<https://github.com/xetorthio/jedis/wiki/Getting-started>

1. Getting started

Jungtaek Lim edited this page 26 days ago · [18 revisions](https://github.com/xetorthio/jedis/wiki/Getting-started/_history)

1. **Getting started**
2. **Installing Jedis**

In order to have Jedis as a dependency in your application you can:

1. **Use the jar files**

download the [latest jedis jar at search.maven.org](http://search.maven.org/#artifactdetails%7Credis.clients%7Cjedis%7C2.4.2%7Cjar) and the [Apache Commons Pool 2 2.0 dependency](http://search.maven.org/#artifactdetails%7Corg.apache.commons%7Ccommons-pool2%7C2.0%7Cjar).

1. **build from source**

This gives you the most recent version.

1. **Clone the github project.**

That is very easy, on the commandline you just need to: git clone git://github.com/xetorthio/jedis.git

1. **build**

Before you package it using maven, you have to pass the tests. To run the tests and package, run make package.

1. **Configure a Maven dependency**

Jedis is also distributed as a Maven Dependency through Sonatype. To configure that just add the following XML snippet to your pom.xml file.

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.6.0</version>

<type>jar</type>

<scope>compile</scope>

</dependency>

1. **Basic usage example**
2. **using Jedis in a multithreaded environment**

You shouldn't use the same instance from different threads because you'll have strange errors. And sometimes creating lots of Jedis instances is not good enough because it means lots of sockets and connections, which leads to strange errors as well. A single Jedis instance is not threadsafe! To avoid these problems, you should use JedisPool, which is a threadsafe pool of network connections. You can use the pool to reliably create several Jedis instances, given you return the Jedis instance to the pool when done. This way you can overcome those strange errors and achieve great performance.

To use it, init a pool:

JedisPool pool **=** **new** **JedisPool(new** **JedisPoolConfig(),** "localhost"**);**

You can store the pool somewhere statically, it is thread-safe.

JedisPoolConfig includes a number of helpful Redis-specific connection pooling defaults. For example, Jedis with JedisPoolConfig will close a connection after 300 seconds if it has not been returned.

You use it by:

*/// Jedis implements Closable. Hence, the jedis instance will be auto-closed after the last statement.*

**try** **(**Jedis jedis **=** pool**.**getResource**())** **{**

*/// ... do stuff here ... for example*

jedis**.**set**(**"foo"**,** "bar"**);**

String foobar **=** jedis**.**get**(**"foo"**);**

jedis**.**zadd**(**"sose"**,** 0**,** "car"**);** jedis**.**zadd**(**"sose"**,** 0**,** "bike"**);**

Set**<**String**>** sose **=** jedis**.**zrange**(**"sose"**,** 0**,** **-**1**);**

**}**

*/// ... when closing your application:*

pool**.**destroy**();**

1. **Setting up master/slave distribution**
2. **enable replication**

Redis is primarily built for master/slave distribution. This means that write requests have to be explicitly addressed to the master (a redis server), which replicates changes to slaves (which are also redis servers). Read requests then can be (but must not necessarily) addressed to the slaves, which alleviates the master.

You use the master as shown above. In order to enable replication, there are two ways to tell a slave it will be "slaveOf" a given master:

* Specify it in the respective section in the Redis Config file of the redis server
* on a given jedis instance (see above), call the slaveOf method and pass IP (or "localhost") and port as argument:

jedis**.**slaveOf**(**"localhost"**,** 6379**);** *// if the master is on the same PC which runs your code*

jedis**.**slaveOf**(**"192.168.1.35"**,** 6379**);**

Note: since Redis 2.6 slaves are read only by default, so write requests to them will result in an error.

If you change that setting they will behave like normal redis servers and accept write requests without errors, but the changes won't be replicated, and hence those changes are at risk to be silently overwritten, if you mix up your jedis instances.

1. **disable replication / upon failing master, promote a slave**

In case your master goes down, you may want to promote a slave to be the new master. You should first (try to) disable replication of the offline master first, then, in case you have several slaves, enable replication of the remaining slaves to the new master:

slave1jedis**.**slaveofNoOne**();**

slave2jedis**.**slaveOf**(**"192.168.1.36"**,** 6379**);**