

# DOM vs. JDOM

DOM	JDOM
<b>Kurzbeschreibung</b>	
<ul style="list-style-type: none"> <li>• DOM = Document Object Model</li> <li>• W3C-Standard</li> <li>• generisches Objektmodell („DOM-Baum“)</li> <li>• flexible Verarbeitung eines XML-Dokuments</li> <li>• sprachenunabhängig</li> <li>• Interface-basiert</li> </ul>	<ul style="list-style-type: none"> <li>• JDOM = Java Document Object Model</li> <li>• JSR-102 (Java Service Request)</li> <li>• Objektmodell („JDOM“)</li> <li>• flexible Verarbeitung von XML-Dokumenten mit Java</li> <li>• klassenbasiert</li> </ul>
<b>Geschichte</b>	
<ul style="list-style-type: none"> <li>• Anfänge von DOM, XML und JavaScript hängen zusammen</li> <li>• 1996: W3C entwickelt XML → Objektmodell als nächstes Ziel</li> <li>• 1997: Konsens über versch. Objektmodelle → W3C DOM</li> <li>• 10/1998: W3C DOM Level 1</li> <li>• 11/2000: W3C DOM Level 2</li> <li>• 04/2004: W3C DOM Level 3</li> </ul>	<ul style="list-style-type: none"> <li>• 4/2000: Ankündigung von JDOM</li> <li>• Anfang 2001: Integration als JSR-102 in JCP</li> <li>• 9/2004: JDOM 1.0 veröffentlicht</li> <li>• aktuelle Version: JDOM 1.1.1</li> </ul>
<b>Grundlegende Eigenschaften</b>	
<ul style="list-style-type: none"> <li>• In-Memory-Repräsentierung der XML</li> <li>• Parsen per Standard über Apache Xerces</li> <li>• Sprachenunabhängig durch Interface-basierte Spezifikation</li> <li>• Aktuelles Level: 3 (Erweiterungsschritte der Ur-DOM-Version)</li> <li>• Unterteilung des Funktionsumfangs in Module</li> <li>• Spezifische Implementierung muss lediglich Core-Modul enthalten, Rest optional</li> <li>• Universell, generisch (alles erbt vom Node-Interface), jedoch auch komplex</li> <li>• Nutzung DOM-eigener Collections</li> </ul>	<ul style="list-style-type: none"> <li>• In-Memory-Repräsentierung der XML</li> <li>• Parsen per Standard über JAXP</li> <li>• Java-optimiert</li> <li>• Keine Abstraktion von DOM!</li> <li>• Für Java-Programmierer intuitiv bedienbar</li> <li>• „Long-term Serialization“, da auch Setter komplettes Objekt zurückgeben</li> <li>• Nutzung von Java-Collections (Listen sind live!)</li> <li>• Method Overloading (eine Funktion – mehrere Ausführungen)</li> <li>• Apache-artige Lizenz</li> <li>• Fokus auf einfaches Handling</li> </ul>
<b>Objektmodell / Handling</b>	
<ul style="list-style-type: none"> <li>• Basis: Node-Interface</li> <li>• Spezifische Knotentypen (Text, Attribute, Comment...)</li> <li>• eigene Collection-Typen</li> <li>• „alles ist ein Knoten“</li> <li>• Instanzierung neuer Objekte über Document-Interface</li> </ul>	<ul style="list-style-type: none"> <li>• Jeder Knoten hat eigene Klasse</li> <li>• Instanzierung neuer Objekte mit new-Operator</li> <li>• Subclassing möglich, da klassenbasiert</li> <li>• Individuelle Factories</li> </ul>
<b>Anwendungszwecke</b>	
<ul style="list-style-type: none"> <li>• Für Business-Anwendung, wo Standardisierung eine Rolle spielt</li> <li>• Sprachunabhängiger Einsatz</li> </ul>	<ul style="list-style-type: none"> <li>• Schneller Einstieg in die modellbasierte Verarbeitung von XML in Java</li> </ul>

# DOM vs. JDOM - Cheatsheet

DOM	JDOM
<b>Parsen</b>	
<pre>DOMImplementationRegistry registry = DOMImplementationRegistry.newInstance(); DOMImplementationLS domImplLS = (DOMImplementationLS) registry     .getDOMImplementation("Core 3.0 LS");  LSInput input = domImplLS.createLSInput(); input.setByteStream(new FileInputStream(file));  LSParser parser = domImplLS.createLSParser(DOMImplementationLS .MODE_SYNCHRONOUS, null);  Document doc = parser.parse(input);</pre>	<pre>SAXBuilder builder = new SAXBuilder(); Document doc = builder.build(file);</pre>
<b>Traversieren</b>	
<ul style="list-style-type: none"> <li>• doc.getDocumentElement();</li> <li>• element.getChildNodes();</li> <li>• element.getFirstChild();</li> <li>• element.getLastChild();</li> <li>• element.getNextSibling();</li> <li>• element.getPreviousSibling();</li> <li>• element.getParentNode();</li> <li>• nodeList.item(index);</li> <li>• element.getElementsByTagName(name);</li> <li>• element.getElementsByTagNameNS(name);</li> <li>• xpath.evaluate(query, doc, returnType);</li> </ul> <pre>XPath xpath = factory.newXPath(); NodeList test = (NodeList) xpath.evaluate("//person/hobbys/hobby[3]", doc, XPathConstants.NODESET);</pre>	<ul style="list-style-type: none"> <li>• doc.getRootElement();</li> <li>• element.getContent();</li> <li>• element.getChildren(name);</li> <li>• element.getChild(name   ns);</li> <li>• element.getParent();</li> <li>• list.get(index);</li> <li>• über Filter realisierbar</li> <li>• XPath.selectNodes(doc, query);</li> </ul> <pre>List hobbys = XPath.selectNodes(doc, "//person/hobbys/hobby[3]");</pre>
<b>Abfragen von Inhalten</b>	
<ul style="list-style-type: none"> <li>• element.getTextContent();</li> <li>• element.getNodeType();</li> <li>• element.getNodeValue();</li> <li>• element.getNodeName();</li> <li>• attr.getValue();</li> </ul>	<ul style="list-style-type: none"> <li>• element.getText();</li> <li>• element.getChildText(name   ns);</li> <li>• element.getClass();</li> <li>• element.getAttributeValue(name   ns)</li> </ul>
<b>Erzeugen eines Documents</b>	
<pre>DOMImplementationRegistry registry = DOMImplementationRegistry.newInstance();  DOMImplementation domImpl = registry.getDOMImplementation("Core 3.0");  doc = domImpl.createDocument(null, "personen", null);</pre>	<pre>Document doc = new Document(new Element("personen"));</pre>

Erzeugen / Einfügen von Elementen	
<ul style="list-style-type: none"> <li>• <code>doc.createElement(tagName);</code></li> <li>• <code>doc.createTextNode(value);</code></li> <li>• <code>doc.createAttribute(name);</code></li> <li>• <code>element.appendChild(element);</code></li> <li>• <code>element.insertBefore(element);</code></li> <li>• <code>element.setAttribute(name, value);</code></li> <li>• <code>element.setTextContent(text);</code></li> <li>• <code>element.setNodeValue(value);</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>new Element(tagName);</code></li> <li>• <code>new Text(...);</code></li> <li>• <code>new Attribute(...);</code></li> <li>• <code>element.addContent(element);</code></li>   <li>• <code>element.setAttribute(name, value);</code></li> <li>• <code>element.setText(text);</code></li> </ul>
Löschen von Elementen	
<ul style="list-style-type: none"> <li>• <code>element.removeChild(node);</code></li>   <li>• <code>element.removeContent();</code></li> <li>• <code>element.removeAttribute(name);</code></li> <li>• <code>element.removeAttributeNode(attrNode);</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>element.removeChild(name   ns);</code></li> <li>• <code>element.removeChildren(name   ns);</code></li> <li>• <code>list.remove(index);</code></li> <li>• <code>list.removeAll();</code></li> <li>• <code>element.removeContent();</code></li> <li>• <code>element.removeAttribute(name / attr);</code></li> </ul>
Modifizieren von Elementen	
<ul style="list-style-type: none"> <li>• <code>element.cloneNode(boolean);</code></li>   <li>• <code>element.replaceChild(newChild, oldChild);</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>element.clone(element);</code></li> <li>• <code>element.detach(element);</code></li> </ul>
Filter	
<ul style="list-style-type: none"> <li>• <b>Filter definieren</b> und Funktion <code>acceptNode()</code> überschreiben</li> <li>• <b>Nodelterator erzeugen</b></li> </ul> <pre>NodeIterator ni = ((DocumentTraversal) doc). createNodeIterator(personen, NodeFilter.SHOW_ALL, new TextFilter(), false);</pre> <ul style="list-style-type: none"> <li>• <b>Iterieren und Nodes ausgeben</b></li> </ul>	<ul style="list-style-type: none"> <li>• <b>Filter definieren</b> und Funktion <code>matches()</code> überschreiben</li> <li>• <b>Iterator mit <code>getDescendants()</code> erzeugen</b></li> </ul> <pre>Iterator i = doc.getDescendants( new TextFilter());</pre> <ul style="list-style-type: none"> <li>• <b>Iterieren und Nodes ausgeben</b></li> </ul>
Serialisieren	
<pre>LSOutput output = domImplLS.createLSOutput(); output.setByteStream(new FileOutputStream("assets/test.xml"));  LSSerializer serializer = domImplLS.createLSSerializer(); serializer.getDomConfig().setParameter("form at-pretty-print", true); serializer.write(doc, output);</pre>	<pre>XMLOutputter outputter = new XMLOutputter(); outputter.setFormat(Format.getPrettyFormat() ); outputter.output(doc, new FileOutputStream("assets/test.xml"));</pre>

## Literaturempfehlungen

McLaughlin, Edelson: *Java & XML*, O'Reilly, 2006

Scholz, Niedermeier: *Java und XML*, Galileo Computing, 2009

Steyer: *XML mit Java*, S&S, 2005

<http://www.ibm.com/developerworks/java/library/j-jdom/>

<http://www.cafeconleche.org/books/xmljava/chapters/ch14.html>

<http://www.cafeconleche.org/books/xmljava/chapters/ch15.html>

<http://www.w3.org/dom/>

<http://www.jdom.org/>