Home Intro Docs Github

Getting started

MapDB has very power-full API, but for 99% cases you need just two classes: DBMaker is builder style factory for configuring and opening a database. It has handful of static 'newXXX' methods for particular storage mode. DB represents storage. It has methods for accessing Maps and other collections. It also controls DB life-cycle with commit, rollback and close methods.

Best place to checkout various features of MapDB are Examples. There is also screencast which describes most aspects of MapDB.

There is `MapDB Cheat Sheet`, on just two pages it is quick reminder of MapDB capabilities.

Maven

MapDB is in Maven Central. Just add code bellow to your pom file to use it. You may also download jar file directly from repo.

```
<dependency>
    <groupId>org.mapdb</groupId>
    <artifactId>mapdb</artifactId>
    <version>1.0.7</version>
</dependency>
```

ChangeFileRegex

We are working on new generation of MapDB. It is faster and more reliable. Latest semi-stable build is at snapshot repository:

ChangeFileRegex

```
<repositories>
    <repository>
        <id>sonatype-snapshots</id>
        <url>https://oss.sonatype.org/content/repositories/snapshots</url>
    </repository>
</repositories>

<dependencies>
    <dependency>
        <groupId>org.mapdb</groupId>
        <artifactId>mapdb</artifactId>
        <version>2.0.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

ChangeFileRegex

Hello World

Hereafter is a simple example. It opens TreeMap backed by file in temp directory, file is discarded after JVM exit:

```
import org.mapdb.*;
ConcurrentNavigableMap treeMap = DBMaker.newTempTreeMap()

// and now use disk based Map as any other Map
treeMap.put(111,"some value")
```

RootFile

More advanced example with configuration and write-ahead-log transaction.

```
import org.mapdb.*;

// configure and open database using builder pattern.
// all options are available with code auto-completion.
DB db = DBMaker.newFileDB(new File("testdb"))
           .closeOnJvmShutdown()
```

```
            .encryptionEnable("password")
            .make();

// open existing an collection (or create new)
ConcurrentNavigableMap<Integer,String> map = db.getTreeMap("collectionName");

map.put(1, "one");
map.put(2, "two");
// map.keySet() is now [1,2]

db.commit();  //persist changes into disk

map.put(3, "three");
// map.keySet() is now [1,2,3]
db.rollback(); //revert recent changes
// map.keySet() is now [1,2]

db.close();
```

RootFile

What you should know

MapDB is very simple to use, however it bites when used wrong way. Here is list of most common usage errors and things to avoid:

- Transactions (write-ahead-log) can be disabled with DBMaker.transactionDisable(), this will speedup writes. However without transactions store gets corrupted when not closed correctly.
- Keys and values must be immutable. MapDB may serialize them on background thread, put them into instance cache... Modifying an object after it was stored is a bad idea.
- MapDB relies on memory mapped files. On 32bit JVM you will need DBMaker.randomAccessFileEnable() configuration option to access files larger than 2GB. RAF introduces overhead compared to memory mapped files.
- MapDB does not run defrag on background. You need to call `DB.compact()` from time to time.
- MapDB uses unchecked exceptions. All `IOException` are wrapped into unchecked `IOError`. MapDB has weak error handling and assumes disk failure can not be recovered at runtime. However this does not affects data safety, if you use durable commits.