http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte2:psl:dataaccess:getting_started

Getting Started

MyBatis applications use SqlSessionFactory instances with no exception. SqlSessionFactory instances is based on SqlSessionFactoryBuilder which builds SqlSessionFactory instances in the XML configuration file.

How to build SqlSessionFactory in XML

Building an SqlSessionFactory instance from an XML file is not a big deal. You may either, use the classpass resources, as recommended, for configuration, or apply the InputStream instances generated out of the URL. MyBatis features loading of resources from the directory classified that of the classpass, by way of utility class called 'Resources'.

```
String resource = "org/mybatis/example/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
```

In MyBatis, core configurations to be made in the XML configuration file include TransactionManager, intended to control transactions, and DataSource to load the database connection instances.
Refer to the following for the coding example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

The foregoing example represents the core elements of potential configuration options for XML. At the top of the foregoing example represents the part intended to check the validity of the XML document. The environment factor in XML represents management of transactions and pooling of connections. The mapper element designates SQL code and breaks down the mapper files in the form of an XML file.

How to build SqlSessionFactory XML-free?

If you prefer Java over XML, you can use the class Configuration for configuration class that offers configuration in the XML file.

```
DataSource dataSource = DeptDataSourceFactory.getDeptDataSource();
TransactionFactory transactionFactory = new JdbcTransactionFactory();
Environment environment = new Environment("development", transactionFactory, dataSource);
Configuration configuration = new Configuration(environment);
configuration.addMapper(DeptMapper.class);
```

```
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(configuration);
```

What you need to work on is to add the class Mapper, a Java class comprising SQL mapping annotations in it. Despite a few restrictions and complexity in configuration, you need to make sure XML mapping is available when you want to map in detail.

How to generate SqlSession in SqlSessionFactory

As the title says it all, SqlSessionFactory can generate an SqlSession instance. In the SqlSession methods contains the methods you need to execute SQL commands, by which you can execute the SQL syntaxes via SqlSession instances. Refer to the following for the coding example:

```
 SqlSession session = sqlSessionFactory.openSession();
 try {
    Dept dept = session.selectOne("egovframework.rte.psl.dataaccess.DeptMapper.selectDept", 101);
 } finally {
    session.close();
 }
```

You might be familiarized with the codes above if you have used one of the previous editions of MyBatis. With the interface describing the parameters and return values in SQL syntaxes not available, (e.g. DeptMapper.class) you may execute the commands without getting bothered by syntax and casting errors.

Refer to the following for the coding example:

```
SqlSession session = sqlSessionFactory.openSession();
try {
   DeptMapper mapper = session.getMapper(DeptMapper.class);
   Dept dept = mapper.selectDept(101);
} finally {
   session.close();
}
```

We're now moving on to more details:

SQL syntaxes mapped

Let us take a brief look at how the classes SqlSession or Mapper are executed. The first thing you need to do is to understand how the mapped SQL syntax is comprised of, the content that you'll find more often than any other. Refer to the following two examples about how SQL syntaxes are mapped.

Before that, you need to keep in mind that the syntax can be defined either by XML or annotation. As for XML, the most common means of mapping for the greater part of MyBatis functions, you'll need to learn from the scratch as those new XML functions must be something that you've never experienced before. Refer to the following coding example for how to map XML to call SqlSession:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
       "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="egovframework.rte.psl.dataaccess.DeptMapper">
        <select id="selectDept"
                parameterType="int"
                    resultType="Dept">
                 <![CDATA[
                        select *
                        from    DEPT
                        where   DEPT_NO = #{deptNo}
                 ]]>
        </select>
```

</mapper>

You can define multiple mapping syntaxes in a single mapper XML file and these are pretty straight-forward, save for the XML headers and doctype. You'll need to define mapping syntaxes in the Namespace egovframework.rte.psl.dataaccess.DeptMapper, defined in the form of egovframework.rte.psl.dataaccess.DeptMapper.selectDept. Refer to the following for the executable form:

Dept dept = (Dept) session.selectOne("egovframework.rte.psl.dataaccess.DeptMapper.selectDept", 101);
you might have familiarized this one to calling the in-class methods. Note that the mapped information comprising the syntax, parameter and return type can be mapped along with the mapper class that is subject to the same parameter and return type, allowing you to call the method in a much simplified manner. Refer to the following example for how to code as mentioned above in an executable form:

DeptMapper mapper = session.getMapper(DeptMapper.class); Dept dept = mapper.selectDept(101);
You can take a variety of advantages working the foregoing coding example as it no longer depends on the syntax, thereby improving reliability of the application. Also, you do not need to be bothered by developing all those SQL syntaxes, not to mention that type casting to the return type is no longer required. All these features make the DeptMapper interface and parameters much more straight-forward and secure.

Note: Namespaces Description

Use of Namespaces are no longer optional, as you must put them to tell the syntax inclusive of the package names from the rest.

With Namespaces, you can work the interface binding, making your coding much more straight forward and applicability of MyBatis much more improved.

Name Resolution: To reduce typing needs, you'll need to apply the name resolution rules for MyBatis to refer to the syntax, result map and cache.

- If you have any fully qualified name such as "com.mypackage.MyMapper.selectAllThings", you can access the corresponding directory to look for the names desired.
- You can also use the shorter name such as "selectAllThings" to refer to the unequivocal entries. With you getting a step closer to the errors when using short names, you are advised to use fully qualified names.

References

- http://mybatis.github.io/mybatis-3/ko
- http://mybatis.github.io/mybatis-3/getting-started.html