

# Java Tutorial: Hibernate, MySQL, XML

19:44:45 18.05.2012



Mit Hibernate ist es möglich, Objekte direkt in der Datenbank abzulegen. Dieses Beispiel soll einfach mal ein funktionierendes Beispiel beschreiben ohne dabei zu sehr in die Tiefe zu gehen. Für die Tiefe gibt es später eine Reihe weiterer guter Webseiten und Bücher.

Systemvoraussetzungen: MySQL Datenbank, Java 5 oder höher, Entwicklungsumgebung mit XML und Java Support, JDBC MySQL Treiber, Hibernate Core, Hibernate Annotations sollte man alles bei der Suchmaschine seines Vertrauens finden können.

Bei den Bezeichnungen der einzelnen Dateien orientiere ich mich an der obigen Grafik. Um uns späteren Ärgern zu ersparen sollte man zuerst die ganzen JARs in sein Projekt einbinden. Normalerweise sollte man immer so wenig einbinden aber da uns die Performance im Moment egal ist und wir erstmal nur sehen wollen wie es funktioniert, binden wir wahllos alles ein.

Als nächstes brauchen wir ein Objekt, den Tisch.

```
[java]
// File: Tisch.java

package obj;

import java.io.Serializable;
import java.text.MessageFormat;

public class Tisch implements Serializable{

    private static final long serialVersionUID = 1L;

    private Integer id;
    private Integer Fuesse;
    private String Material;
    private Tisch tisch;

    public Tisch(){}

    private Integer getId() {return id;}

    private void setId(Integer id) {this.id = id;}

    public Integer getFuesse() {return Fuesse;}

    public void setFuesse(Integer fuesse) {Fuesse = fuesse;}

    public String getMaterial() {return Material;}
```

```
public void setMaterial(String material) {this.Material = material;}

private Tisch getTisch() {return tisch;}

private void setTisch(Tisch tisch) {this.tisch = tisch;}

public String toString() {
    return MessageFormat.format("{0}: Fuesse={1}, Material={2}", new Object[] {
        getClass().getSimpleName(), Fuesse, Material });
}
}
[/java]
```

Aus Gründen der Übersichtlichkeit habe ich den Code hier etwas zusammengefasst. Anschließend müssen wir mittels XML sagen wie der Tisch in unserer Datenbank gemapped werden soll.

[xml]

[/xml]

Damit später nur eine Verbindung zur Datenbank vorhanden ist muss man die Session als Singleton implementieren.

```
[java]
// File: SingSessionFactory.java

package hibe;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class SingSessionFactory {

    private static org.hibernate.SessionFactory sessionFactory;

    private SingSessionFactory() {}
    static {

        final Configuration cfg = new Configuration();
```

```
cfg.configure("/hibernate.cfg.xml");
cfg.buildSessionFactory();
sessionFactory = cfg.buildSessionFactory();
}

public static SessionFactory getInstance() {
    return sessionFactory;
}
}
[/java]
```

Anschließend können wir unsere Main implementieren.

```
[java]
// File: HibeMySQL.java

package exa;

import hibe.SingSessionFactory;
import org.apache.log4j.Logger;
import org.hibernate.Session;
import org.hibernate.Transaction;

import obj.Tisch;

public class HibeMySQL {

    private static Logger log = Logger.getLogger(HibeMySQL.class);

    public static void main(String[] args) {
        Tisch wohnzimmer = new Tisch();
        wohnzimmer.setMaterial("Eiche");
        wohnzimmer.setFuesse(3);

        Session ses = SingSessionFactory.getInstance().getCurrentSession();
        Transaction tx = ses.beginTransaction();

        ses.save(wohnzimmer);

        tx.commit();
    }
}
[/java]
```

Am Schluss brauchen wir noch zwei Konfigurationsdateien. Die erste für Hibernate und der MySQL Verbindung.

```
[xml]
```

```
jdbc:mysql://localhost:3306/bank
root
com.mysql.jdbc.Driver
org.hibernate.dialect.MySQLDialect
```

```
org.hibernate.transaction.JDBCTransactionFactory
thread
true
```

```
true
```

```
update
```

```
[/xml]
```

Als letztes brauchen wir noch eine Konfiguration für Log4j damit wir eine schöne Ausgabe erhalten was die Datenbank so treibt.

```
[xml]
# File: log4j.properties

### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

### direct messages to file hibernate.log ###
#log4j.appender.file=org.apache.log4j.FileAppender
#log4j.appender.file.File=hibernate.log
#log4j.appender.file.layout=org.apache.log4j.PatternLayout
#log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

log4j.rootLogger=debug, stdout

log4j.logger.org.hibernate=info
#log4j.logger.org.hibernate=debug

### log HQL query parser activity
#log4j.logger.org.hibernate.hql.ast.AST=debug

### log just the SQL
log4j.logger.org.hibernate.SQL=debug

### log JDBC bind parameters ###
```

```
log4j.logger.org.hibernate.type=info
#log4j.logger.org.hibernate.type=debug

### log schema export/update ###
log4j.logger.org.hibernate.tool.hbm2ddl=info

### log HQL parse trees
#log4j.logger.org.hibernate.hql=debug

### log cache activity ###
log4j.logger.org.hibernate.cache=info

### log transaction activity
#log4j.logger.org.hibernate.transaction=debug

### log JDBC resource acquisition
#log4j.logger.org.hibernate.jdbc=debug

### enable the following line if you want to track down connection ###
### leakages when using DriverManagerConnectionProvider ###
#log4j.logger.org.hibernate.connection.DriverManagerConnectionProvider=trace
[/xml]
```

Die Konfigurationseinstellungen sind zum größten Teil aus anderen Beispielen entliehen. Es geht hier prinzipiell nur darum mal ein einfaches Beispiel zu zeigen.