



Java Programming

Languages in Depth Series

JDBC
Spring Framework
Web Services

Marco Piccioni
May 31st 2007



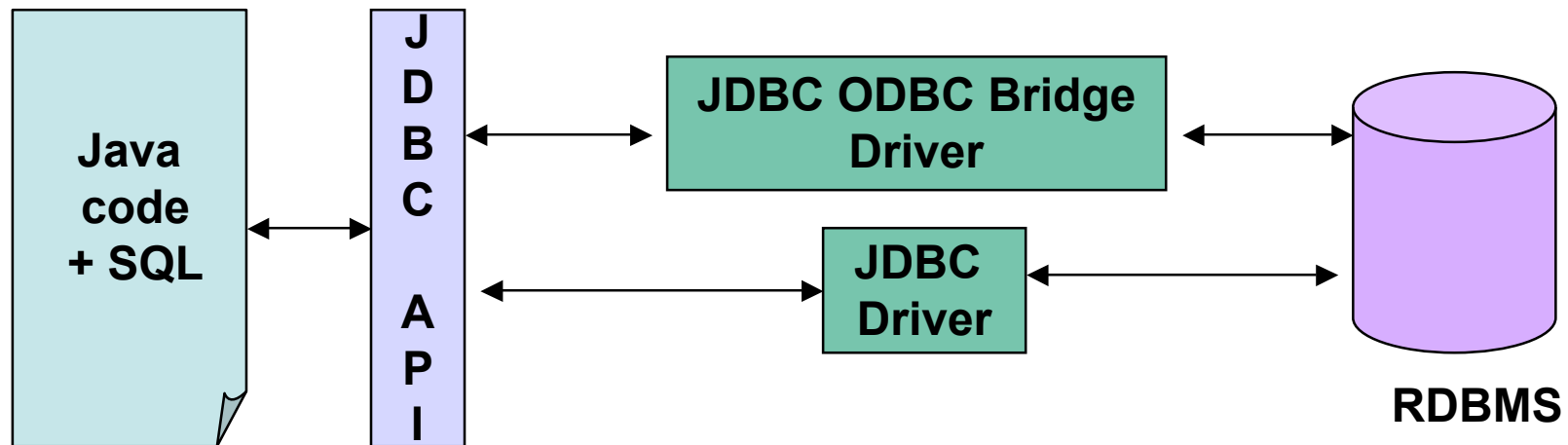
What will we be talking about

- Java Data Base Connectivity
- The Spring Framework: introduction
- Spring and data access: JDBC support
- Web Services
- Spring web services support



Java Data Base Connectivity

General picture



Connecting via a *DriverManager* object



- The basic service for managing a set of JDBC drivers
- We need to load and create an instance of the driver and to register it with the *DriverManager*

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")
```

- How can this happen?



Getting a connection

- We need to get the connection object that will be used to communicate with the database

```
Connection con = DriverManager.getConnection  
("jdbc:odbc:MyDSN", "MyLogin", "MyPassword")
```

- Java code just knows about DSN (Data Set Name) which is in turn mapped to the real database name
 - For example in Windows you can use Control Panel (ODBC Data Source) to set the mapping

Connecting via a *DataSource* object



- A factory for connections to the physical data source
- An object that implements the *DataSource* interface will typically be registered with a naming service based on the Java Naming and Directory Interface (JNDI) API
- Implemented by each driver vendor
 - Basic -- produces a *Connection* object
 - Connection pooling -- produces a *Connection* object that will automatically participate in connection pooling



Statements and queries

- The *Statement* class is used to send SQL statements to the DBMS

```
Statement stmt = con.createStatement();
```

- A sample SQL query:

```
stmt.executeQuery("SELECT * FROM CUSTOMERS;");
```

- If you need an update, insert or delete:

```
stmt.executeUpdate("an SQL update, insert or  
delete here");
```



Handling the result set

- *executeQuery()* returns a *ResultSet* object

```
ResultSet rs = stmt.executeQuery("SELECT...");
```

- You can use the *ResultSet* object to iterate through the result

- The *next()* method moves the cursor to the next row, and returns false when there are no more rows
- There are *getter* methods (*getBoolean()*, *getLong()*, etc.) for retrieving column values from the current row



Cleaning up everything

- Remember to explicitly destroy the connection and the statement objects after having done
 - `con.close()`
 - `stmt.close()`

- How can you be sure about that?



A JDBC sample

```
public String getPassword(String name) throws
    ApplicationException{
    String sql = "SELECT password FROM Customers WHERE
    id='" + name + "' ";
    String password = null;
        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        try {            con = <get connection from DataSource>;
s = con.createStatement();
rs = s.executeQuery (sql);
while (rs.next()) { password = rs.getString(1);
}
        rs.close(); s.close();        con.close();
        }catch (SQLException ex) { throw new
ApplicationException ("Couldn't run query [" + sql +
    "]", ex);        }        return password;    }
```



A JDBC sample (improved)

```
public String getPassword(String name) throws
    ApplicationException{
    <as before...>
    rs. close(); s. close();
    } catch (SQLException ex) { throw new
    ApplicationException ("Couldn't run query [" + sql +
    "]", ex);    }
    finally {
        try {
            if (con != null) { con. close();}
        } catch (SQLException ex) { throw new
    ApplicationException ("Failed to close connection",
    ex);}
    } //end finally
    return password;
}
```

Statement issues



- Across different DBMS
 - Strings quoting may be different
 - Code gets bloated with ugly string concatenations

```
String query = "Select * from Courses where " +  
" id = \"\" + id + \"\" \" +  
" and name = \"\" + name + \"\" \" +  
" and year = \" + year;
```



Prepared statements

- Prepared statements (pre-compiled queries) are more readable and the resulting code is more portable
- Favour query optimization

```
String SQL = "select * from Courses where id = ? and  
name = ? and year = ? ";
```

```
PreparedStatement pstmt =  
connection.prepareStatement(SQL);
```

```
pstmt.setString(1,id);
```

```
pstmt.setString(2,name);
```

```
pstmt.setInt(3,year);
```

```
pstmt.executeQuery();
```



The Spring Framework: introduction

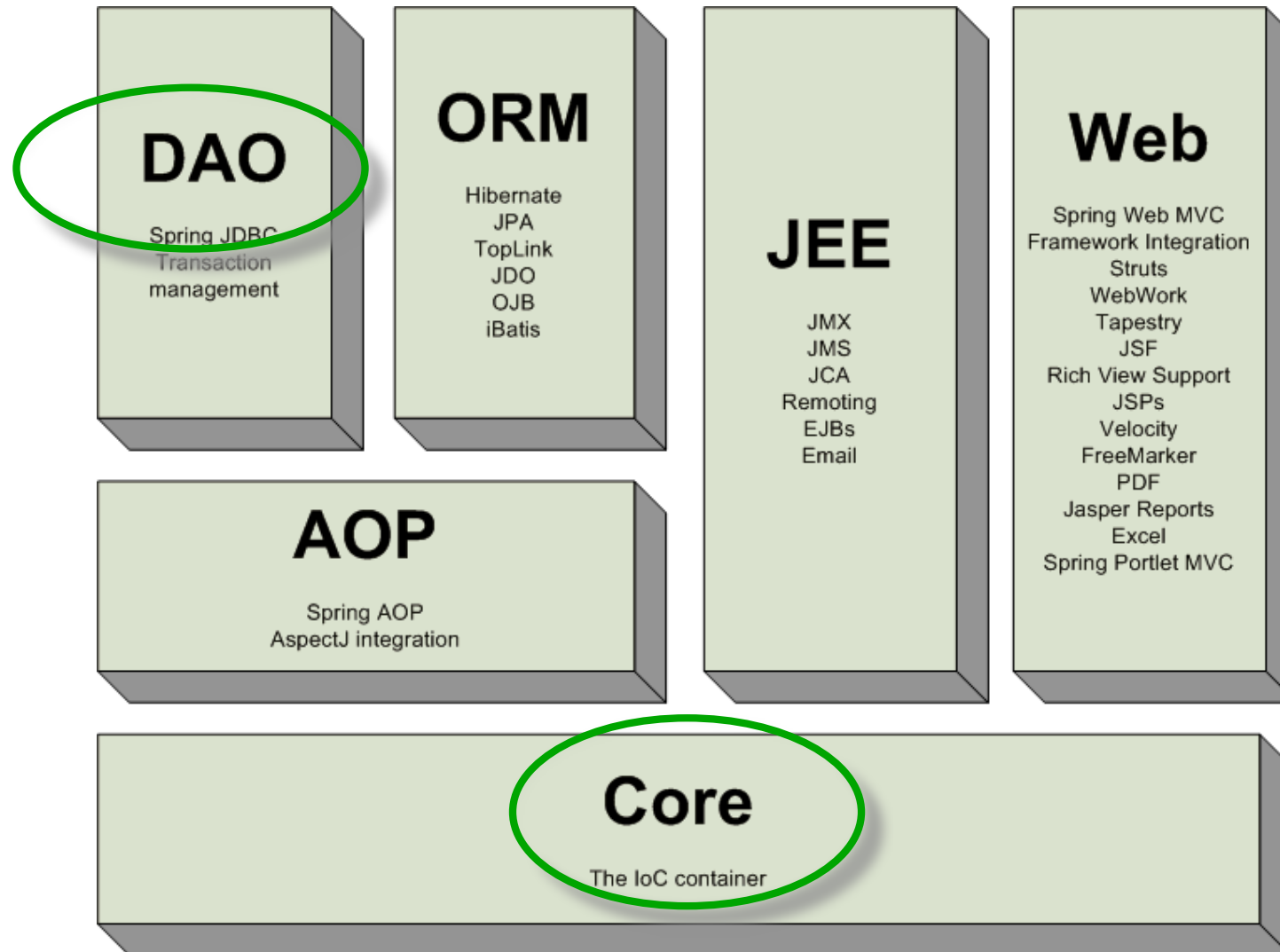


Yet another framework?

- It addresses areas that other frameworks don't
- Does not reinvent the wheel
- It is both comprehensive and modular
- Consistent and simple programming model (based on POJOs)
- Designed to help in writing code that is easy to unit test
- Effectively organizes middle tier objects
- Centralizes configuration
- A light weight alternative to EJB's (most of the time)
- Eliminates Singleton proliferation
- Consistent framework for data access
- Applies good OO practices
- Open source project since February 2003
- 20 fully devoted developers, more than 1 million downloads



The overall picture



Picture from <http://static.springframework.org/spring/docs/2.0.x/reference/introduction.html>



What's next?

- A step by step introduction to the Spring Core
 - Inversion of Control/Dependency Injection
 - Architectural model
 - Centralized configuration



A deceptively simple example

```
public class HelloWorld
{
    public static void main(String[] aaargh)
    {
        System.out.println("Hello again?!");
    }
}
```

Things happen to change...



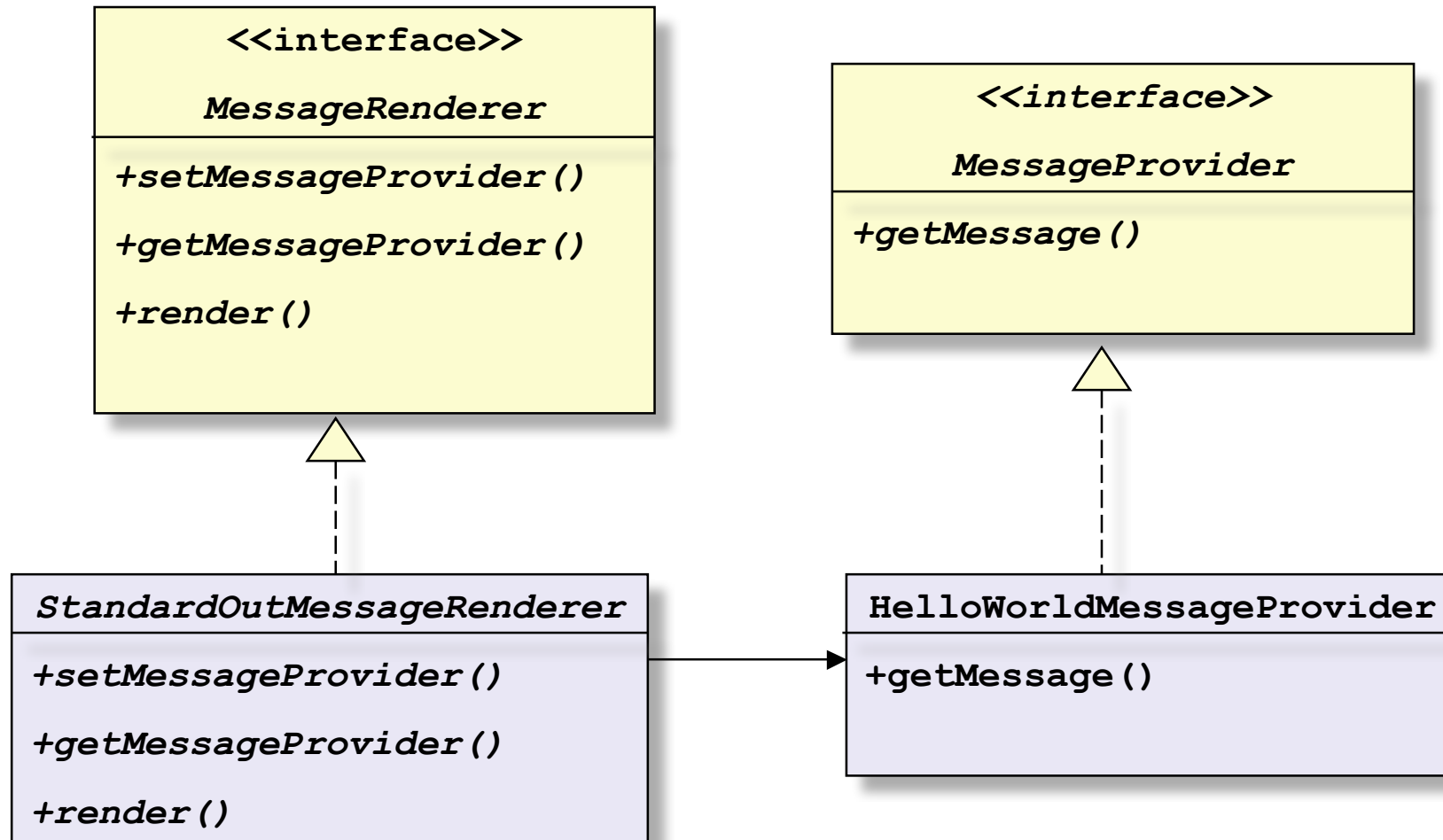
- What if we want to change the message?
- What if we want to output the message differently, maybe to *stderr* instead of *stdout*, or enclosed in *HTML tags* rather than as plain text?
- New requirements
 - Our application should support a simple, flexible mechanism for changing the message, and
 - It should be simple to change the rendering behavior.



Two little exercises

- Which is the simplest way in which we can refactor the code so that we can change the message without changing the code?
 - Using it as a command line argument
- How to decouple the message handling in the code?
 - Separating the component that renders the message from the component that obtains the message

A more flexible design





A decoupled Hello World

```
public class DecoupledHelloWorld
{
    public static void main(String[] args)
    {
        MessageRenderer mr = new
            StandardOutMessageRenderer();
        MessageProvider mp = new
            HelloWorldMessageProvider();
        mr.setMessageProvider(mp);
        mr.render();
    }
}
```



But we can do better...

```
public class DecoupledHelloWorldWithFactory
{
    public static void main(String[] args)
    {
        MessageRenderer mr=
MessageSupportFactory.getInstance().getMessageRe
nderer();
        MessageProvider mp =
MessageSupportFactory.getInstance().getMessagePr
ovider();
        mr.setMessageProvider(mp);
        mr.render();
    }
}
```



A Factory snippet

```
private MessageSupportFactory() {
    bundle=ResourceBundle.getBundle("ch.ethz.inf.java.hello.msgConf");
    String
    rendererClass=bundle.getString("renderer.class");
    String
    providerClass=bundle.getString("provider.class");
    try {
        renderer=(MessageRenderer)
        Class.forName(rendererClass).newInstance();
        provider=(MessageProvider)
        Class.forName(providerClass).newInstance();
    } catch (InstantiationException e) {
        //exception handling code
    } catch (IllegalAccessException e) {
        //exception handling code
    } catch (ClassNotFoundException e) {
        //exception handling code
    }
}
```



Flexibility comes at a price

- We had to write lots of “glue” code
- We still had to provide the implementation of *MessageRenderer* with an instance of *MessageProvider* manually

Hello Spring



- Here is where the Spring Framework comes into play
 - Provides lots of well written plumbing code
 - Sets the dependencies automatically



Let's have a look at this

```
public class XmlConfigHelloSpringDI {
    public static void main(String[] args) throws
    Exception
    {
        BeanFactory factory = getBeanFactory();
        MessageRenderer mr =
        (MessageRenderer) factory.getBean("renderer");
        mr.render();
    }

    private static BeanFactory getBeanFactory() throws
    Exception
    {
        XmlBeanFactory factory = new XmlBeanFactory(new
        FileSystemResource("src/ch/ethz/inf/java/hello/msgConf
        .xml"));
        return factory;
    }
}
```



Sample Spring configuration file

```
<!DOCTYPE beans PUBLIC "-//SPRING/DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-
beans.dtd">
<beans>
  <bean id="renderer"

class="ch.ethz.inf.java.hello.StandardOutMessageRen
derer">
    <property name="messageProvider">
      <ref local="provider"/>
    </property>
  </bean>
  <bean id="provider"

class="ch.ethz.inf.java.hello.HelloWorldMessageProv
ider"/>
</bean></beans>
```



Kinds of dependency handling

- Dependency lookup
 - Dependency pull
- Dependency injection (DI)
 - Setter DI
 - Constructor DI

The Spring *BeanFactory* interface



- Your application interacts with the Spring DI container via the *BeanFactory* interface
- All the *BeanFactory* implementations act as centralized registers that provide beans by name

The Spring *ApplicationContext* interface



- Provides more services with respect to the *BeanFactory* interface
 - Internationalization
 - Handles many configuration files
 - Publishes events
 - Handles more in depth beans lifecycle
 - ...



Externalizing the message

- *HelloWorldMessageProvider* returns the same hard-coded message for each call of *getMessage()*
- In the Spring configuration file, I can easily create a configurable *MessageProvider* that allows the message to be defined externally



A configurable message provider

```
public class ConfigurableMessageProvider
    implements MessageProvider
{
    private String message;

    private ConfigurableMessageProvider (String
message)
    {
        this.message=message;
    }

    public String getMessage ()
    {
        return message;
    }
}
```



Constructor injection sample

...

```
<bean id="provider"
  class="ch.ethz.inf.java.hello.ConfigurableMessageProvider">
  <constructor-arg>
    <value>This is a configurable
      message</value>
  </constructor-arg>
</bean>
```

...



Configuring values and references

- The framework provides many *PropertyEditor* implementations to convert the string values in the configuration file into:
 - URL's, Locales, file paths,...

- You can also set references and in particular handle collections



Sample reference configuration

```
...
<bean id="sportsman" class="ch.ethz.inf.java.injection.Sportsman">
  <property name="sponsor">
    <ref local="aSponsor"/>
  </property>
  <property name="sportEvents">
    <map>
      <entry key="Olympiads 2000">
        <value>100 mt</value>
      </entry>
      <entry key="Olympiads 2004">
        <value>200 mt</value>
      </entry>
    </map>
  </property>
</bean>
<bean id="aSponsor"
  class="ch.ethz.inf.java.injection.TechnicalSponsor">
  <property name="name">
    <value>Adidas</value>
  </property>
</bean>
...
```



Singleton issues

- Dependence on the singleton class is hard coded in many other classes
- Singletons are interface-unfriendly
- Every singleton must handle its own configuration (private constructor...)
- Testing with mock objects is more difficult
- It's difficult to manage all singletons configuration in a centralized and consistent way
- No inheritance (no overriding of *static* methods...)



The Spring solution

...

```
<bean id="nonSingleton" class="pickupkg.aClass"  
      singleton="false"/>
```

...

➤ As a default the framework provides singletons



Singletons in multithreaded environments

When yes:

- Shared objects with no state
- Shared objects with read-only state
- Shared objects with shared state (use synchronization)
 - When there are only a few writings
 - When many objects are created
 - When creation is expensive
 - Synchronization granularity should be high

Singletons in multithreaded environments (2)

When no:

- Shared objects with many writable attributes
- Objects with private state (not intended to be conceptually shared)



Spring and data access: JDBC support



A JDBC sample (improved)

```
public String getPassword(String name) throws
    ApplicationException{
    <as before...>
    rs. close(); s. close();
    } catch (SQLException ex) { throw new
    ApplicationException ("Couldn't run query [" + sql +
    "]", ex);    }
    finally {
        try {
            if (con != null) { con. close();}
        } catch (SQLException ex) { throw new
    ApplicationException ("Failed to close connection",
    ex);}
    } //end finally
    return password;
}
```



What's wrong with JDBC?

- JDBC is a good, but quite low level API
- It requires lots of effort on the developer side to write and maintain glue code
 - Loading and registering driver
 - Handling connections
 - Handling exceptions (JDBC API provides only a few exceptions)
 - Handling CRUD operations in a standard way



Spring and data exceptions

- Spring provides a wealth of useful **runtime** exceptions:
 - `DataAccessResourceFailureException`
 - `CleanupFailureDataAccessException`
 - `InvalidDataAccessApiUsageException`
 - `TypeMismatchDataAccessException`
 - `InvalidDataAccessResourceUsageException`
 - `DataRetrievalFailureException`
 - `IncorrectUpdateSemanticDataAccessException`
 - `DataIntegrityViolationException`
 - `DeadlockLoserDataAccessException`
 - `OptimisticLockingFailureException`
 - `UncategorizedDataAccessException`
 - ...

Spring and data exceptions (2)



- Checked exceptions are overused in Java
- Spring will not force you to catch any exception from which you are unlikely to be able to recover
- All Spring data exceptions are “unchecked”
- Quiz: can we try-catch a runtime exception?

Spring and data exceptions (3)



- Spring provides a default implementation of the *SQLExceptionTranslator* interface, which takes care of translating the generic SQL error codes into runtime Spring JDBC exceptions
- You can even handle specific error codes by extending *SQLErrorCodeSQLExceptionTranslator* and overriding `DataAccessException customTranslate (String task, String sql, SQLException sqllex)`



Class *JdbcTemplate*

- Handles JDBC core workflow
- It is configurable with a *javax.sql.DataSource* and publishes various methods to query and update data
- Intercepts JDBC exceptions translating them
- A developer has to implement some callback interfaces
 - *PreparedStatementSetter*
 - *PreparedStatementCallback*
 - *RowCallbackHandler*
 - *RowMapper*



JdbcTemplate: useful methods

```
List query(String query,  
           PreparedStatementSetter pss,  
           RowCallbackHandler rch)
```

```
int update(String updateQuery,  
           PreparedStatementSetter pss)
```



JdbcTemplate: sample code

...

```
JdbcTemplate jdbcTemplate = new
JdbcTemplate(dataSource);
ObjectRowMapper rowMapper = new
UserRowMapper();
List allUsers =
jdbcTemplate.query("select * from
user", new
RowMapperResultReader(rowMapper,
10));
```

...

JdbcTemplate: sample code (2)



...

```
List userResults =
jdbcTemplate.query("select * from
user where id=?", new Object[] {id},
new RowMapperResultReader(rowMapper,
1));
User user = (User)
DataAccessUtils.uniqueResult(userResu
lts);
```

...



Web Services

Web Services basics



- A software system designed to support interoperable machine to machine interaction over a network (W3C)
- Self-contained and self-describing. Its interface is described in **WSDL**
- Clients and servers (**endpoints**) communicate via **XML** messages that follow the **SOAP** standard
- Provides an API accessible over a network and executed on a remote system hosting the requested services

Core specifications: SOAP



- Service Oriented Architecture Protocol (former Simple Object Access Protocol)
 - XML-based communication protocol
 - Has bindings to underlying protocols as HTTP, HTTPS, SMTP, XMPP
 - Message-wrapping format
 - Platform and language independent

Core specifications: WSDL

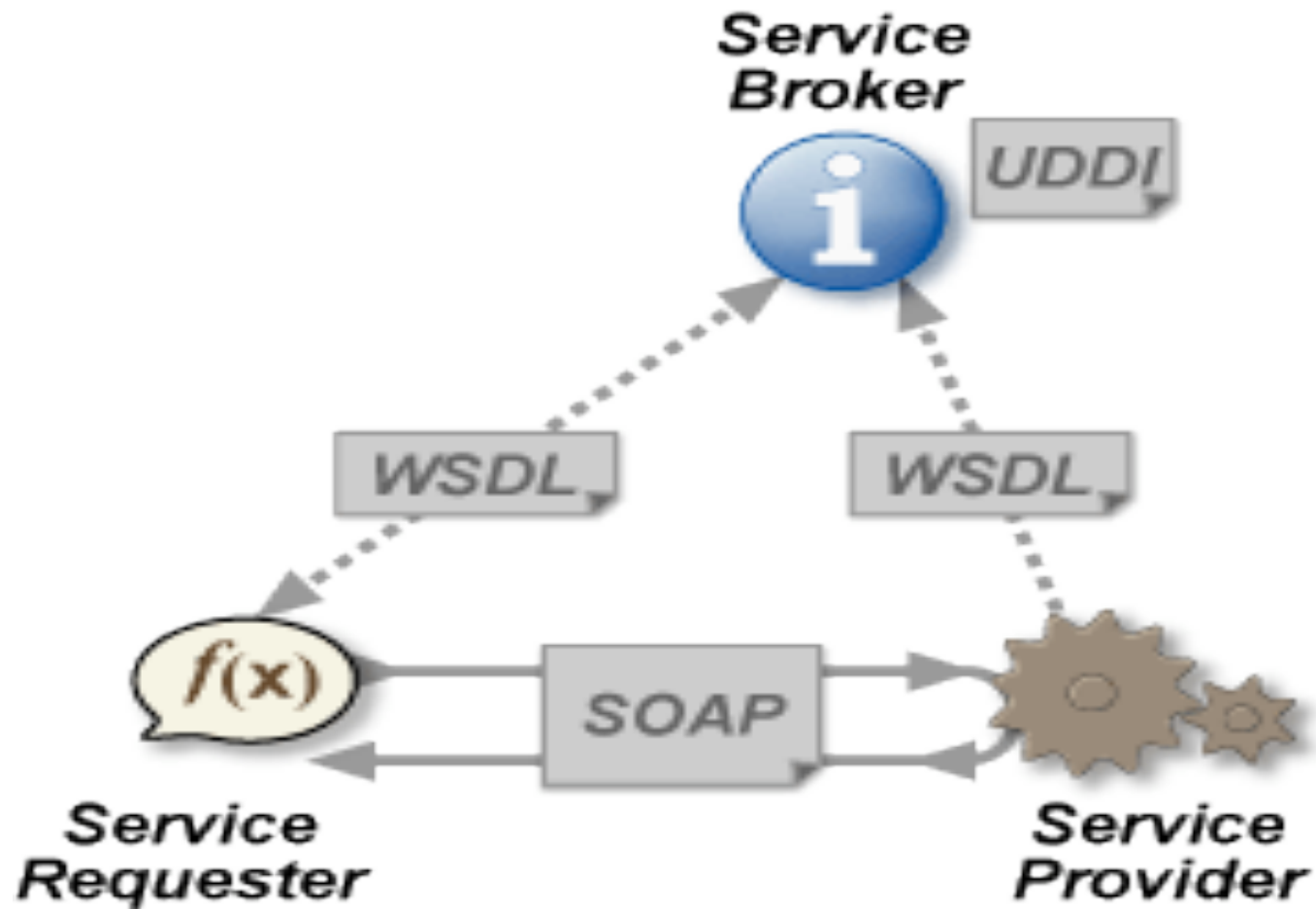


- Web Services Description Language
 - An XML format used to describe web services interfaces and protocol bindings details in a platform independent fashion
 - Also used for configuration and for generating client and server code
 - Heavily backed by Microsoft (uses it for .NET)



Core specifications: UDDI

- Universal Description, Discovery and Integration
 - A platform-independent, XML-based registry for businesses to list themselves on the internet
 - Each business can publish a service listing
- An UDDI business registration is composed by:
 - White Pages (address, contact and known identifiers)
 - Yellow Pages (industrial categorizations)
 - Green Pages (technical info about exposed services)
- It is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the web services listed in its directory



Picture from http://en.wikipedia.org/wiki/Web_services



Styles of use

- RPC (Remote procedure calls)
 - Present a distributed function call interface
 - Tightly coupled to language-specific function

- SOA (Service Oriented Architecture)
 - A message is the basic unit of communication
 - Focus is on WSDL contract

- RESTful (REpresentational State Transfer)
 - Constrain the interface to a set of standard operations
 - Focus is on interacting with stateful resources





A Java RPC-style implementation



- JAX-RPC (Java API for XML-based Remote Procedure Calls) provides a standard way for accessing and exposing RPC-style SOAP web services.
- JAX-RPC 2.0 has been renamed to JAX-WS 2.0 (Java API for XML Web Services)
- The most popular JAX-RPC flavor is Apache Axis, a fully JAX-RPC-compliant SOAP stack



How JAX-WS works

-  A Java program invokes a method on a stub (local object representing the remote service)
 -  The stub invokes routines in the JAX-WS Runtime System
 -  The JAX-WS Runtime System converts the remote method invocation into a SOAP message
 -  the JAX-WS Runtime System transmits the message as an HTTP request
- Therefore the web service can be implemented on the server side as a servlet or EJB container



Web Services: assessment

- Provide interoperability between different web applications running on different platforms
- Represent a possible solution for remoting
- Adopt a specific tradeoff, namely interoperability and extensibility over ease-of-use and performance
- RMI/EJB or Hessian/Burlap are better for plain Java to Java communication



Spring Web Services support



Spring and Web Services

- Spring Web Services is a separate Spring subproject
 - RC1 released on 22/5/2007
- Main features:
 - Contract-First development
 - Loose coupling between contract and implementation
 - XML API support (DOM, SAX, StAX, JDOM, dom4j, XOM,...)
 - Powerful mappings for distributing incoming XML requests to objects based on message payload, SOAP Action header, XPath expressions
 - Flexible XML marshalling (support for JAXB 1 & 2, Castor, XMLBeans, JiBX, XStream)
 - Support for security (sign, encrypt, decrypt and authenticate against SOAP messages)



Contract-First Web Services

- **Contract-Last approach**
 - Write (Java) code first (interfaces) , and let the Web Service contract (WSDL) be generated from that

- **Contract-First approach**
 - Write contract (WSDL) first, and use the language (Java) to implement that contract

Issues with Contract-Last development



- Object/XML impedance mismatch
- Fragility
- Performance
- Reusability
- Versioning



Object/XML impedance mismatch

- XML (and XSD) are hierarchical, OO languages are modeled by graphs
- XSD extensions are not like inheritance
 - E.g. a XSD restriction that uses a regular expression is lost
- Unportable types
 - e.g. `java.util.TreeMap`, time and dates
- Cyclic graphs
 - A refers to B that refers to A



Fragility and Performance

- Not all SOAP stacks generate the same web service contract from a Java contract
- By automatically transforming Java into XML, there is no guarantee on what is sent over the wire (dependencies can be very deep and unwanted)
- Using contract-first you can choose what information has to be sent



Reusability and Versioning

- Defining your schema in a separate file you can reuse it in different scenarios
- Starting from Java, when a contract evolves you have no choice but to assign a different name to your interface. The old contract must be kept around for clients that need the older version
- Starting from contracts, you can implement the old and the new version of the contract in one class using XSLT for conversions



How to write a contract-first web service

➤ See the tutorial at:

[http://static.springframework.org/spring-
ws/site/reference/html/tutorial.html](http://static.springframework.org/spring-
ws/site/reference/html/tutorial.html)



References

- Rob Harrop, Jan Machacek: *Pro Spring*. Apress 2005
- Rod Johnson: *Expert One-on-One J2EE Development without EJB*. Wrox Press, 2004
- Rod Johnson: *Expert One-on-One J2EE Design and Development*. Wrox Press, 2001
- Craig Walls, Ryan Breidenbach: *Spring in Action* Manning 2005



More references

- <http://www.springframework.org>
- <http://static.springframework.org/spring-ws/site/>
- <http://www.w3.org/TR/ws-arch>
- <http://ws.apache.org/axis>
- <http://www.w3schools.com/xml/default.asp>
- <http://www.w3schools.com/schema/default.asp>
- <http://www.w3schools.com/soap/default.asp>
- <http://www.w3schools.com/wsdl/default.asp>
- <http://www.soapui.org/>