

Императивна обработка на XML съдържание чрез Simple API for XML – SAX 2.0 и Streaming API for XML - StAX

Предимства на SAX

SAX интерфейси

Използване

StAX

DOM, StAX и SAX

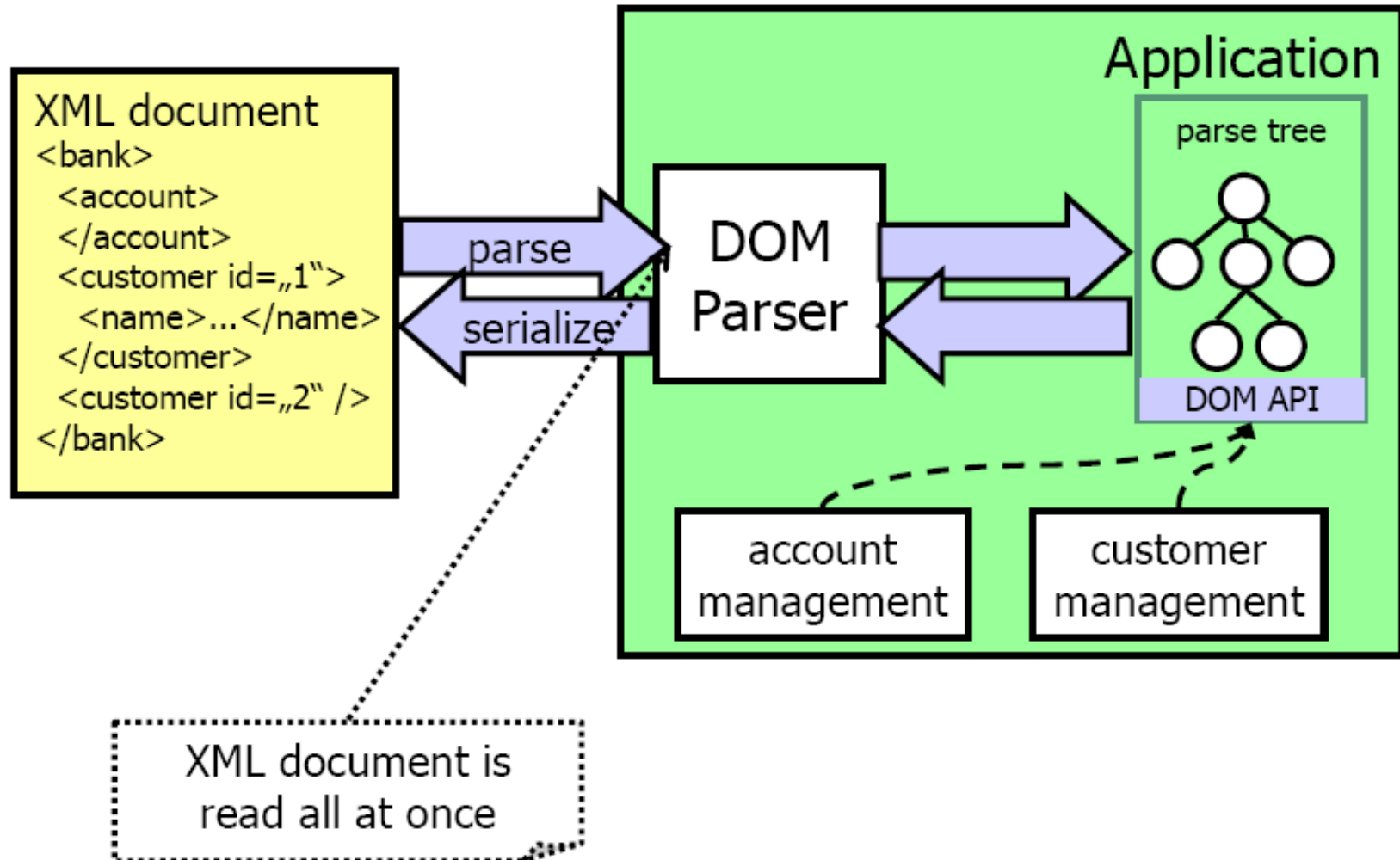
Типове XML парсъри

- Съществуват различни групи от типове:
 - Валидиращи спрямо невалидиращи парсъри
 - Парсъри, използващи Document Object Model (DOM)
 - Парсъри, използващи Simple API for XML (SAX)
 - Парсъри, използващи Streaming API for XML (StAX)
 - Парсъри, написани на конкретен език (Java, C++, Perl, etc.) без използване на определен API

Разбор (парсване) на XML съдържание

- Три широко-известни API's
 - DOM (Document Object Model)
 - Дефинира логическо дърво, представящо парсвания XML документ
 - SAX (Simple API for XML)
 - Дефинира манипулатори (handlers), съдържащи методи за разбор на XML документа (push)
 - Streaming API for XML (StAX)
 - Парсване на XML, базирано на итератори (a la push)
- Приложения без сложна манипулация на XML, но с ограничения по памет, могат да ползват SAX и StAX
- Структурната манипулация на XML елементи изисква използването на DOM

Работен процес на DOM



SAX спрямо DOM *(фигури от Beginning XML, 2nd Edition)*

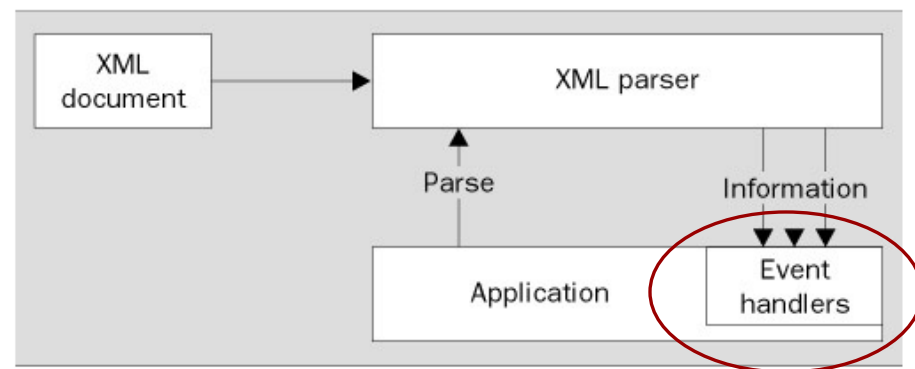
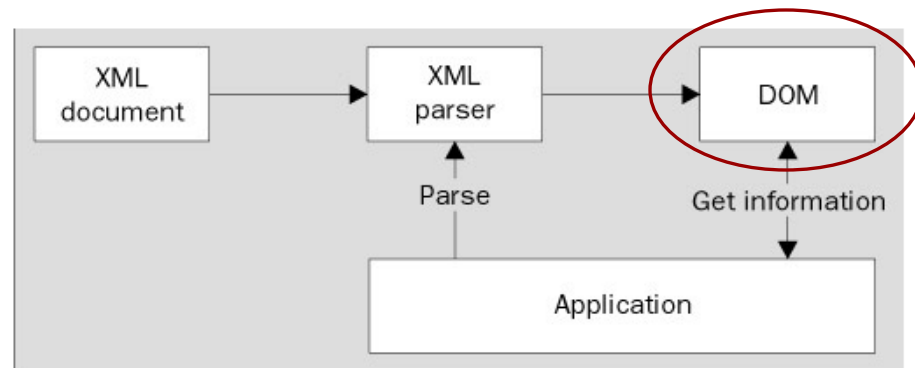
- За по-ефикасен анализ на големи *XML* документи
- Цел: да реши проблема на DOM – създаването на масивно дърво на документа в паметта, преди да започнем работа с него, т.е. така да спести:

- памет

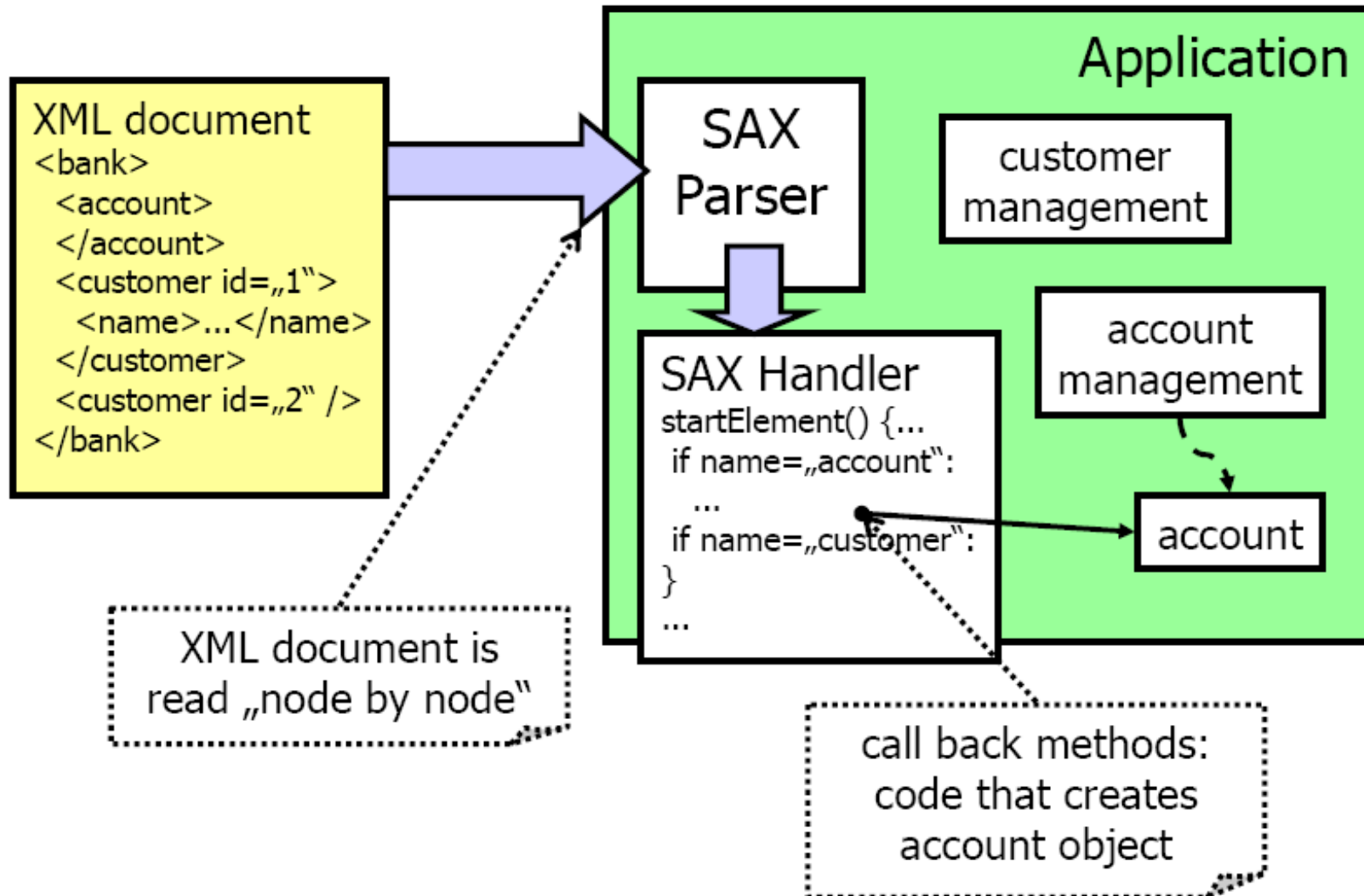
- време

- Решение:

SAX



Работен процес на SAX

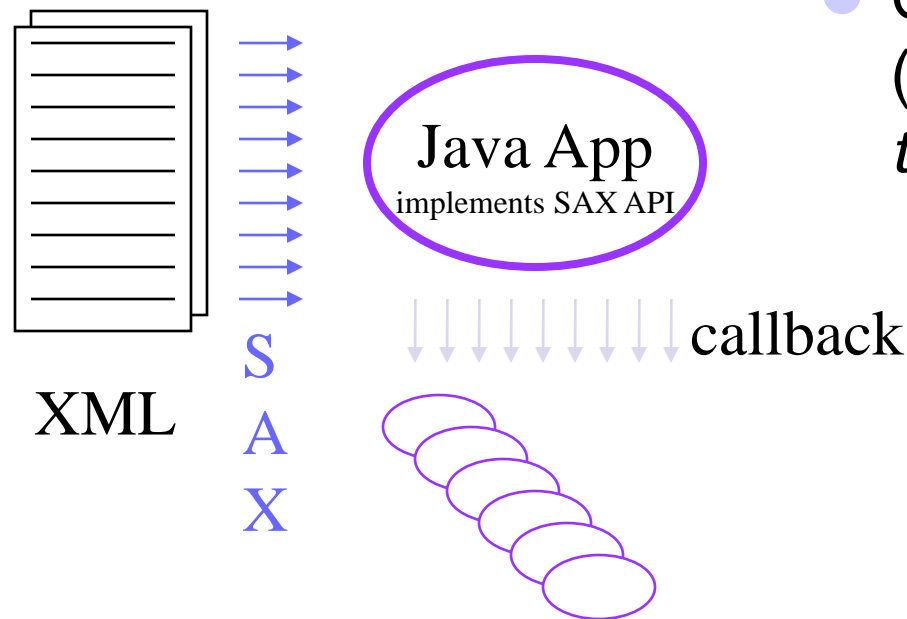


История на SAX

- Разработен от членовете на XML-DEV mailing list (сега поддържан от OASIS на <http://www.oasis-open.org/>), с цел *по-ефикасен анализ на големи XML документи*
- Спестява: **time & space**
- SAX 1.0 спецификация - май 1998
- SAX 2.0 спецификация - май 2000; David Megginson координира развитието на SAX (<http://www.megginson.com>)
- Голям брой SAX compliant парсъри — повечето отворени
 - Java парсъри - <http://xml.apache.org/xerces-j>, Crimson, nanoXML,

...

Simple API for XML - SAX



- С управление по събития (*event-driven API*) вместо *tree-based API*
 - XML документът се изпраща до SAX парсър
 - XML файлът се прочита ред по ред
 - Парсърът известява за събития, вкл. и грешки
 - Имплементациите на API методите обработват събитията

Парсър, управляван по събития

- За документа:
 - `<?xml version="1.0"?> <doc> <para>Hello, world!</para> </doc>`
- Парсър, управляван по събития, ще създаде събитията:
 - start document
 - start element: doc
 - start element: para
 - characters: Hello, world!
 - end element: para
 - end element: doc
 - end document
- Приложението ги обработва без да кешира целия документ

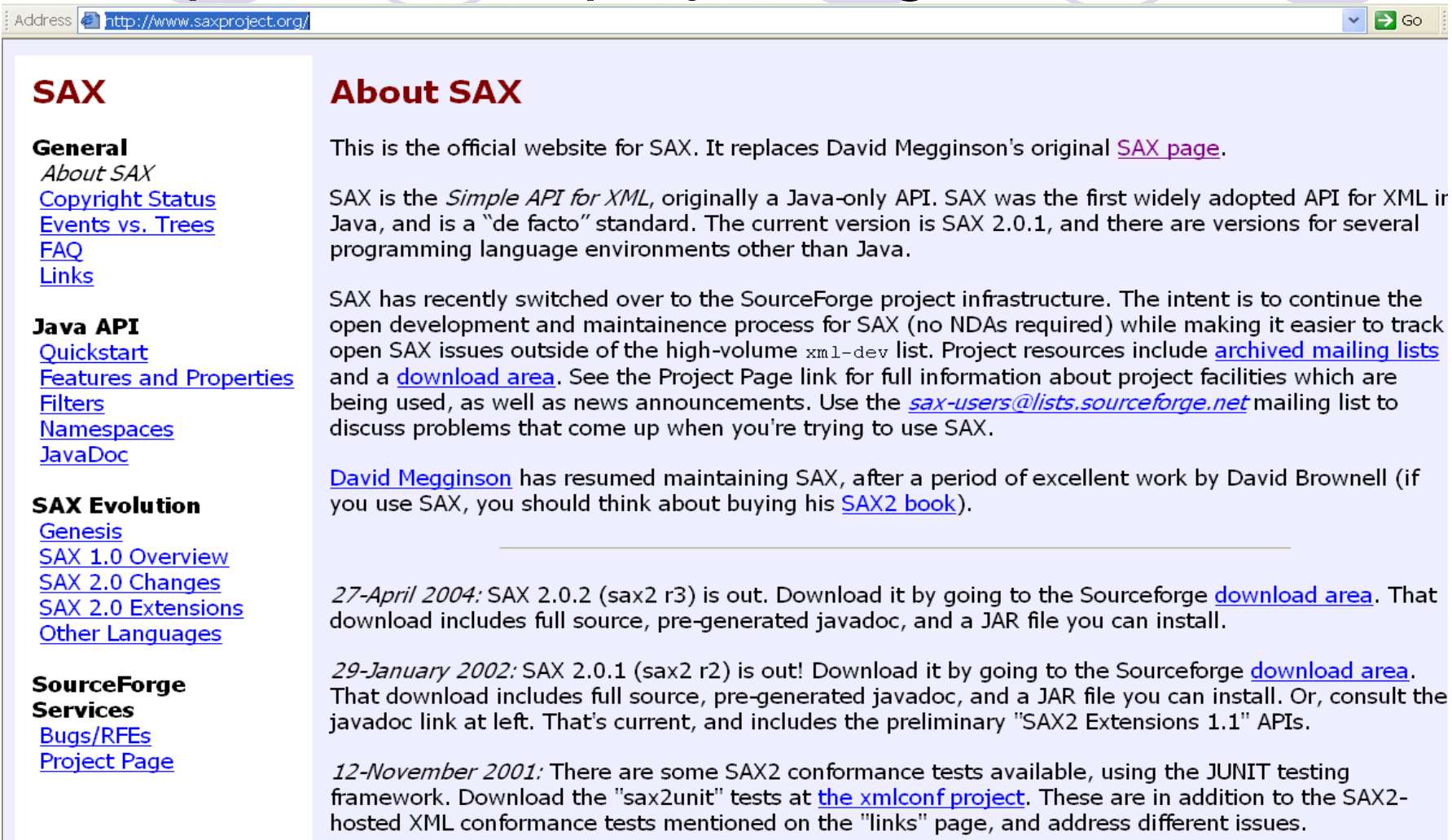
Simple API for XML - SAX

- SAX парсърът генерира събития
 - При начало и край на документа
 - При начало и край на елемент
 - При четене на символи в елемент
 - При грешки
 - При намиране на negligible whitespace
 - и много други...
- Използва *callback* механизъм за известяване на приложението
- Можем да напишем код за обработката на всяко събитие

A callback function that is passed (by reference) to another function, which calls the callback function under defined conditions (for example, upon completion).

Повече за SAX?

► <http://www.saxproject.org/>

A screenshot of a web browser displaying the SAX project website. The browser's address bar shows 'http://www.saxproject.org/'. The website has a light blue background. On the left, there is a sidebar with a table of contents. The main content area on the right is titled 'About SAX' and contains several paragraphs of text, including information about the project's history, its status as a standard, and recent updates. The text mentions David Megginson and David Brownell, and provides links to various resources like the 'download area' and 'mailing list'.

SAX

General
[About SAX](#)
[Copyright Status](#)
[Events vs. Trees](#)
[FAQ](#)
[Links](#)

Java API
[Quickstart](#)
[Features and Properties](#)
[Filters](#)
[Namespaces](#)
[JavaDoc](#)

SAX Evolution
[Genesis](#)
[SAX 1.0 Overview](#)
[SAX 2.0 Changes](#)
[SAX 2.0 Extensions](#)
[Other Languages](#)

SourceForge Services
[Bugs/RFEs](#)
[Project Page](#)

About SAX

This is the official website for SAX. It replaces David Megginson's original [SAX page](#).

SAX is the *Simple API for XML*, originally a Java-only API. SAX was the first widely adopted API for XML in Java, and is a "de facto" standard. The current version is SAX 2.0.1, and there are versions for several programming language environments other than Java.

SAX has recently switched over to the SourceForge project infrastructure. The intent is to continue the open development and maintenance process for SAX (no NDAs required) while making it easier to track open SAX issues outside of the high-volume `xml-dev` list. Project resources include [archived mailing lists](#) and a [download area](#). See the Project Page link for full information about project facilities which are being used, as well as news announcements. Use the sax-users@lists.sourceforge.net mailing list to discuss problems that come up when you're trying to use SAX.

[David Megginson](#) has resumed maintaining SAX, after a period of excellent work by David Brownell (if you use SAX, you should think about buying his [SAX2 book](#)).

27-April 2004: SAX 2.0.2 (sax2 r3) is out. Download it by going to the Sourceforge [download area](#). That download includes full source, pre-generated javadoc, and a JAR file you can install.

29-January 2002: SAX 2.0.1 (sax2 r2) is out! Download it by going to the Sourceforge [download area](#). That download includes full source, pre-generated javadoc, and a JAR file you can install. Or, consult the javadoc link at left. That's current, and includes the preliminary "SAX2 Extensions 1.1" APIs.

12-November 2001: There are some SAX2 conformance tests available, using the JUNIT testing framework. Download the "sax2unit" tests at [the xmlconf project](#). These are in addition to the SAX2-hosted XML conformance tests mentioned on the "links" page, and address different issues.

Пакетът org.xml.sax.*

- Основен SAX1 интерфейс
 - **DocumentHandler** – управление на събитията за съдържанието на документа в *SAX1*:
 - `StartDocument`
 - `EndDocument`
 - `StartElement`
 - `EndElement`
 - `Characters`
- **Deprecated.** *Този интерфейс е заменен в SAX2 от **ContentHandler**, който поддържа и Namespace.*
- Имплементация на **DocumentHandler** - **HandlerBase**

org.xml.sax.**ContentHandler**

- Основен нов интерфейс в SAX2
 - **ContentHandler** – управление на събитията за съдържанието на документа. Съдържа:
 - `StartDocument`
 - `EndDocument`
 - `StartElement`
 - `EndElement`
 - `Characters`
- Подобен на SAX 1.0 `DocumentHandler` (deprecated), но с поддръжка на пространства от имена
- Имплементация на **ContentHandler**: **DefaultHandler**
- Забележка: съществува Java клас с име **ContentHandler** в пакета `java.net`; внимание с:
- **`import java.net.*; import org.xml.sax.*;`**

Как да получаваме SAX събития

- **public class MyClass implements ContentHandler**
- **MyClass** ще имплементира callback методите на интерфейса за събития, свързани с парсването на съдържание (напр. събития за намирането на дадени елементи, атрибути и тяхното съдържание).
- Тези методи ще се извикват от SAX парсърът при настъпването на събитията.
- Интерфейсът **ContentHandler** съдържа голям брой методи, повечето от които са излишни за 80% от случаите. Затова SAX предоставя празна (default) имплементация на интерфейса, наречена **DefaultHandler**:
- **public class MyClass extends DefaultHandler**
- Можем да пренапишем (method **overriding**) онези методи, които обработват важни за нас събития.

Пример от “Beginning XML” — *the Band XML*

```
<?xml version="1.0"?>
<bands>
  <band type="progressive">
    <name>King Crimson</name>
    <guitar>Robert Fripp</guitar>
    <saxophone>Mel
Collins</saxophone>
    <bass>Boz</bass>
    <drums>Ian Wallace</drums>
  </band>
  <band type="punk">
    <name>X-Ray Spex</name>
    <vocals>Poly
Styrene</vocals>
    <saxophone>Laura
Logic</saxophone>
    <guitar>Someone
else</guitar>
  </band>
</bands>
```

```
<band type="classical">
  <name>Hilliard Ensemble</name>
  <saxophone>Jan
Garbarek</saxophone>
</band>
<band type="progressive">
  <name>Soft Machine</name>
  <organ>Mike Ratledge</organ>
  <bass>Hugh Hopper</bass>
  <drums>Robert Wyatt</drums>
  <saxophone>Elton
Dean</saxophone>
</band>
</bands>
```

Примерен Java код 1/2

```
import org.xml.sax.helpers.XMLReaderFactory;  
import org.xml.sax.XMLReader;  
import org.xml.sax.SAXException;  
import org.xml.sax.Attributes;  
import org.xml.sax.helpers.DefaultHandler;
```

```
public class BandReader extends DefaultHandler  
{  
    public static void main(String[] args) throws Exception  
    {  
        System.out.println("Here we go ...");  
        BandReader readerObj = new BandReader();  
        readerObj.read(args[0]);  
    }  
}
```


Примерен Java код 2/2

```
public void read (String fileName) throws Exception  
{
```

```
    XMLReader readerObj =  
XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAX  
Parser");
```

```
    readerObj.setContentHandler (this);  
    //we registered for interface callback!!  
    readerObj.parse (fileName);
```

```
}  
public void startDocument() throws SAXException  
{ System.out.println("Starting ...");  
}
```

```
public void endDocument() throws SAXException  
{ System.out.println("... Finished");  
}
```

```
public void startElement(String uri, String localName, String qName, Attributes  
atts) throws SAXException  
{ System.out.println("Element is " + qName);  
}
```

```
} XML
```

Результат

- Now let's run it:
 - `java BandReader bands.xml`
- Here's what we see:
 - Here we go ...
 - Starting ...
 - Element is bands
 - Element is band
 - Element is name
 - Element is guitar
 - Element is saxophone
 -
 - Finished

XMLReader (SAX 2.0)

- public interface **XMLReader** – интерфейс за четене на XML документ с използване на callbacks
- Въпреки името си, този интерфейс не разширява стандартния Java **Reader** интерфейс, защото четенето на XML е много по-различно от простото четене на символни данни
- Разрешава приложението да зададе и провери свойства (features, properties) за парсъра, да регистрира обработчици на събития (event handlers) за обработка на документа и да инициира самото парсване

XMLReader (SAX 2.0) спрямо Parser (SAX 1.0)

- Всички SAX интерфейси са *синхронни*:
 - Методите `parse` не връщат управлението преди приключване на парсването, и
 - Четците (readers) трябва да изчакат връщане от `event-handler callback` преди да рапортуват следващото събитие
- Интерфейсът `XMLReader` заменя `SAX 1.0 Parser` (deprecated). `XMLReader` добавя две важни неща към стария `Parser` интерфейс:
 - Стандартен начин за задаване и проверка на свойства (features, properties) за парсъра;
 - Поддръжка на пространства от имена

Интерфейс XMLReader

- Регистрира други обекти за callbacks
 - `void setContentHandler(ContentHandler handler)`
 - `void setDTDHandler(DTDHandler handler)` – разрешава приложението да регистрира DTD event handler:
 - Ако приложението не регистрира DTD handler, всички DTD събития, рапортувани от SAX парсера, се игнорират.
 - Важно: приложението може да регистрира нов (различен) handler в процеса на парсване и от този момент SAX парсерът започна да го използва.
 - `void setErrorHandler(ErrorHandler handler)` – разрешава приложението да регистрира error handler.
 - `void setEntityResolver(EntityResolver resolver) – ...`
- Стартираме парсването чрез метода **`parse()`**
- Когато парсерът прочете значим за него участък от документа, той извиква метод от регистрирания обект
- Парсерът продължава четенето на XML файла след изпълнението на метода

Отново за ContentHandler

- Интерфейс за получаване на основни събития по маркирано съдържание
- Или използваме базовата имплементация – **HandlerBase** (SAX1) или **DefaultHandler** (SAX2) класа и пренаписваме някои методи
- Или имплементираме класа ...
- ... и регистрираме негов екземпляр като **ContentHandler**

```
class myClass implements ContentHandler {
```

```
...
```

```
    myParser.setContentHandler(this) ;
```

```
...
```

ContentHandler имплементации

- **DefaultHandler** (SAX2) – заменя класа **HandlerBase** от SAX1 ;
- Предоставя базови имплементации за всички callbacks на 4 основни обработващи интерфейси в SAX2:
 - [EntityResolver](#)
 - [DTDHandler](#)
 - [ContentHandler](#)
 - [ErrorHandler](#)
- **XMLFilterImpl** – стои между [XMLReader](#) и event handlers на приложението и предава събитията на обработчиците без промяна; негови подкласове могат да пренапишат специфични методи
- **XMLReaderAdapter** - обвива SAX2 [XMLReader](#) и го представя като SAX1 [Parser](#)
- *Кодът и документацията са Public Domain ->*
NO WARRANTY

ContentHandler методи 1/2

- Интерфейсни методи (повечето хвърлят **SAXException**):
 - `void startDocument()` – извикван в началото на документа
 - `void endDocument()`
 - `void startElement(String namespaceURI, String localname, String qName, Attributes attr)`
 - извикван в началото на всеки елемент, с параметри:
 - **Namespace URI** и **local name** – изискват се при стойност на свойството `namespaces` равна на *true* (default), и са опционални при `namespaces == false`; за повече виж <http://www.saxproject.org/namespaces.html>
 - **localName** – локално име (без префикс) ако не се извършва обработка на `Namespace`
 - **qName** - квалифицирано име (с префикс) или празен стринг ако не се използват такива имена
 - **attr** – атрибутите на елемента

ContentHandler методи 2/2

- `void endElement(String name) throws SAXException`
– при край на всеки елемент в XML документа; кореспондира си със събитието `startElement`
- `void characters(char[] ch, int start, int length) throws SAXException` - при четене на символни данни; приложението не трябва да опитва да чете извън обхвата
- `void ignorableWhitespace(char[] ch, int start, int length)` – извикван при ignorable whitespace в съдържанието на елемент. Валидиращите парсъри използват този метод за рапортуване на всяка порция от празни пространства
- `void processingInstruction(String target, String data)` – при инструкция за обработка

SAXFinder пример от Beginning XML 2nd Ed. 1/2

```
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;

public class SaxFinder extends DefaultHandler
{
    private StringBuffer saxophonist = new StringBuffer(); private boolean isSaxophone = false;

    public static void main(String[] args) throws Exception
    {
        System.out.println("Here we go ...");
        SaxFinder readerObj = new SaxFinder();
        readerObj.read(args[0]);
    }

    public void read (String fileName) throws Exception
    {
        XMLReader readerObj =
            XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
        readerObj.setContentHandler (this);
        readerObj.parse (fileName);
    }

    public void startDocument() throws SAXException
    {
        System.out.println("Starting ...");
    }
}
```

SAXFinder пример от Beginning XML 2nd Ed. 2/2

```
public void endDocument() throws SAXException
{
    System.out.println("... Finished");
}

public void startElement(String uri, String localName, String qName, Attributes atts) throws SAXException
{
    if (qName.equals("saxophone"))
    {
        isSaxophone = true;
        saxophonist.setLength(0);
    }
    else
        isSaxophone = false;
}

public void endElement(String uri, String localName, String qName) throws SAXException
{
    if (isSaxophone)
    {
        System.out.println("Saxophonist is " + saxophonist.toString());
        isSaxophone = false;
    }
}

public void characters(char[] chars, int start, int len) throws SAXException
{
    if (isSaxophone)
        saxophonist.append(chars, start, len);
}
}
```

Извличане на атрибути

- `public void startElement(String uri, String localName, String qName, Attributes atts) throws SAXException`
- Четири основни метода за **atts** обекта:
 - `getLength` – връща броя на атрибутите в списъка
 - `getQName` – връща квалифицираното име на атрибут на дадена позиция в списъка (броене от 0).
 - `getValue` – връща стойност на атрибут (определен по име или по позиция от 0 до N-1)
 - `getType` – връща тип на атрибут (определен по име или по позиция от 0 до N-1).

ErrorHandler интерфейс

- Ако SAX приложение трябва да имплементира специфична обработка на грешка, то трябва да:
 - имплементира **ErrorHandler** интерфейса и да регистрира екземпляр в XML reader чрез метода [setErrorHandler](#); тогава парсърът ще рапортува всички грешки през този интерфейс
 - не указва къде е станала грешката
- Три нива на изключения
 - `void error(SAXParseException ex);` – викан при възстановяема грешка
 - `void fatalError(SAXParserException ex);` – викан при невъзстановяема грешка
 - `void warning(SAXParserException ex);`

Locator интерфейс

- Парсърът може да предостави (но не непременно!) обект `Locator` с цел използването му в обработващите събитията методи за намиране къде сме в документа, напр. при трасиране на грешки. Асоциира SAX събитие с локация в документа. Регистрира се чрез:

`setDocumentLocator(Locator loc)`

- Ако имаме регистриран клас, имплементиращ `Locator` интерфейс, парсърът може да информира за ред и позиция на грешката и др.

Методи на Locator

`int getLineNumber() ;` - връща номер на ред

`int getColumnNumber() ;` - връща номер на колона

`String getSystemId() ;` - име на файл с парсван XML документ

`String getPublicId() ;` - публичен идентификатор на документа

Пример от “Beginning XML”

```
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax Locator;
import org.xml.sax.SAXParseException;
```

```
public class BandValidator extends DefaultHandler
{
    private StringBuffer saxophonist = new StringBuffer();
    private boolean isSaxophone = false;
    private StringBuffer bandName = new StringBuffer();
    private boolean isName = false;
    private String bandType = new String();
    private Locator locatorObj;

    public static void main(String[] args) throws Exception
    {
        System.out.println("Here we go ...");
        BandValidator readerObj = new BandValidator();
        readerObj.read(args[0]);
    }
}
```


public void read (String fileName) throws Exception

```
{ XMLReader readerObj =  
XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");  
try {  
    readerObj.setFeature("http://xml.org/sax/features/validation", true);  
} catch (SAXException e){  
    System.err.println("Cannot activate validation");  
}  
readerObj.setContentHandler (this);  
readerObj.setErrorHandler (this);  
readerObj.parse (fileName);  
}
```

```
public void startDocument() throws SAXException  
{    System.out.println("Starting ..."); }
```

```
public void endDocument() throws SAXException  
{    System.out.println("... Finished"); }
```

```
public void startElement(String uri, String localName, String qName, Attributes atts) throws  
SAXException
```

```
{    if (qName.equals("band"))  
    { bandType = atts.getValue("type");  
      if (bandType == null)  
          { if (locatorObj != null)
```

```
              System.err.println ("Error in " + locatorObj.getSystemId() + " at line " +  
locatorObj.getLineNumber() + ", column " + locatorObj.getColumnNumber());  
              throw new SAXException("Band type not specified");
```

```
          }
```

```
XML
```

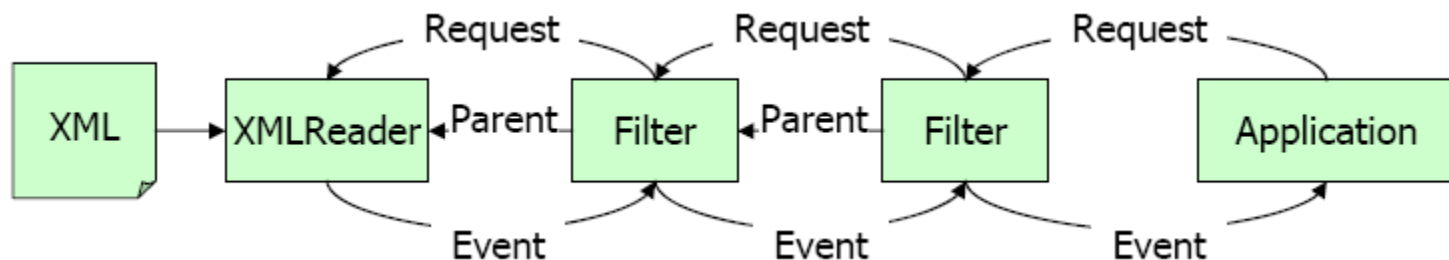
```
}
```

SAX & StAX

*Трябва да проверим
дали обектът Locator
не е Null - ако
парсерът не
поддържа Locator
обект.*

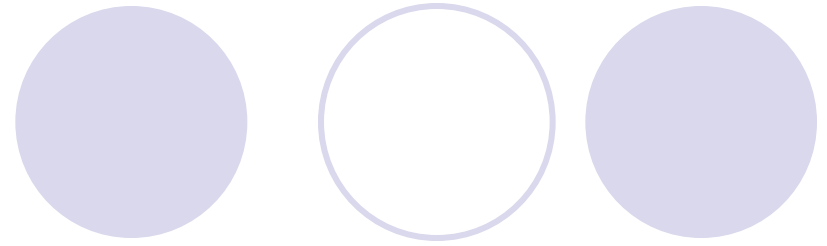
XMLFilter 1/2

- Наследява XMLReader
- Работи по събития
- Конвейер от събития:



- Регистриране на предшестващ филтър
 - `setParent(XMLReader)`
- За удобство: XMLFilterImpl
 - Имплементира XMLFilter, ContentHandler, ErrorHandler
 - Предава събитията без промяна

XMLFilter 2/2



- Трансформации (при запазване на структурата)
 - Преименоване на пространства, елементи, ...
 - Трансформации на стойности на атрибути и др.

```
public class ElementFilter extends XMLFilterImpl {...
    public ElementFilter(XMLReader parent, String old, String new) {
        super(parent);
        ...}
    public void startElement(... String name ...) {
        if (name.equals(old))
            super.startElement(... new ...);
        else
            super.startElement(... name ...);
        ...}
}
```

DTDHandler интерфейс

- Предоставя callback методи за известяване за DTD събития
- Ако SAX приложението трябва да информира за нотации и единици (entities), то имплементира този интерфейс и го регистрира негов екземпляр в парсъра чрез метода **setDTDHandler**.

Методите накратко:

- void [notationDecl](#)([String](#) name, [String](#) publicId, [String](#) systemId)
Известяван за декларация на нотация
- void [unparsedEntityDecl](#)([String](#) name, [String](#) publicId, [String](#) systemId, [String](#) notationName)
Известяван за декларация на unparsed entity

Кога да ползваме SAX (Simple API for XML) 1/2

- Типична употреба:
 - Наш нов клас разширява `DefaultContentHandler`
 - Имплементираме callback методи (напр. `startElement, ...`)
- Обработка при парсването – само последното събитие е в паметта
- За комплексни структури:
 - Нужда от променливи на състоянието
 - Тежка модуларизация

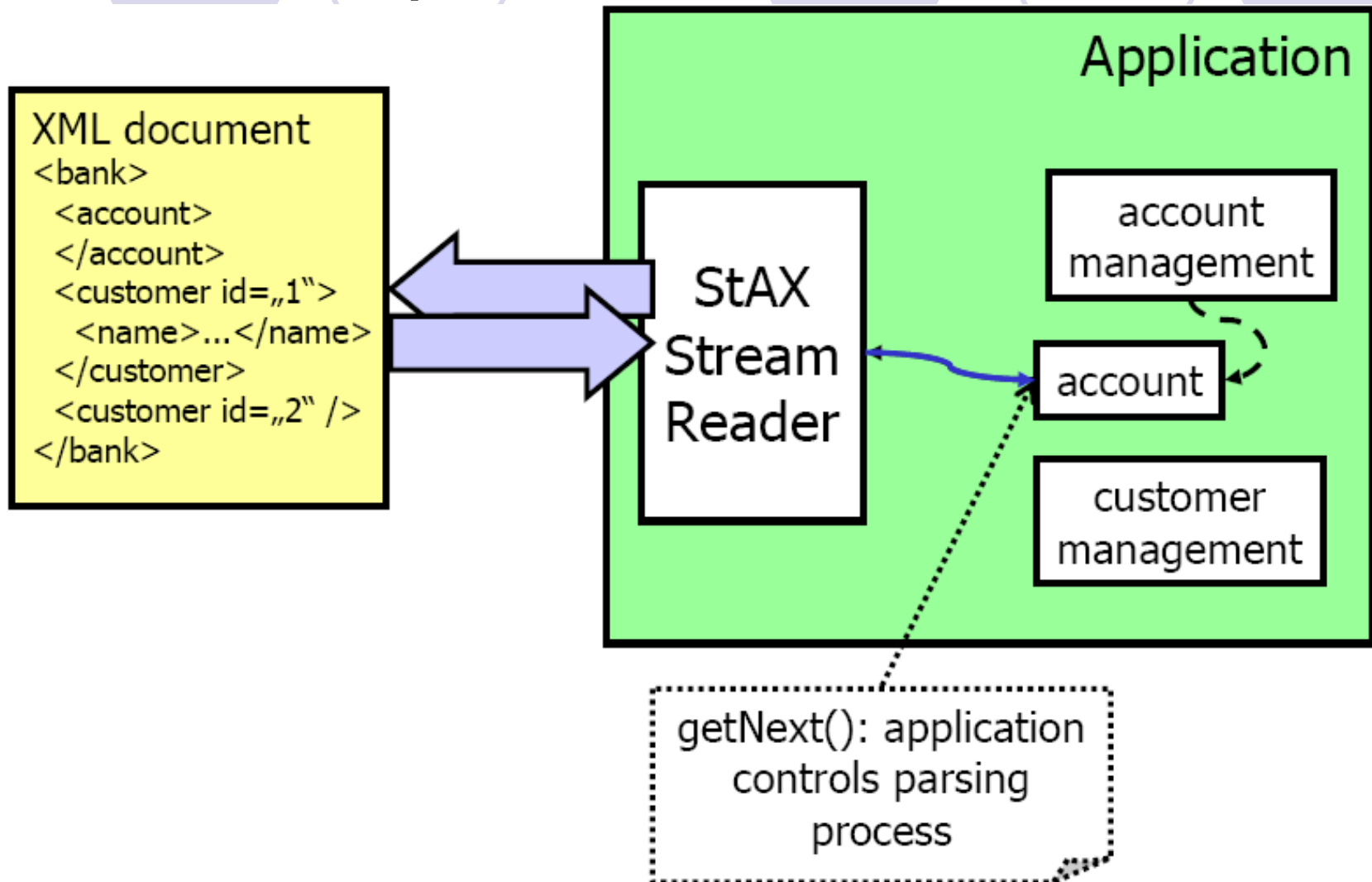
Кога да ползваме SAX (Simple API for XML) 2/2

- Идеален за прости операции над XML файлове
 - Като четене на елементи и атрибути
- Подходящ за много големи XML документи (спрямо DOM)
- Не е удобен за манипулиране на XML структурата
- Не е проектиран за генериране на XML – само за четене!

Streaming API for XML (StAX)

- Цел
 - Лесен като DOM
 - Бърз като SAX
 - Икономичен спрямо паметта като SAX
 - API за четене и запис
- Приложението управлява парсъра
 - Лесно е за програмиране
 - Достъп до parse-events при нужда (за работа като event driven)
 - Pull-parsing (вместо Push parsing a la SAX)
- StAX SE е част от Java 1.6+ API

Работен процес на StAX



Два приложни StAX интерфейса 1/3

- **Cursor API**: по-директен и ефикасен:
 - XMLStreamReader, XMLStreamWriter
 - Директен достъп до данните
 - Много единични методи
 - По-малък и ефективен код, по-добра производителност
 - **Iterator API** (конвейерна обработка): филтриране, потоци, по-добра поддръжка, по-четим код и модулен дизайн
 - XMLEventReader, XMLEventWriter
 - Данните се съхраняват в XMLEvent immutable objects
- XML – достъпни при следващото събитие

Два приложни StAX интерфейса 2/3

- **Cursor API**: представлява курсор, с който можете да се разхождате по XML документ от началото до края. Курсорът сочи една конструкция в даден момент и винаги се движи напред.
- Два основни курсор интерфейси: **XMLStreamReader** и **XMLStreamWriter**.
- XMLStreamReader включва методи за достъп до цялата възможна информация на XML информационния модел, вкл. кодиране на документа, имената на елементите, атрибути, пространства от имена, текст, стартиращи тагове, коментари, инструкции за обработка, граници на документи и др.

Два приложни StAX интерфейса 3/3

- **Iterator API**: представя XML документния поток като набор от дискретни обекти-събития. Тези събития се извличат от приложението чрез парсъра в реда, в който се четат във входния XML документ.
- Базовият итераторен интерфейс се нарича **XMLEvent**.
- Основният парсърен интерфейс за четене на събития е **XMLEventReader** и основният интерфейс за писане на събития е **XMLEventWriter**.
- **XMLEventReader** имплементира **java.util.Iterator**

Създаване на StAX reader

- При работа със StAX:
 - `import javax.xml.stream.*`
- Както при SAX и DOM, първо се получава фабрика чрез извикване на статичен метод
 - `XMLInputFactory factory = XMLInputFactory.newInstance();`
- За фабриката можем да задаваме различни свойства
 - `factory.setProperty("javax.xml.stream.isValidating", "true");`
- XML се подава през `InputStream` или `Reader`
 - `FileReader fileReader = new FileReader("somefile.xml");`
 - Може да изхвърли `FileNotFoundException`
- Едва сега чрез фабриката създаваме `XMLStreamReader`
 - `XMLStreamReader reader = factory.createXMLStreamReader(fileReader);`
 - Може да изхвърли `XMLStreamException`

Използване на StAX парсър

- StaAX парсърът (reader) се държи като **Iterator**
 - Методът **boolean hasNext()** указва дали има друго събитие за четене
 - **int next()** прочита следващото събитие
 - Връщаният резултат **int** задава типа на това събитие
 - Възможни стойности са **START_ELEMENT**, **END_ELEMENT**, **ATTRIBUTE**, **CHARACTERS** (content), **COMMENT**, **SPACE**, **END**
- След **next()**, парсърът е стигнал до „текущ елемент“ и може да бъде разпитван относно него
 - Например, **getLocalName()** връща името на текущия елемент
- Важно: парсърът се движи само напред; лесно можем да пропуснем ценна информация
- По-лесно е да се анализира документа, ако знаем неговата структура
 - Валидиращ парсер може да провери структурата спрямо DTD
 - **isValidating** е свойство на Java StAX парсера

Използване на `getLocalName()`

- `getLocalName()` връща името на маркера (тага) като `String`
- понеже не можем да ползваме `switch` за `String`, трябва да го сравняваме отделно чрез `equals`:
 - `String name = reader.getLocalName();`
 `if (name.equals(someTag)) { ... }`
 `else if (name.equals(someOtherTag)) { ... }`
 `else if (name.equals(someOtherTag)) { ... }`
 `...`

Константи

- **int next()** премества към следващото събитие и връща **int** за указване на типа на събитието:
 - **START_DOCUMENT**
 - **END_DOCUMENT**
 - **START_ELEMENT**
 - **END_ELEMENT**
 - **ATTRIBUTE**
 - **CHARACTERS**
 - **COMMENT**
 - **SPACE**
 - **DTD**
 - **PROCESSING_INSTRUCTION**
 - **NAMESPACE**
 - **CDATA**
 - **ENTITY_REFERENCE**
- Тези константи са дефинирани в **XMLStreamReader** обекта

Методи

- Съществуват множество методи, дефинирани в **XMLStreamReader**; някои от тях са:
 - **boolean hasNext()** – **true** ако има друго събитие
 - **int next()** – придвижва се до следващото събитие и връща типа му (**int**)
 - **int nextTag()** – придвижва до start или end маркер и връща типа му
 - **getLocalName()** – взима името на текущия елемент или entity reference
 - **getAttributeCount()** – взима броя на атрибутите на текущия елемент
 - **getAttributeLocalName(*index*)** – името на атрибута
 - **getAttributeValue(*index*)** – стойността на атрибута
 - **getElementText()** – връща текста на **START_ELEMENT**; след извикване, текущ елемент става **END_ELEMENT**
 - **getText()** – връща текстовата стойност на **CHARACTERS**, **COMMENT**, **ENTITY_REFERENCE**, **CDATA**, **SPACE**, or **DTD**

Създаване на StAX writer

- Всичко за StAX е в `javax.xml.stream`
 - `import javax.xml.stream.*`
- Получаваме метод-фабрика:
 - `XMLOutputFactory factory = XMLOutputFactory.newInstance();`
- За запис в XML файл, използваме `OutputStream` или `Writer`
 - `FileWriter fileWriter = new FileWriter("somefile.xml");`
 - Може да изхвърли `IOException`
- Създаваме writer за XML
 - `XMLStreamWriter writer = factory.createXMLStreamWriter(fileWriter);`
 - Може да изхвърли `XMLStreamException`

Методи

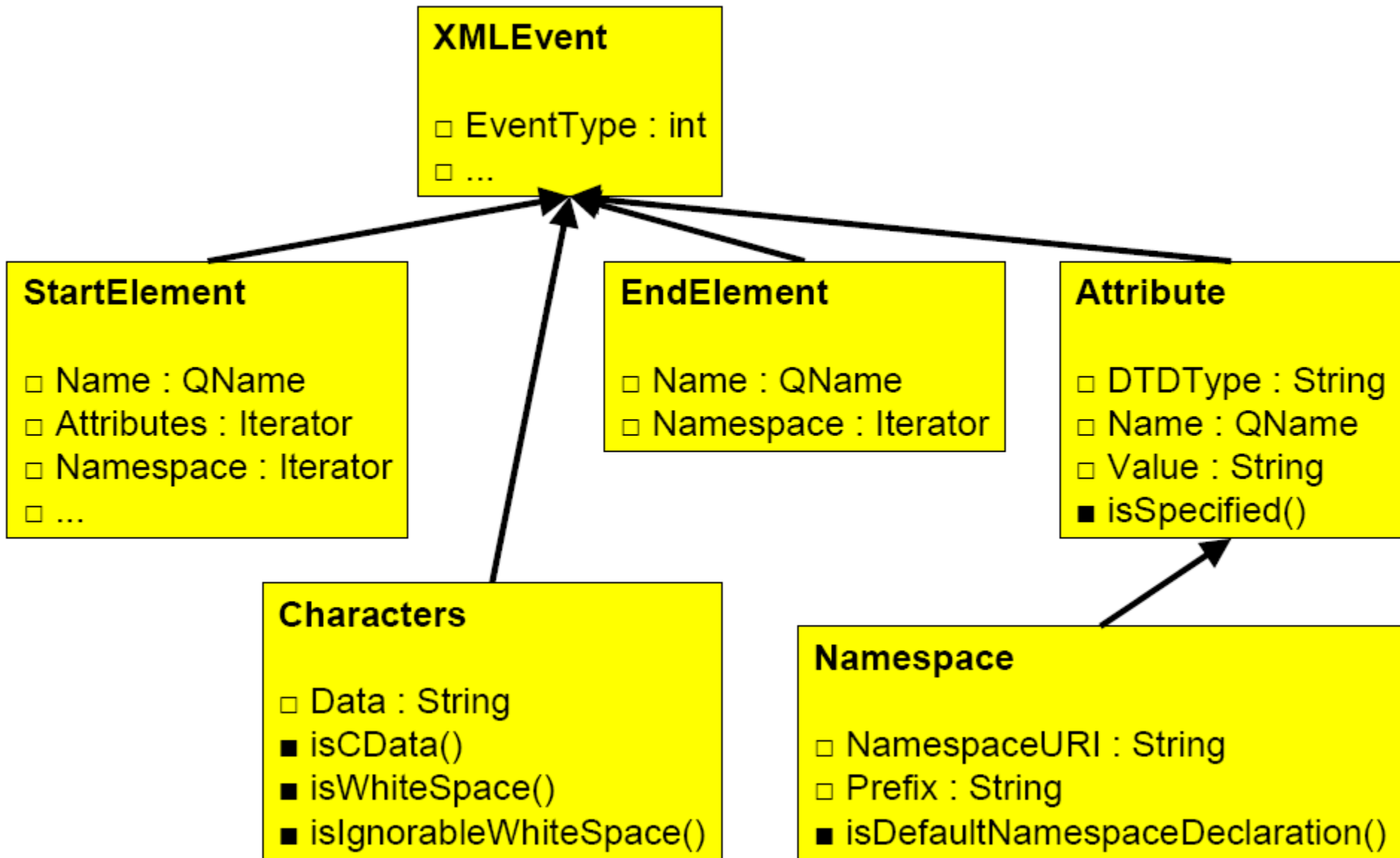
- Дефинирани са множество методи за `XMLStreamWriter`, напр.:
 - `writeStartDocument(version)` – записва XML хедър (оглавление)
 - `writeStartElement(name)` – записва стартов маркер
 - `writeAttribute(name, value)` – записва атрибут
 - `writeCharacters(value)` – записва текст, кодирайли символи като `<` `>` и `&`
 - `writeComment(value)` – записва коментар
 - `writeDTD(value)` – записва цялата DTD дефиниция
 - `writeEndElement()` – записва краен маркер
 - `flush()` – принуждава записа на буфериран изход
 - `close()` – затваря `XMLStreamWriter`

Преглед на Iterator API – Read

XMLStreamReader

- `java.util.Iterator`:
 - `public boolean hasNext()`;
 - `public Object next()`;
- Следващо събитие с преместване върху него:
 - `public XMLEvent nextEvent()`
throws `XMLStreamException`;
- Следващо събитие без преместване
 - `public XMLEvent peek()` throws `XMLStreamException`;

Йерархия на класа XMLEvent



Преглед на Iterator API – write

XMLStreamWriter

- **public void add (XMLEvent e)**
throws XMLStreamException;
- За всички writers
 - **public void flush() throws XMLStreamException;**
- За всички readers и writers
 - **public void close() throws XMLStreamException;**

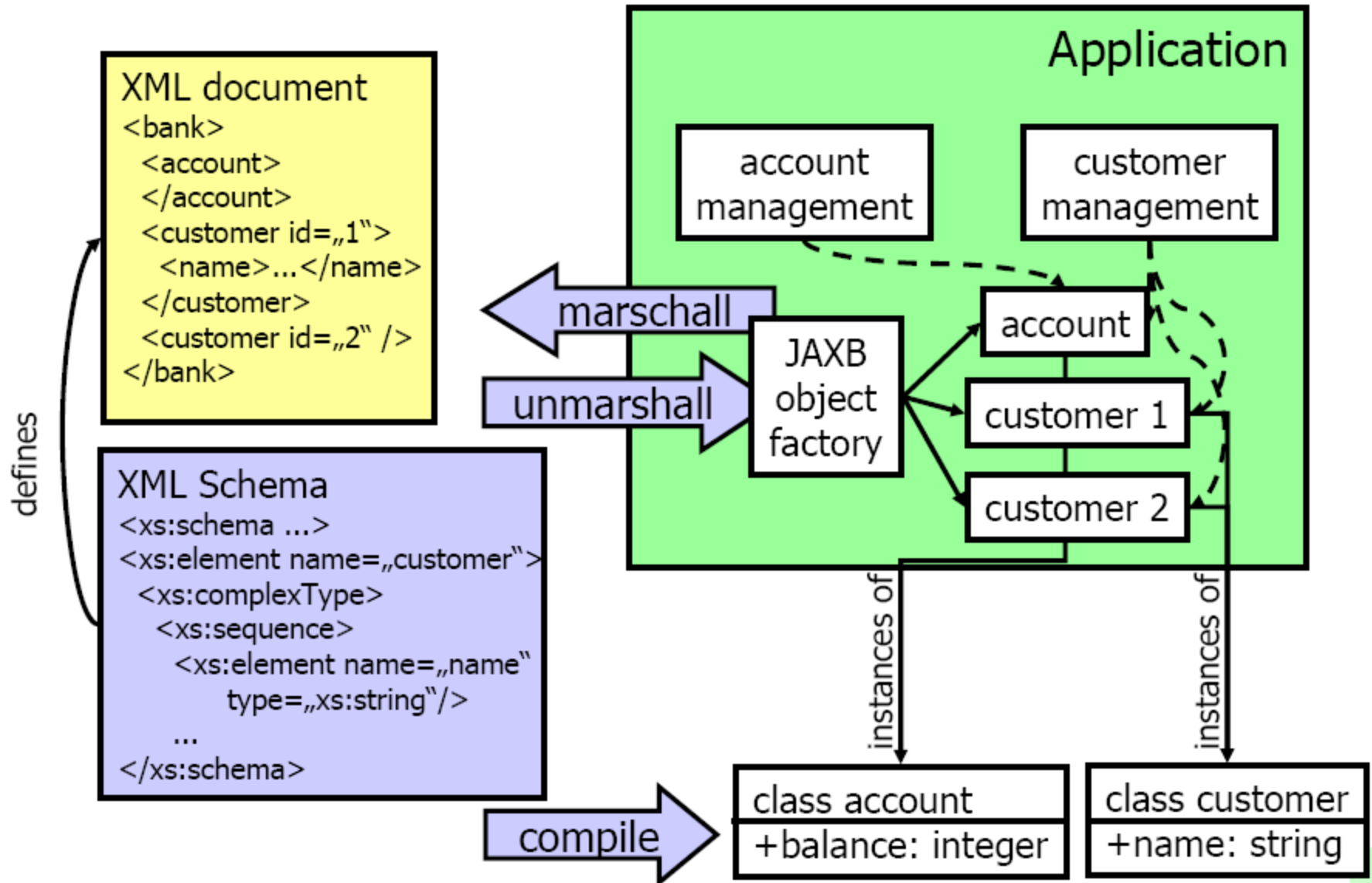
StAX - обобщение

- Предимства на StAX:
 - По-лесен за ползване от SAX, особено поради липсата на callbacks
 - Може да пише в XML файлове, како и да чете от тях
- Недостатъци на StAX
 - Налага ползването на **if-then-else** за разпознаване какво се парсва
 - Както при SAX, движението е само напред
- Сравнение с DOM:
 - StAX е по-бърз, по-ефикасен и по-прост
 - DOM позволява манипулирането на дърво в паметта

JAXB (Java Architecture for XML Binding)

- Java Architecture for XML Binding (JAXB) разрешава на Java проектантите да съпоставят Java класове на XML.
- JAXB адресира два въпроса: възможността да се разполагат (marshal) Java обекти в XML документи и обратното (unmarshal) XML обратно до Java бекти. Т.е., чрез JAXB данни могат да се запазват и извличат от паметта в произволен XML формат без имплементиране на XML зареждане.
- JAXB особено полезна, когато спецификацията е сложна и променяща се.
- JAXB е API в Java EE платформата и част от Java Web Services Development Pack (JWS DP).

Работен процес на JAXB - Java Architecture for XML Binding



Повече за JAXB

- [*java.sun.com/developer/technicalArticles/WebServices/**jaxb**/*](http://java.sun.com/developer/technicalArticles/WebServices/jaxb/)
- [*java.sun.com/developer/technicalArticles/xml/**jaxb**/*](http://java.sun.com/developer/technicalArticles/xml/jaxb/)
- [*en.wikipedia.org/wiki/**JAXB***](http://en.wikipedia.org/wiki/JAXB)

DOM спрямо StAX

DOM	StAX
whole document in memory (tree)	only recent part (event) in memory
first read whole document	read and process data at one time
free navigation in tree	serial processing
high memory usage	low memory usage
computing intensive	high throughput
maximum flexibility	processing of well-known data structures

SAX спрямо StAX

SAX	StAX
Parser controls flow	Application controls flow
Parser sends data, whether applicator is ready or not	Application asks explicitly for data
Push parsing	Pull parsing
low memory usage	low memory usage
only read	read and write

Заключение: StAX, SAX и DOM

	StAX	SAX	DOM
API Type	pull	push	tree
Usage	easy	complex	easy
XPath-Support	no	no	yes
Efficiency (Memory, CPU)	good	good	bad
only forward parsing	yes	yes	no
XML read	yes	yes	yes
XML write	yes	no	yes
create, modify, delete	no	no	yes