. Based on these cases, we can learn lot of things: The connection should live only in the block where it is used. It should not live before or after that. Both cases can happen at the same time since they are in a multi threaded environment. So, a single database connection must not be available to be used by two threads at the same time, in order to avoid result problems. From last sentence, each database operation or group of similar operations like Case 2 should be handled in an atomic operation. To assure this, the connection must not be stored in a singleton object, instead it must be live only in the method being used. It will simply fail. Instead, declare only the Connection as field of your DBConnection class. Since you must close the connection after its usage, then add two more methods: These methods will handle the database connection retrieval and closing that connection. Additional, since the DBConnection looks like a wrapper class for Connection class and database connection operations, I would recommend to have at least three more methods: These methods will be plain wrappers for Connection setAutoCommit Connection close and Connection rollback respectively. Then you can use the class in this way: When working in an application server, you should not open connections naively e. You can roll on some database connection pooling libraries like C3P0 as explained here: How to establish a connection pool in JDBC? Or configure one in your application server, as I explain here: Is it a good idea to put jdbc connection code in servlet class? If this is for learning purposes, then roll on your own classes to handle the communication with your database. Instead, use a database connectivity framework like ORMs e. The choice is yours.

Chapter 6 : Using JDBC or Enterprise JavaBeans

Every request creates a new JDBC connection and instantiates the required Dao objects. Obviously expensive both from the connection creation and the Dao instantiation Every request fetches a connection from a JNDI datasource and instantiates the required Dao objects.

Atomist automates your software deliver experience. In software engineering, a connection pool is a cache of database connections maintained so that the connections can be reused when future requests to the database are required. Connection pools are used to enhance the performance of executing commands on a database. Opening and maintaining a database connection for each user, especially requests made to a dynamic database-driven website application, is costly and wastes resources. In connection pooling, after a connection is created, it is placed in the pool and it is used over again so that a new connection does not have to be established. If all the connections are being used, a new connection is made and is added to the pool. Connection pooling also cuts down on the amount of time a user must wait to establish a connection to the database. Wikipedia In this tutorial, we shall create a connection pool to a MySQL database in Glassfish web server, then create a simple web application that makes use of the connection pool. Then we navigate to the admin console. Select the Resource Type as javax. Note that you need to change the value for URL and not Url case sensitive. I set mine as jdbc: We now test whether this connection is working. Click on Tutorial and it should open such an interface: Once succeeded, scroll to the bottom of the page and click to select the check-box and enable Non Transactional Connections: That is it for that section. For the Pool Name, select Tutorial. Leave the rest as default and click OK You should now see your resource as having been created. We now create our web application that makes use of the connection pool. Click Next and give the project the name tutorial. Choose the server as Glassfish Server 3. The project will be created and the start page, index. It has 8 active connections, which is the minimum that Glassfish offered when we were creating the connection pool: Get the open source Atomist Software Delivery Machine and start automating your delivery right there on your own laptop, today! Read More From DZone.

Chapter 7 : Looking For Connection Bean For Connection Pool (JSP forum at Coderanch)

The jsp pages could then simply the bean. Tomcat uses Tyrex as a JDBC Connection Pool. Setting up DataSources in Tomcat is acheived by: 1. Adding.

Chapter 8 : Sample Applications

To configure the pool in a stand alone project using bean instantiation, the bean to instantiate is blog.quintoapp.comurce. The same attributes (documented below) as you use to configure a connection pool as a JNDI resource, are used to configure a data source as a bean.

Chapter 9 : java - Singleton in JSP, how to properly tidy up on close? - Stack Overflow

I am just curious to know how connection pooling in handled in spring jdbc? If spring is taking care of connections, then where can I specify the max number of connections allowed for my application? Another question is how is connection pooling handled in simple jdbc.