

Module 9 sample

Pool and DBCP: creating and using object pools

9.1 The Pool component	1
9.2 The DBCP component	18
9.3 Summary	37
Index	39

Pool and DBCP are the Jakarta Commons components that deal with creating, managing, and using object pools. DBCP (which stands for Database Connection Pool and uses the Pool component to manage database connections) represents a successful implementation of the Pool component because it builds on it to supply a connection-pooling mechanism for managing database connectivity. The generalized nature of the Pool component is brought into narrow focus by DBCP. It's therefore natural for us to discuss these two components together.

The Pool component represents a generic API for implementing pools of objects. These objects can represent any Java class, and the Pool API provides the infrastructure to maintain these objects in a pool for easy access, easy return, and lifecycle maintenance.

The DBCP component also represents a generic API for maintaining connections to a database. As we said, it uses the Pool component to maintain these connections, but it also provides support services for maintaining interactions between a database and the application that is accessing it. It does so in a seamless fashion without needing to know the underlying database structure.

In this module, we'll start with the Pool component. We'll discuss object pooling and look at the structure of the Pool API, and we'll use it to create a basic employee pooling mechanism. We'll then move on to explore DBCP and build some concrete examples.

9.1 The Pool component

The Pool component contains a set of APIs that let you manage objects in a pool for reuse and maintainability. The idea of using a pool is to provide reusable objects. A pooling mechanism should not make any assumptions about the objects it contains and should provide facilities for easy creation, management, reuse, and destruction of those objects. The Pool component of Jakarta Commons is such a mechanism; it goes a step further by providing a means to create a factory for creation of the pool itself. But we're getting ahead of ourselves. Let's first look at the need for object pooling and consider some of the issues involved. We'll then examine the structure of the Pool API and discuss in detail the facilities provided by the Pool component.

9.1.1 The case for object pooling

Object pools provide a better way to manage available resources. You've probably come across object pools before in one form or another, most notably as connection pools for managing access to databases. However,

most developers use other object pools without realizing they're doing so: Application servers, servlet engines, mail servers, even database servers, all use some sort of thread-pooling mechanism to service the requests they receive. *Thread pooling* is a way by which most high-end servers manage to respond to requests for services quickly. These servers don't create a thread for each request that comes in; instead, they maintain a pool of ready threads that can be rapidly assigned to a request. The thread does its business and then is returned to the pool for future requests.

Object pooling relies on three development requirements as strong cases for its use:

- *Responsiveness*—Because objects are already created and in a pool, there's no overhead associated with object creation. Your application can quickly dip into the pool, pick an object, and assign it to the task at hand. Contrast this with the tasks needed if object pooling isn't used: Objects must be created and initialized, and only then can they be assigned to the task at hand. Complex objects are the worst offenders and will cause your application to suffer from poor response times. As with most things, there is a tradeoff in this scenario. Heavy objects are a drain on system resources, and maintaining them in system memory waiting for tasks can be problematic. You need to maintain a fine balance between the minimum objects required in the pool and your system's responsiveness.
- *Reusability*—Objects in a pool are reused. Although this may seem similar to the first feature, there is difference because of the way object pools are managed. Consider the case of an object pool where each time you took out an object, a new object was created to replace it (behind the scenes, by a separate thread). The object you took out wasn't returned to the pool but was discarded. Although this process would serve the first purpose of a responsive object pool, it would create overhead for the system. To avoid the overhead of object creation and subsequent garbage collection for the discarded objects, objects pools implement a mechanism by which the object borrowed from the pool can be returned. This returned object can then be used by other tasks.
- *Controlled access*—Objects in a pool are controlled because they're accessed and maintained within the pool. While the pool is sitting idle, it may perform maintenance operations like repairing broken objects, releasing unusable objects, and so on. Access to these objects is limited and controlled via the pool. A request for an object from the pool is handled internally using any protocol, without the application knowing anything about it. Further, by restricting access, you limit the number of these objects in the system at any time, thus improving performance and security. An object pool is an example of the Factory Pattern for creation of objects.

An object pool may not be the best way to manage your objects, however. Object pools are best used in cases where the objects that are being pooled are complex; have large initialization, creation, and destruction times; are used frequently for short small tasks; and can be recycled. Examples of some generic objects that should be pooled include database connections, socket connections, and threads in application servers. In the next section, we'll examine the difference between using an object pool to manage objects and letting the garbage collector in your JVM do its object-collection trick.

Object pooling vs. the garbage collector

The alternate to using an object pool is to let the garbage collector in Java do its business. You use the new keyword to create your objects, and the garbage collector cleans them up when they aren't needed. The humble garbage collector has seen many performance advances since its inception, to the stage that it's now a rival to object pools. A fast and efficient garbage collector can quickly reclaim objects that are no longer needed by your application, thus freeing up memory and making that memory available to other objects. This