

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

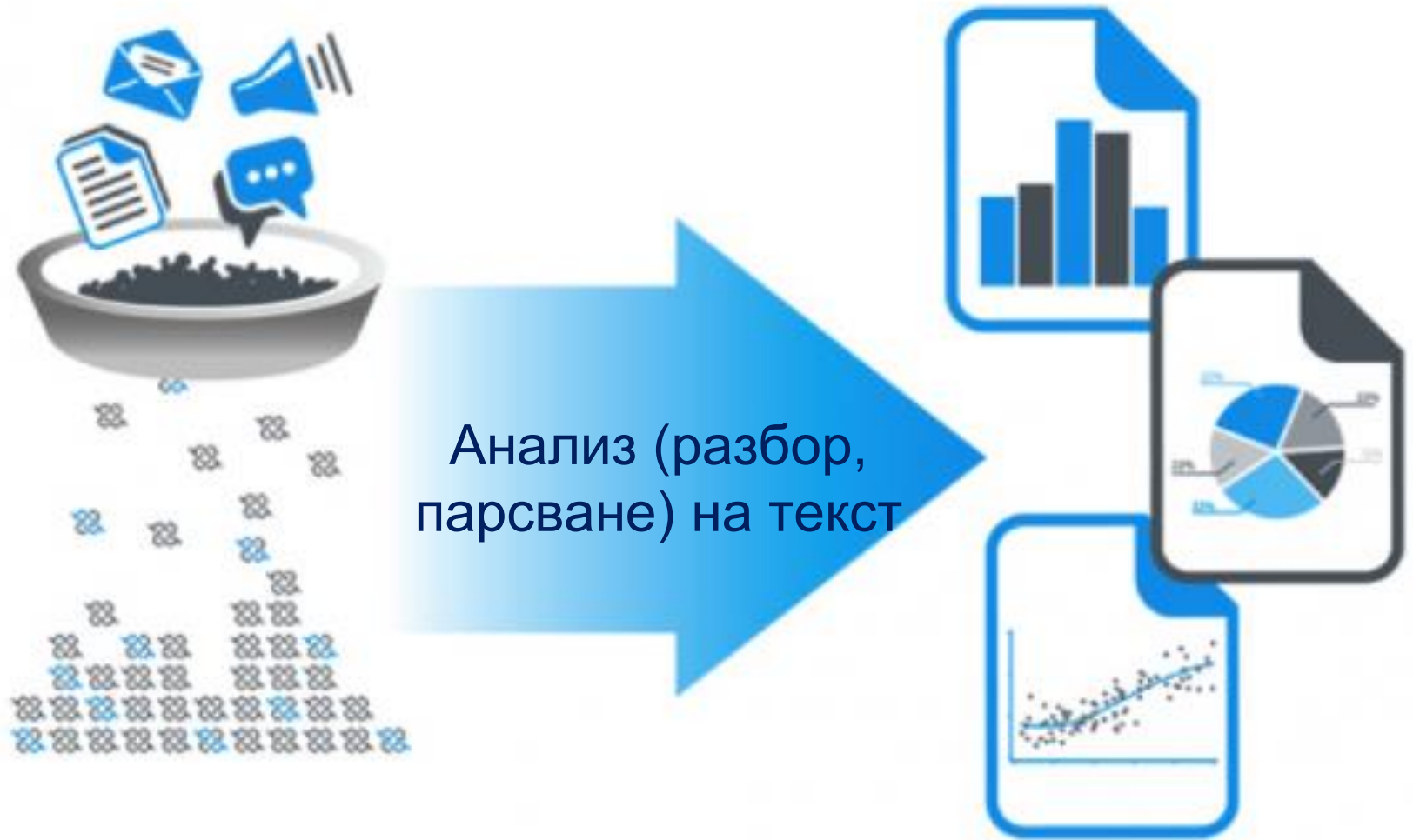
Предимства на SAX
SAX интерфейси

Използване

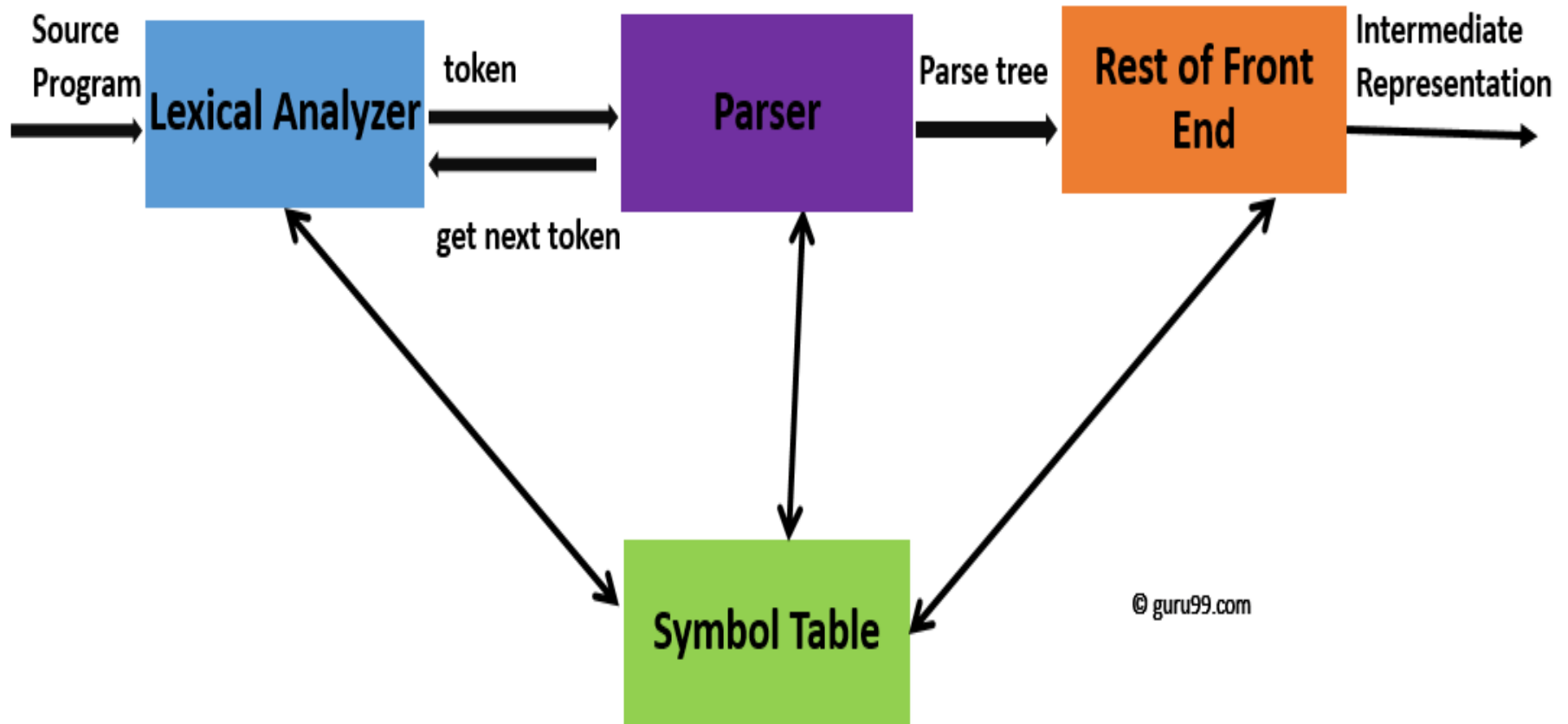
StAX

DOM, StAX и SAX





Процес на анализ (разбор, парсване) на текст



Използване на XML Parser

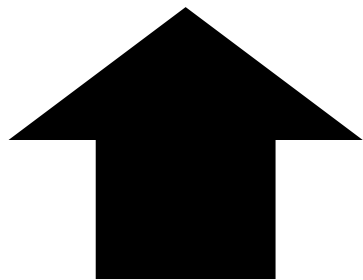
- Цел и приложение
- Три основни стъпки в използването на XML парсер:
 - Създаване на парсер-обект
 - Предаване на XML документа на парсера
 - Обработка на резултатите
- Принципно, генерацията на XML или друг формат е извън обхвата на парсерите

Типове XML парсъри

Съществуват различни групи от типове:

- Валидиращи спрямо невалидиращи парсери
- Парсери, използващи определена приложна рамка (API):
 - Парсери, използващи Document Object Model (DOM)
 - Парсери, използващи Simple API for XML (SAX)
 - Парсери, използващи Streaming API for XML (StAX)
- Парсери, написани на конкретен език (Java, C++, Perl, etc.) без използване на определен API

Начини на обработка на XML съдържание



Зареждане на целия
документ в паметта



Последователна
обработка на документа

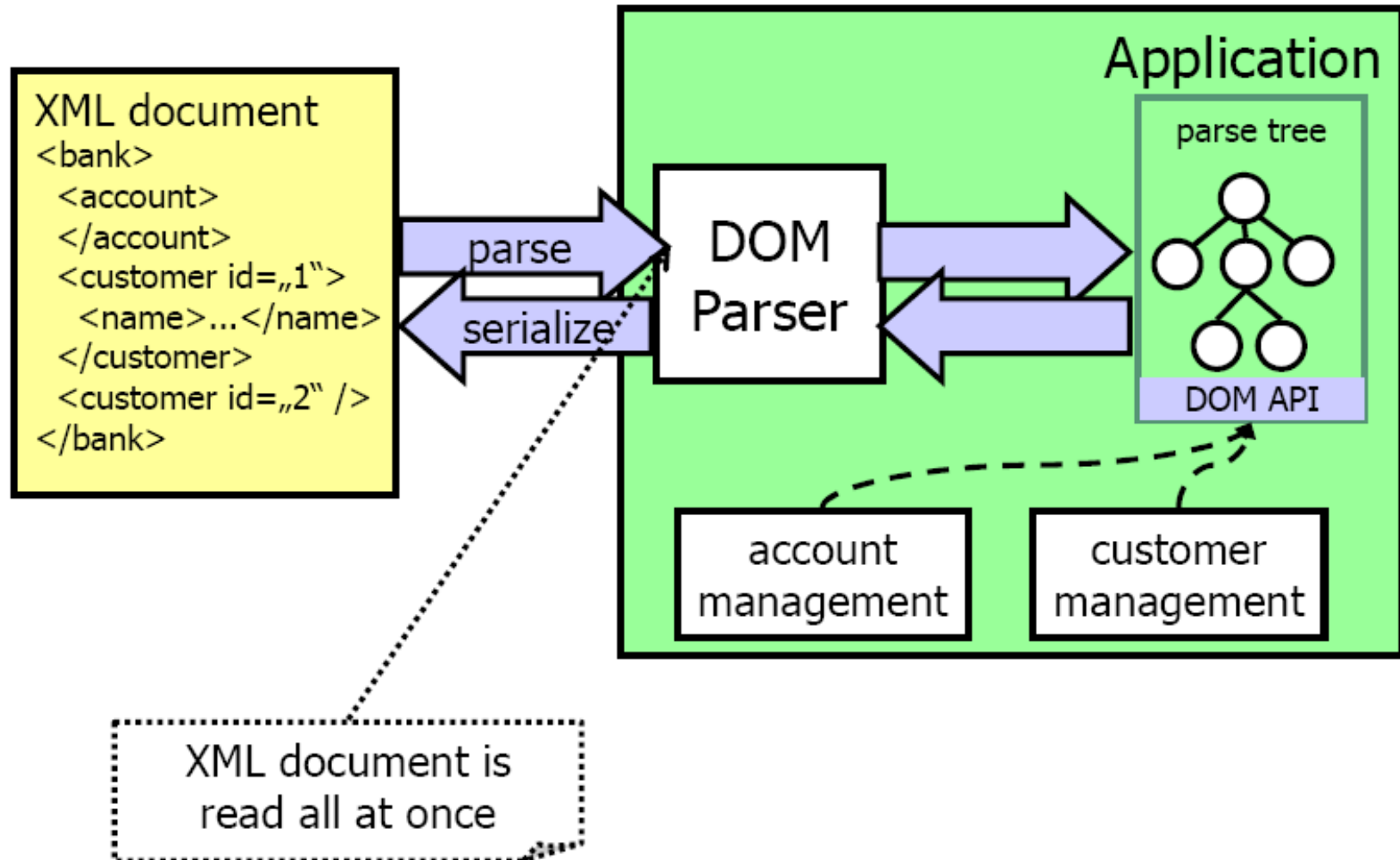
Последователна обработка

- Начини за последователна обработка
 - Обработка, базирана на събития
 - Обработка, базирана на съдържание
- Недостатъци на последователната обработка
 - Невъзможност за връщане назад при четене
 - Евентуална необходимост от повторно четене на документа при необходимост от валидация – напр. за намирането на съответствие между намерен IDREF и последващ ID – *как да я избегнем?*

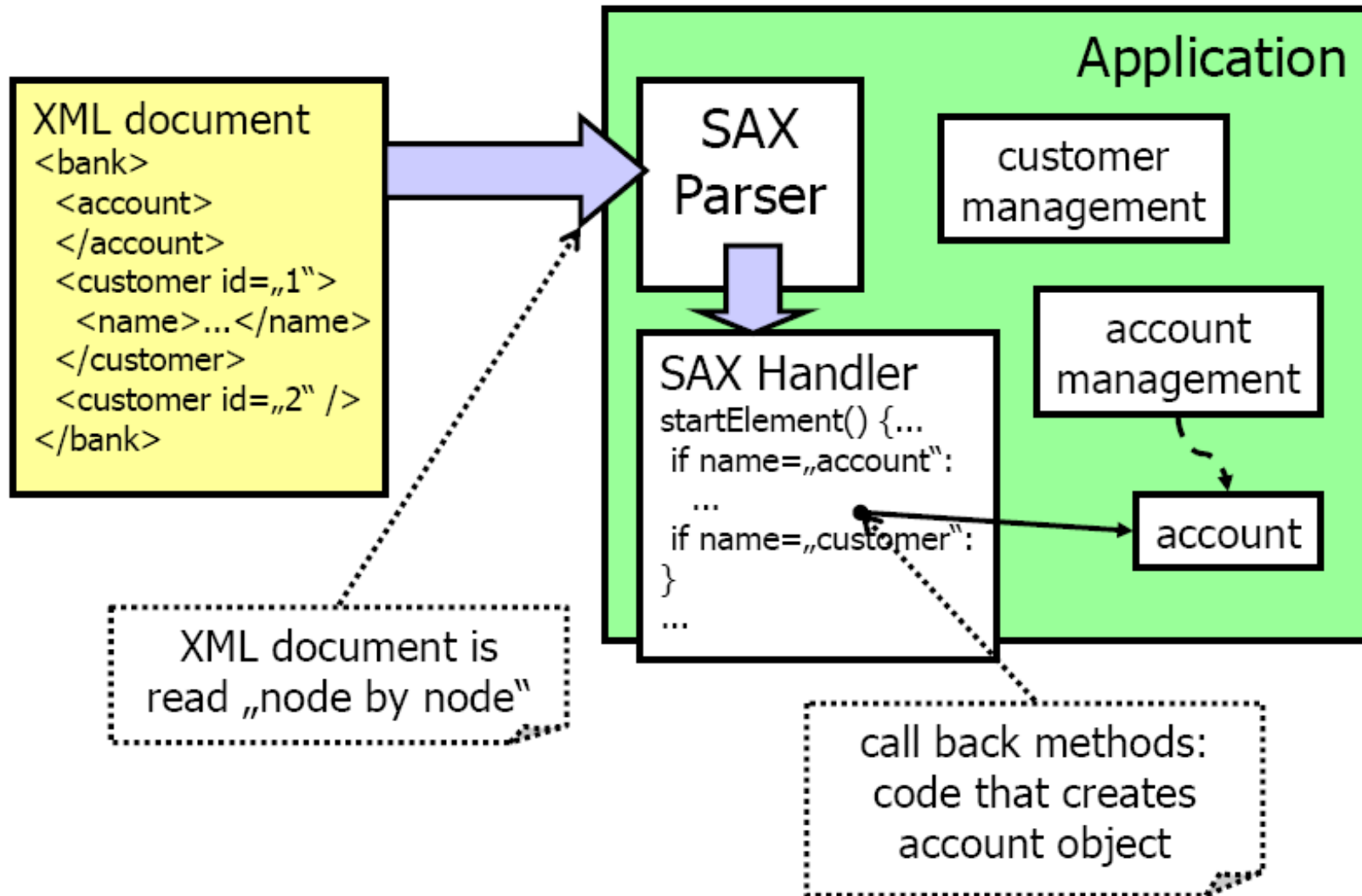
Разбор (парсване) на XML съдържание

- 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 pull)
- 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



Работен процес на SAX



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

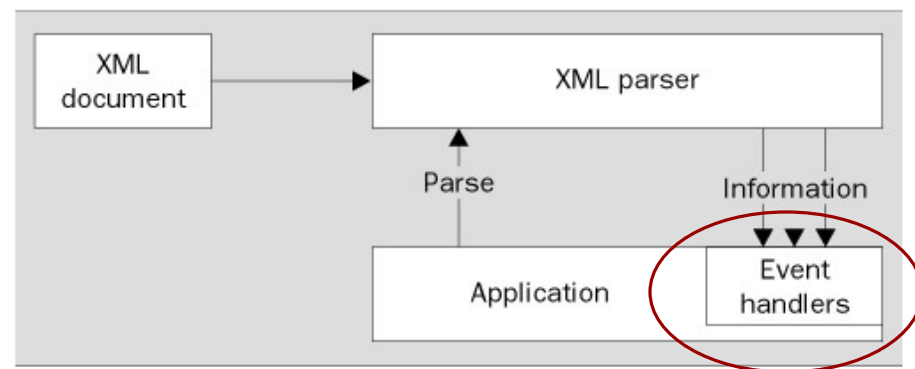
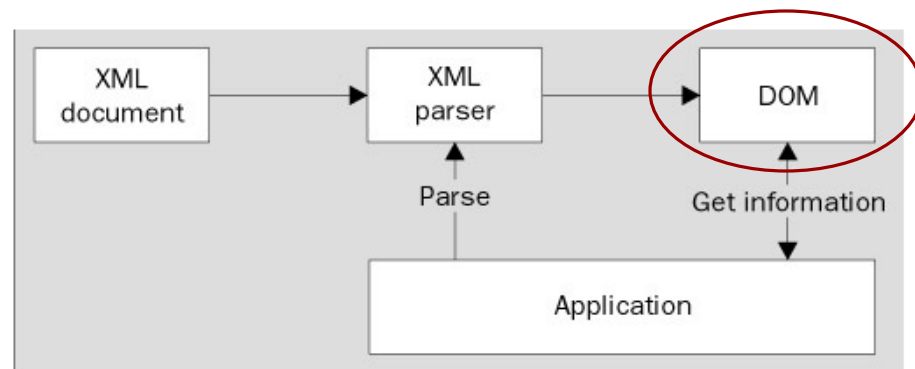
- За **по-ефикасен** анализ на големи *XML* документи
- Цел: да реши проблема на **DOM** – създаването на масивно дърво на документа в паметта, преди да започнем работа с него, т.е. така да спести:

- памет

- време

- Решение:

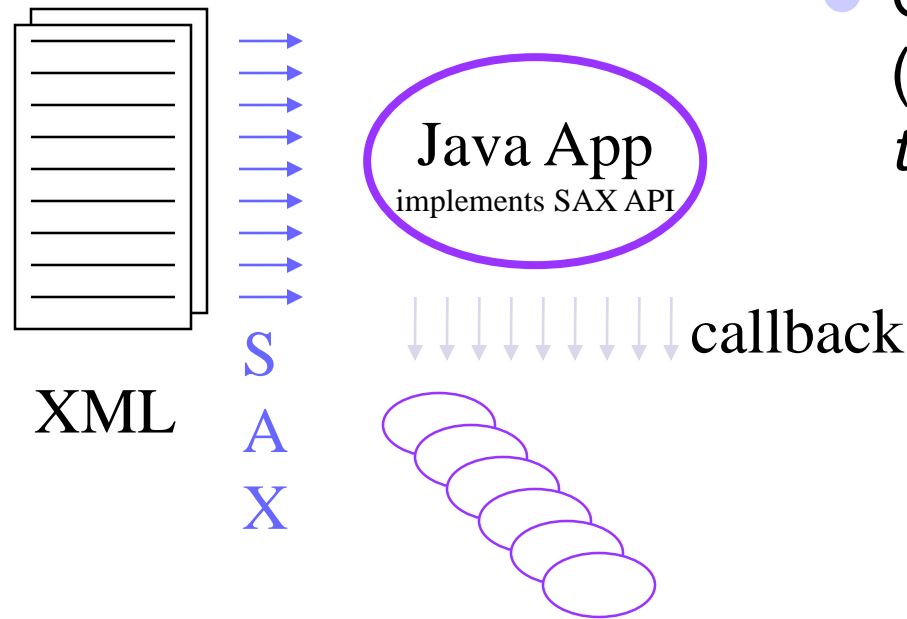
SAX



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 - 1998
- SAX 2.0, 2000 - подобрява се поддръжката на пространства с имена + строго придържане към XML спецификацията
- SAX 2.0.2, 2004 - специфициран е като множество от Java интерфейси (www.saxproject.org, <http://sourceforge.net/projects/sax>)
- Голям брой SAX compliant парсъри — повечето отворени:
 - <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 парсера
 - XML файлът се прочита ред по ред
 - Парсерът 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>`
- Парсер, 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).

SAX събития

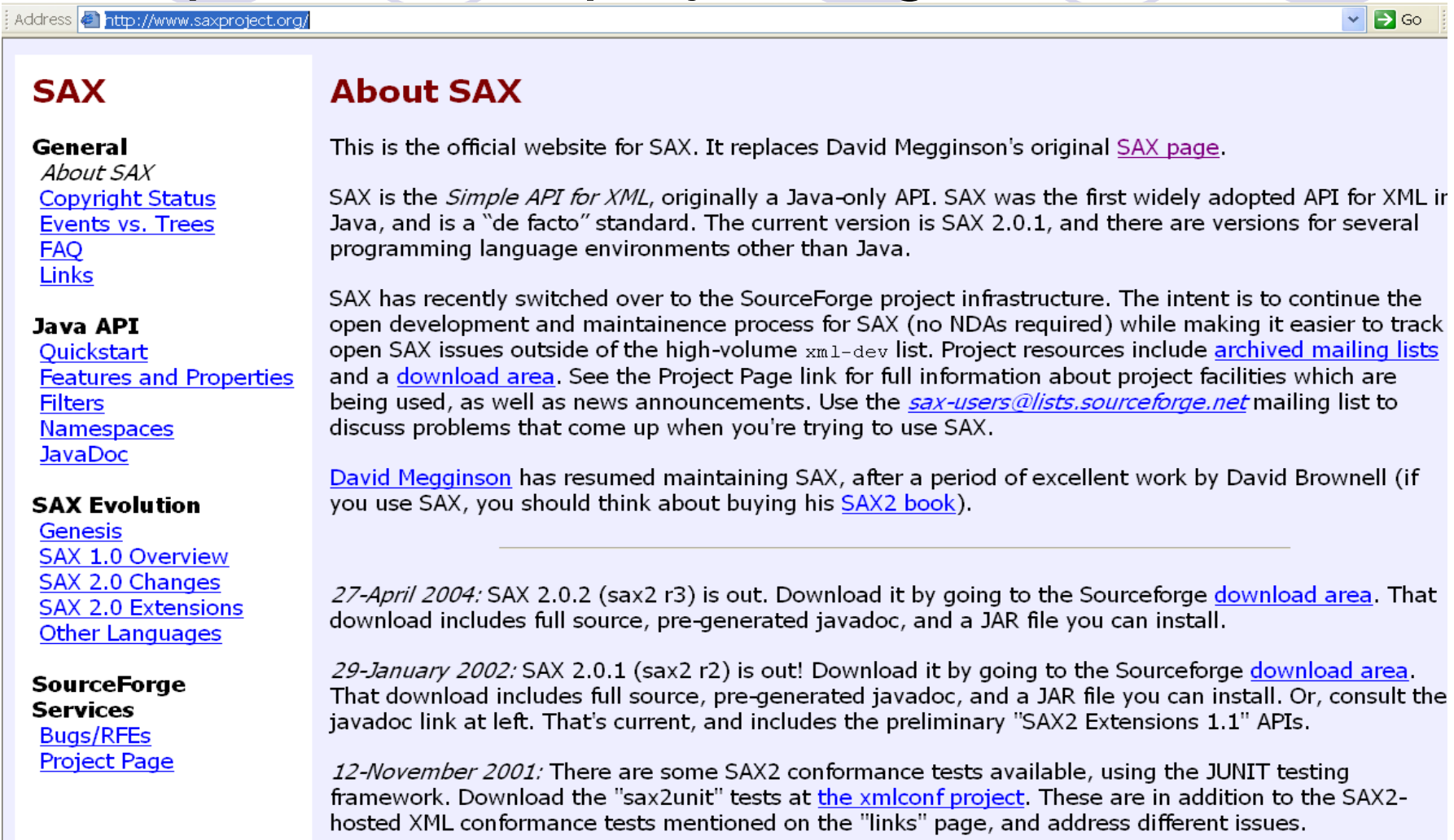
- **startDocument**
 - Начало на обработката и възникване на първото събитие
- **endDocument**
 - Край на обработката и възникване на последното събитие
- **startElement**
 - Достигане до отварящ таг на елемент (<document>)
- **endElement**
 - Достигане до затварящ таг на елемент (</document>)
- **characters**
 - Достигане до низ от символи ("Sample text")

- **processingInstruction**
 - Достигане до инструкция за обработка (xml-stylesheet href="web.xsl" type="text/xml")
- **ignorableWhitespace**
 - Достигане до секция с празни символи, които не са част от документа
- **skippedEntry**
 - Възниква, когато външен обект е пропуснат
- **setDocumentLocator**
 - Позволява да подате на парсера Locator обект на приложението

DefaultHandler Class

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 'SAX' header and several categories: 'General' (with links for 'About SAX', 'Copyright Status', 'Events vs. Trees', 'FAQ', and 'Links'), 'Java API' (with links for 'Quickstart', 'Features and Properties', 'Filters', 'Namespaces', and 'JavaDoc'), 'SAX Evolution' (with links for 'Genesis', 'SAX 1.0 Overview', 'SAX 2.0 Changes', 'SAX 2.0 Extensions', and 'Other Languages'), and 'SourceForge Services' (with links for 'Bugs/RFEs' and 'Project Page'). The main content area is titled 'About SAX' and contains three paragraphs of text. The first paragraph states that this is the official website for SAX, replacing David Megginson's original page. The second paragraph explains that SAX is the 'Simple API for XML', originally a Java-only API, and is a 'de facto' standard. The third paragraph mentions that SAX has recently switched to the SourceForge project infrastructure. Below this, there is a paragraph about David Megginson resuming maintenance of SAX. Further down, there are two dated announcements: one from April 2004 about SAX 2.0.2 and another from January 2002 about SAX 2.0.1. The final paragraph mentions SAX2 conformance tests available using the JUnit testing framework.

Address <http://www.saxproject.org/> Go

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.

SAX наръчник

docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html



The Java™ Tutorials

[Hide TOC](#)

[Simple API for XML](#)

[When to Use SAX](#)

[Parsing an XML File](#)

[Using SAX](#)

[Implementing SAX](#)

[Validation](#)

[Handling Lexical Events](#)

[Using the DTDHandler and](#)

[EntityResolver](#)

[Further Information](#)

[« Previous](#) • [Trail](#) • [Next »](#)

[Home Page](#) > [Java API for XML Processing \(JAXP\)](#) > [Simple API for XML](#)

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.

See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Parsing an XML File Using SAX

In real-life applications, you will want to use the SAX parser to process XML data and do something useful with it. This section examines an example JAXP program, `SAXLocalNameCount`, that counts the number of elements using only the `localName` component of the element, in an XML document. Namespace names are ignored for simplicity. This example also shows how to use a `SAX ErrorHandler`.

Note - After you have downloaded and installed the sources of the JAXP API from the [JAXP download area](#), the sample program for this example is found in the directory `install-dir/jaxp-1_4_2-release-date/samples/sax`. The XML files it interacts with are found in `install-dir/jaxp-1_4_2-release-date/samples/data`.

Creating the Skeleton

The `SAXLocalNameCount` program is created in a file named `SAXLocalNameCount.java`.

```
public class SAXLocalNameCount {
    static public void main(String[] args) {
        // ...
    }
}
```

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 парсерът 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>
```

```
    <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 парсър,
 - 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 парсера;
 - 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 интерфейса **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).

четене на XML файл 1-5

<People>

<Person bornDate="1874-11-30"
diedDate="1965-01-24">

<Name>Winston
Churchill</Name>

<Description>
Winston Churchill was a
mid-20th ...

</Description>
</Person>

<Person bornDate="1917-11-19"
diedDate="1984-10-31">

<Name>Indira Gandhi</Name>
<Description>

Indira Gandhi was India's
first female...

</Description>

</Person>

<Person bornDate="1917-05-29"
diedDate="1963-11-22">

<Name>John F. Kennedy</Name>
<Description>

JFK, as he was
affectionately known,...

</Description>

</Person>

</People>

четене на XML файл 2-5

```
public class SaxParser1 extends DefaultHandler {
    public void startDocument( ) throws SAXException {
        System.out.println( "SAX Event: START DOCUMENT" );
    }
    public void endDocument( ) throws SAXException {
        System.out.println( "SAX Event: END DOCUMENT" );
    }
    public void startElement(String namespaceURI, String
localName, String qName, Attributes attr) throws SAXException
{
        System.out.println( "SAX Event: START ELEMENT[ " +
localName + " ]");
    }
    public void endElement(String namespaceURI, String
localName, String qName) throws SAXException {
        System.out.println( "SAX Event: END ELEMENT[ " + localName
XML" ]" );
    }
}
```

четене на XML файл 3-5

```
public void characters(char[] ch, int start,
int length ) throws SAXException {
    System.out.print( "SAX Event: CHARACTERS[ "
);
    try {
        OutputStreamWriter output = new
OutputStreamWriter(System.out);
        output.write( ch,start,length );
        output.flush();
    } catch ( Exception e ) {
        e.printStackTrace();
    }
    System.out.println( " ]" );
}
```

четене на XML файл 4-5

```
public static void main( String[] argv ) {  
    String inputFile = argv[0];  
    System.out.println( "Processing \"" + inputFile  
+ "\" ." );  
    System.out.println( "SAX Events:" );  
    try {  
        XMLReader reader =  
XMLReaderFactory.createXMLReader();  
        reader.setContentHandler( new SaxParser1() );  
        reader.parse(new InputSource(new  
FileReader(inputFile)));  
    } catch ( Exception e ) {  
        e.printStackTrace();  
    }  
}
```

Четене на XML файл 5-5

SAX Events:

SAX Event: START DOCUMENT

SAX Event: START ELEMENT[People]

SAX Event: CHARACTERS [
]

SAX Event: START ELEMENT[Person]

SAX Event: CHARACTERS [
]

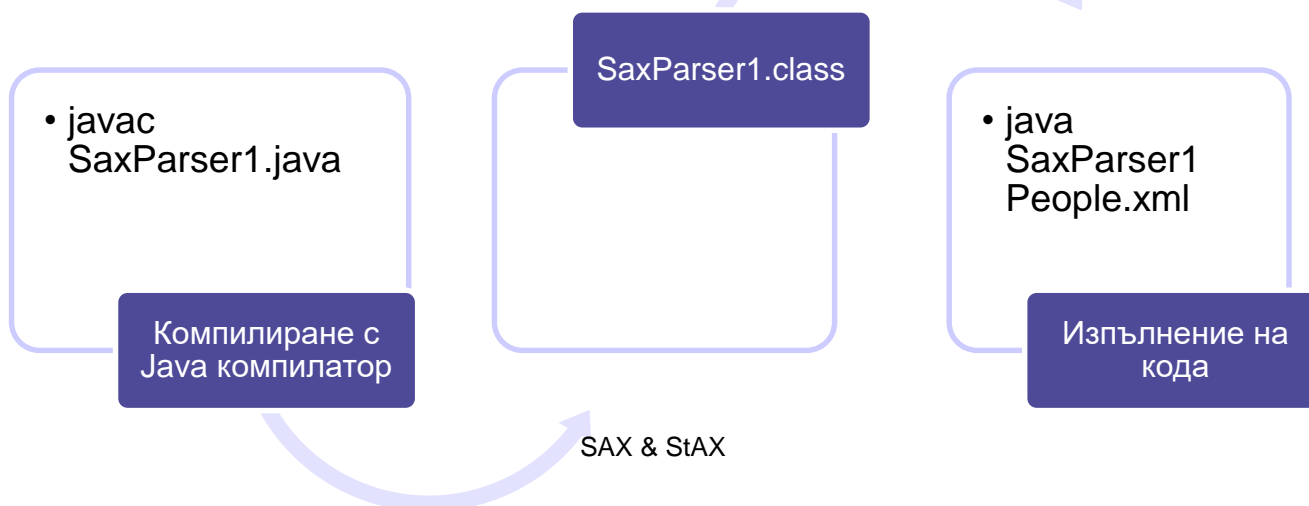
SAX Event: START ELEMENT[Name]

SAX Event: CHARACTERS[Winston
Churchill]

...

SAX Event: END ELEMENT[People]

SAX Event: END DOCUMENT



четене на Атрибути 1-2

```
public void startElement (String namespaceURI, String
localName, String qName,
                        Attributes attr ) throws
SAXException {
    System.out.println( "SAX Event: START ELEMENT[ " + localName
+ " ]" );
    for ( int i = 0; i < attr.getLength(); i++ ){
        System.out.println( " ATTRIBUTE: " + attr.getLocalName(i)
+ " VALUE: "
                        + attr.getValue(i) );
    }
}
```

- Достъп до конкретен атрибут в колекцията Attributes

```
getValue(String qName)
```

```
getValue(String uri, String localName)
```

четене на Атрибути 2-2

SAX Event: START ELEMENT[Person]

ATTRIBUTE: bornDate VALUE: 1917-05-29

ATTRIBUTE: diedDate VALUE: 1963-11-22

SAX Event: CHARACTERS[
]

SAX Event: START ELEMENT[Name]

SAX Event: CHARACTERS[John F. Kennedy]

SAX Event: END ELEMENT[Name]

SAX Event: CHARACTERS[
]

SAX Event: START ELEMENT[Description]

SAX Event: CHARACTERS[

JFK, as he was affectionately known, was a

United States president

who was assassinated in Dallas, Texas.]

SAX Event: CHARACTERS[
]

SAX Event: END ELEMENT[Description]

~~SAX~~ Event: CHARACTERS[
] ...

SAX & StAX

Прихващане на събитие characters

1-2

- При вложени текстови елементи възниква повече от едно събитие characters

- Деклариране на обект StringBuffer

```
public class SaxParser3 extends DefaultHandler {  
    private StringBuffer buffer = new StringBuffer();
```

- Добавяне на обработчик за събитието startElement

```
public void startElement(String namespaceURI, String  
    localName, String qName, Attributes attr ) throws SAXException  
{  
    System.out.println( "SAX Event: START ELEMENT[ " + localName  
+ " ]" );  
    for ( int i = 0; i < attr.getLength(); i++ ){  
        System.out.println( " ATTRIBUTE: " + attr.getLocalName(i)  
+ " VALUE: " + attr.getValue(i) );  
    }  
    buffer.setLength(0);
```

XML

SAX & StAX

43

Прихващане на събитие characters

2-2

- Добавяне на текст в буфера в обработчика на събитието characters

```
public void characters(char[] ch, int start, int length )
throws SAXException {
    try {
        buffer.append(ch, start, length);
    } catch (Exception e) {
        e.printStackTrace();    }    }
```

- Преобразуване на буфера в символен низ в обработчика на събитието endElement

```
public void endElement(String namespaceURI, String localName,
String qName ) throws SAXException {
    System.out.print("SAX Event: CHARACTERS[ " );
    System.out.println(buffer.toString());
    System.out.println( " ]" );
    System.out.println( "SAX Event: END ELEMENT[ " + localName +
" ]" );    }
```

събитието ignorableWhitespace

- Парсерът използва DTD дефиницията (ако е реферирана в документа), за да определи дали секциите с празни символи са част от документа
- Ако в DTD дефиницията е указано, че елемент може да съдържа единствено символни данни от тип PCDATA, то наличието на празни редове ще доведе до възникване на събитието ignorableWhitespace

```
public void ignorableWhitespace(char[ ] ch,  
                                int start, int len) throws SAXException
```

събитието skippedEntity

- Възниква, когато SAX парсерът достигне до съдържание, което приложението може или трябва да пропусне:
 - Референция към външен източник, който не може да бъде парснат или открит
 - Външен глобален ресурс със стойност false на характеристиката <http://xml.org/sax/features/external-general-entities>
 - Външен параметричен ресурс със стойност false на характеристиката <http://xml.org/sax/features/external-parameter-entities>
 - Името на параметъра, който трябва да се пропусне, се подава от събитието с ‘%’

```
public void skippedEntity(String name) throws SAXException
```

Събитието processingInstruction

- Възниква, когато SAX парсерът достигне до инструкция за обработка

```
public void processingInstruction(String target, String data)
throws SAXException
```

- Пример

```
<?xml-stylesheet type="text/xsl" href="myTransform.xsl"?>
```

- Параметърът target се инициализира с xml-stylesheet
- Параметърът data съдържа `type="text/xsl"`
`href="myTransform.xsl"`
- **XML декларацията в началото на XML документа не се третира като инструкция за обработка!**

Прихващане на невалидно съдържание

- DTD и XML Schema се прилагат при валидация, но осигуряват ограничени възможности
- Пример за случай, който не може да се дефинира с DTD и XML Schema
 - *Ако атрибут x е равен на y , то следващият елемент трябва да бъде $\langle a \rangle$, в противен случай $\langle b \rangle$*
- Изключението SAXException
 - Създава изключение с дефинирано потребителско съобщение
SAXException(String message)
 - Прихваща предефинирано изключение от тип Exception
SAXException(Exception e)
 - Прихваща предефинирано изключение от тип Exception и дефинира потребителски дефинирана информация за него
SAXException(String message, Exception e)

Събитието setDocumentLocator

- Използва се предимно за подобряване на разбираемостта на съобщенията за грешка
- Получава като аргумент обект от клас **Locator**
- **getLineNumber()**
 - Връща номера на реда за текущото събитие
- **getColumnNumber()**
 - Връща номера на колоната за текущото събитие (SAX спецификацията номерира колоните отлясно наляво)
- **getSystemId()**
 - Връща системния идентификатор на документа за текущото събитие
- **getPublicId()**
 - Връща публичния идентификатор на документа за текущото събитие

Събитието setDocumentLocator:

Пример 1-2

- Създаване на екземпляр на класа Locator

```
public class SaxParser4 extends DefaultHandler {  
    private Locator docLocator = null;  
    private StringBuffer buffer = new StringBuffer();  
  
    ...  
}
```

- Добавяне на обработчик за събитието setDocumentLocator

```
public void setDocumentLocator(Locator locator)  
{  
    docLocator = locator;  
}
```

Събитието setDocumentLocator:

Пример 2-2

- Получаване на текущия номер на ред в събитието startElement

```
public void startElement(String namespaceURI, String localName,
String qName, Attributes attr ) throws SAXException {
    int lineNumber = 0;
    if (docLocator != null)
    {
        lineNumber = docLocator.getLineNumber();
    }
    System.out.println( "SAX Event: START ELEMENT[ " + localName + "
]");
    if (lineNumber != 0)
    {
        System.out.println("\t"(Found at line number: " + lineNumber +
".)");
    }
    for ( int i = 0; i < attr.getLength(); i++ ){
        System.out.println( " ATTRIBUTE: " + attr.getLocalName(i) + "
VALUE: " + attr.getValue(i) );
    }
}
```

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

Събитието setDocumentLocator: Резултат

Processing 'people.xml'.

SAX Events:

SAX Event: START DOCUMENT

SAX Event: START ELEMENT[People
]

(Found at line number: 1.)

SAX Event: START ELEMENT[Person
]

(Found at line number: 2.)

ATTRIBUTE: bornDate VALUE:
1874-11-30

ATTRIBUTE: diedDate VALUE:
1965-01-24

SAX Event: START ELEMENT[Name]
(Found at line number: 3.)

SAX Event: CHARACTERS[Winston
Churchill
XML
]

SAX Event: END ELEMENT[Name]

SAX Event: START ELEMENT[
Description]

(Found at line number: 4.)

SAX Event: CHARACTERS[
Winston Churchill was a mid
20th century

British politician who
became famous as

Prime Minister during the
Second World
War.

]

SAX Event: END ELEMENT[
Description]

SAX Event: CHARACTERS[

SAX & StAX

...

Интерфейсът ErrorHandler

- Осигурява информация за възникнали грешки
- Събития
 - **warning**
 - Осигурява възможност на парсера да извести (нотифицира) приложението с предупреждение
 - **error**
 - Осигурява възможност на парсера да нотифицира приложението за грешка (не блокира парсването)
 - **fatalError**
 - Осигурява възможност на парсера да нотифицира приложението за фатална грешка (блокира парсването)
- Стъбове за събитията се осигуряват от класа **DefaultHandler**
 - Генерира **SAXException** при настъпването на събитията

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 парсерът 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);`

Интерфейсът ErrorHandler: пример 1-3

```
public static void main( String[] argv ){
    String inputFile = argv[0];
    System.out.println("Processing \"" + inputFile + "\".");
    System.out.println( "SAX Events:" );
    try {
        XMLReader reader = XMLReaderFactory.createXMLReader();
        SaxParser parser = new SaxParser();
        reader.setContentHandler(parser);
        reader.setErrorHandler(parser);
        reader.parse( new InputSource(new FileReader(
inputFile )));
    }catch ( Exception e ) {
        e.printStackTrace();
    }
}
```

Интерфейсът ErrorHandler: пример 2-3

- Активиране на характеристиката за валидация

```
try
{
    reader.setFeature(
        "http://xml.org/sax/features/validation", true);
} catch (SAXException e) {
    System.err.println("Cannot
        activate validation");
}
reader.parse(new InputSource(
    new
    FileReader(inputFile )));
```

- Създаване на DTD дефиниция

```
<!DOCTYPE People [
<!ELEMENT People (Person*)>
<!ELEMENT Person (Name,
    Description)>
<!ATTLIST Person bornDate CDATA
    #REQUIRED>
<!ATTLIST Person diedDate CDATA
    #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
]>
<!-- rest of people.xml -->
```


Интерфейсът ErrorHandler: пример 3-3

- Създаване на обработчик за събитието **warning**

```
public void warning (SAXParseException exception) throws SAXException  
{  
    System.err.println("[Warning] " + exception.getMessage() + " at  
line " + exception.getLineNumber() + ", column " +  
exception.getColumnNumber() ); };
```

- Създаване на обработчик за събитието **error**

```
public void error (SAXParseException exception) throws SAXException {  
    System.err.println("[Error] " + exception.getMessage() + " at line  
" + exception.getLineNumber() + ", column " +  
exception.getColumnNumber() ); };
```

- Създаване на обработчик за събитието **fatalError**

```
public void fatalError (SAXParseException exception) throws  
SAXException {  
    System.err.println("[Fatal Error] " + exception.getMessage() + " at  
line " + exception.getLineNumber() + ", column " +  
exception.getColumnNumber() );  
XML throw exception; };
```

Интерфейсът ErrorHandler: резултат от примера

- При премахване на атрибута `diedDate` от втория `<Person>` елемент (`Indira Gandhi`)
- Съобщение за грешка при обработка на елемента

`[Error] Attribute "diedDate" is required and must be specified for element type`

`"Person" at line 17, column 33`

`SAX Event: START ELEMENT[Person] (Found at line number: 17.)`

`ATTRIBUTE: bornDate VALUE: 1917-11-19`

DTDHandler интерфейс

- Предоставя методи за известяване за DTD събития
 - notationDecl
 - Осигурява възможност на парсера да нотифицира приложението за декларации на нотации
 - unparsedEntityDecl
 - Осигурява възможност на парсера да нотифицира приложението за декларации на ресурси, които не подлежат на парсване
- Ако SAX приложението трябва да информира за нотации и единици (entities), то трябва да имплементира този интерфейс и да регистрира негов екземпляр в парсера чрез метода **setDTDHandler**.
 - **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

Интерфейс EntityResolver 1-2

- Определя поведението на SAX парсера при преобразуване на външни референции в рамките на DTD дефиниция
- Осигурява функция `resolveEntity`, позволяваща на приложението да прихваща събития, възникващи при опит на SAX парсера да преобразува външна референция
- Реализира се от класа `DefaultHandler`
- Начин на използване
 - Аналогичен на начина на използване на интерфейсите `ContentHandler` и `ErrorHandler`

```
reader.setEntityResolver(SaxParser) ;
```

Интерфейс EntityResolver 2-2

● Метод **resolveEntity**

```
InputSource resolveEntity(String publicId, String systemId)
```

```
<!ENTITY People PUBLIC "-//People//people xml 1.0//EN"  
"http://wrox.com/people.xml">
```

○ Параметри

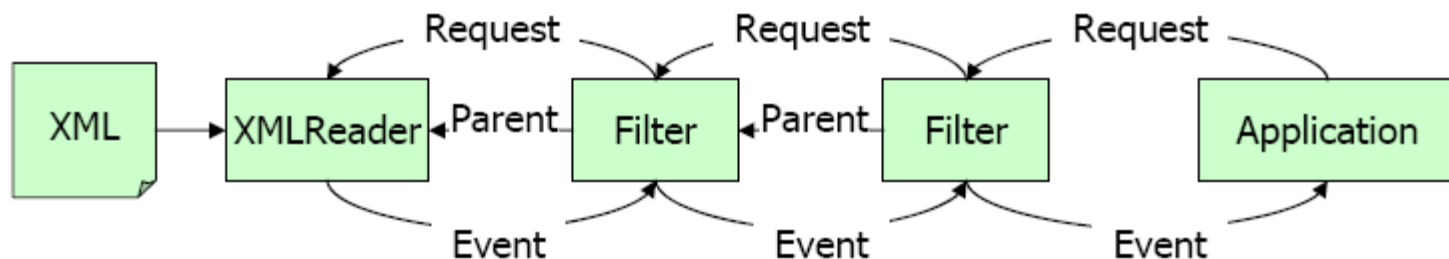
- **publicId**: Публ. идентиф. (//People//people xml 1.0//EN)
- **systemId**: Системен идентификатор
(http://wrox.com/people.xml)

○ Резултат

- **InputSource**: обектът може да се създаде въз основа на системния идентификатор или резултат върнат от база от данни, хеш таблица или каталог за избор на стойност по ключ въз основа на публичния идентификатор

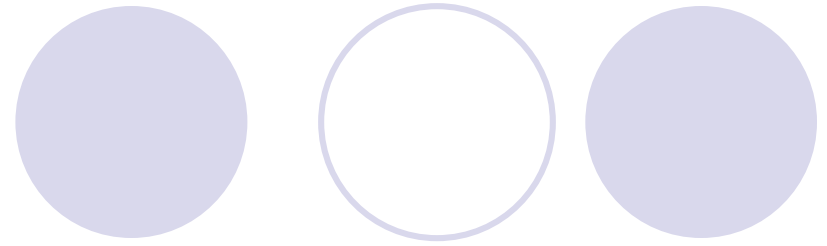
XMLFilter 1/2

- 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 ...);
        ...}
}
```

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

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

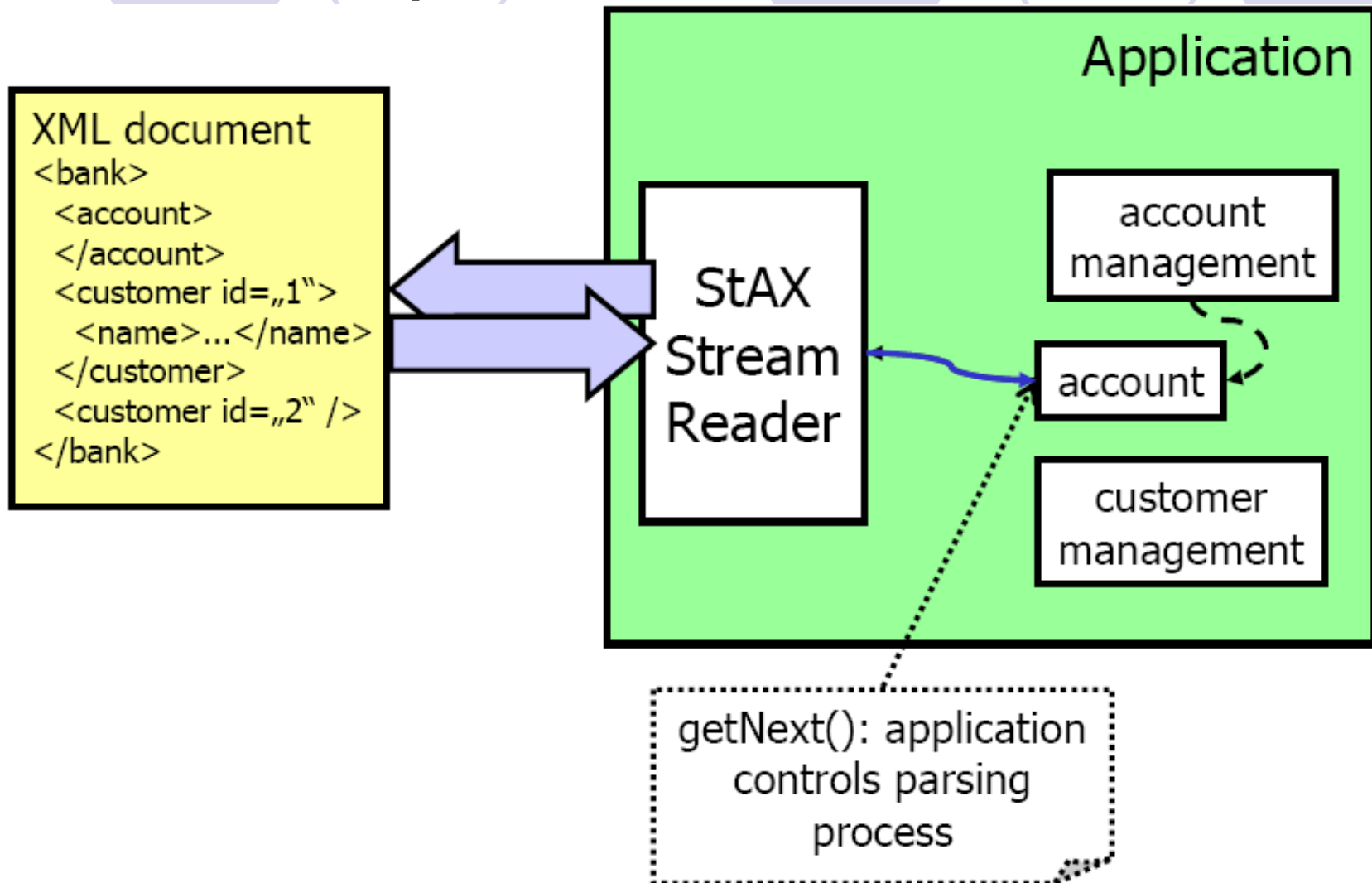
Кога да ползваме SAX (Simple API for XML) 2/2

- 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!

Алтернатива на SAX - Streaming API for XML (StAX)

- Цел
 - Лесен като DOM
 - Бърз като SAX
 - Икономичен спрямо паметта като SAX
 - API за четене и запис
- Приложението управлява парсера
 - 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 можем да се разхождаме 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 парсера в реда, в който се четат във входния XML документ.
- Базовият итераторен интерфейс се нарича **XMLEvent**.
- Основният парсерен интерфейс за четене на събития е **XMLEventReader** и основният интерфейс за писане на събития е **XMLEventWriter**.
- **XMLEventReader** имплементира **java.util.Iterator**

Създаване на StAX XMLStreamReader

- 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 парсер

- StaX парсерът (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()**, парсерът е стигнал до „текущ елемент“ и може да бъде разпитван относно него
 - Например, **getLocalName()** връща името на текущия елемент
- Важно: парсерът 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

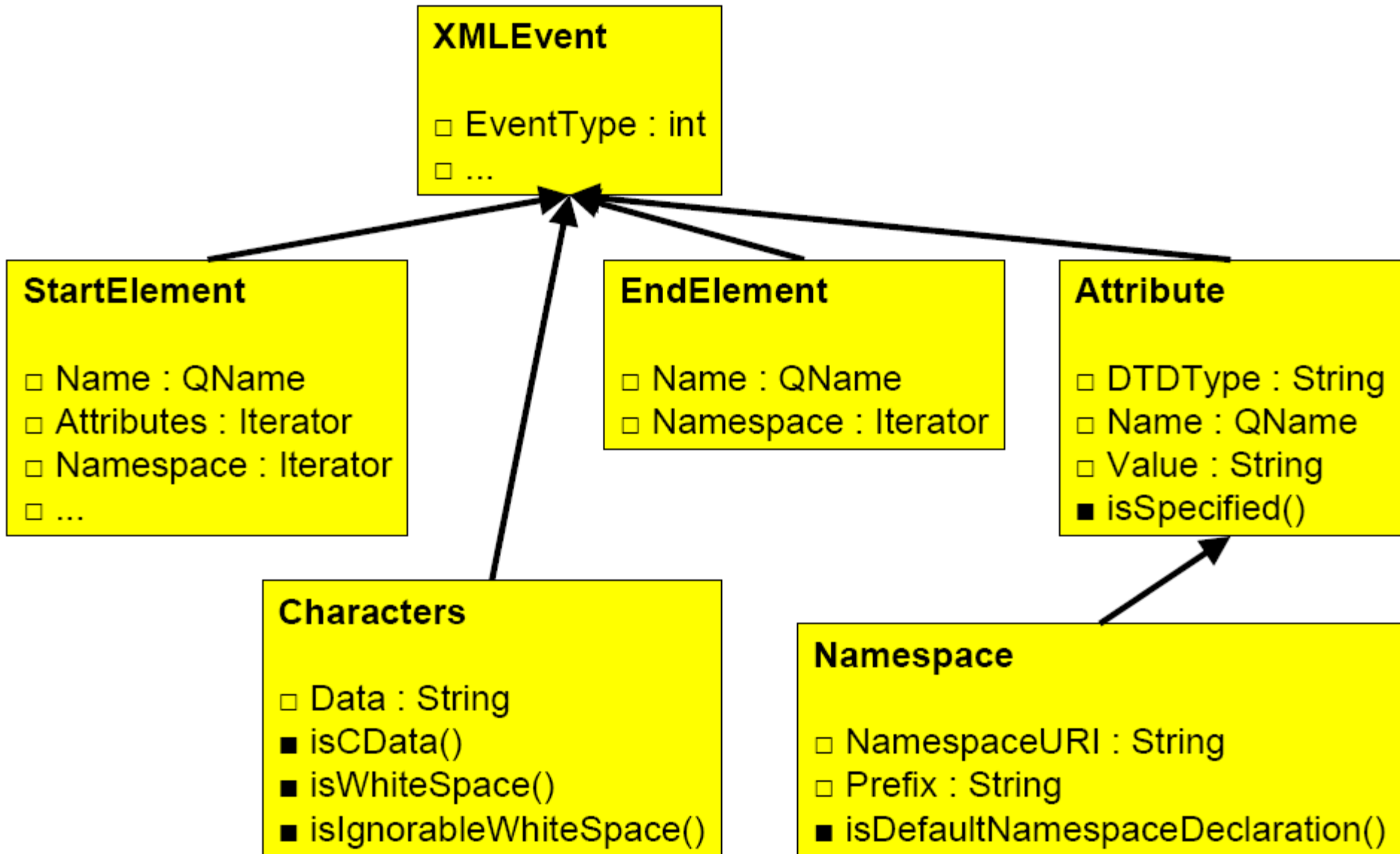
- Дефинирани са множество методи за `XMLStreamWriter`, напр.:
 - `writeStartDocument(version)` – записва XML хедър (оглавление)
 - `writeStartElement(name)` – записва стартов маркер
 - `writeAttribute(name, value)` – записва атрибут
 - `writeCharacters(value)` – записва текст, кодирайки символи като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

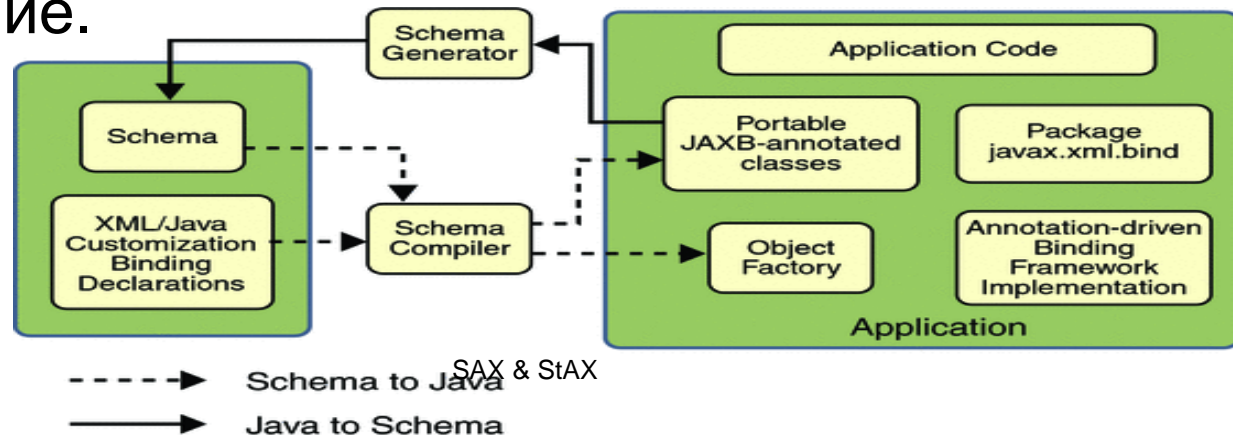
- Предимства на StAX:
 - По-лесен за ползване от SAX, особено поради липсата на callbacks
 - Може да пише в XML файлове, както и да чете от тях
- Недостатъци на StAX
 - Налага ползването на **if-then-else** за разпознаване какво се парсва
 - Както при SAX, движението е само напред
- Сравнение с DOM: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

JAXB (Java Architecture for XML Binding)

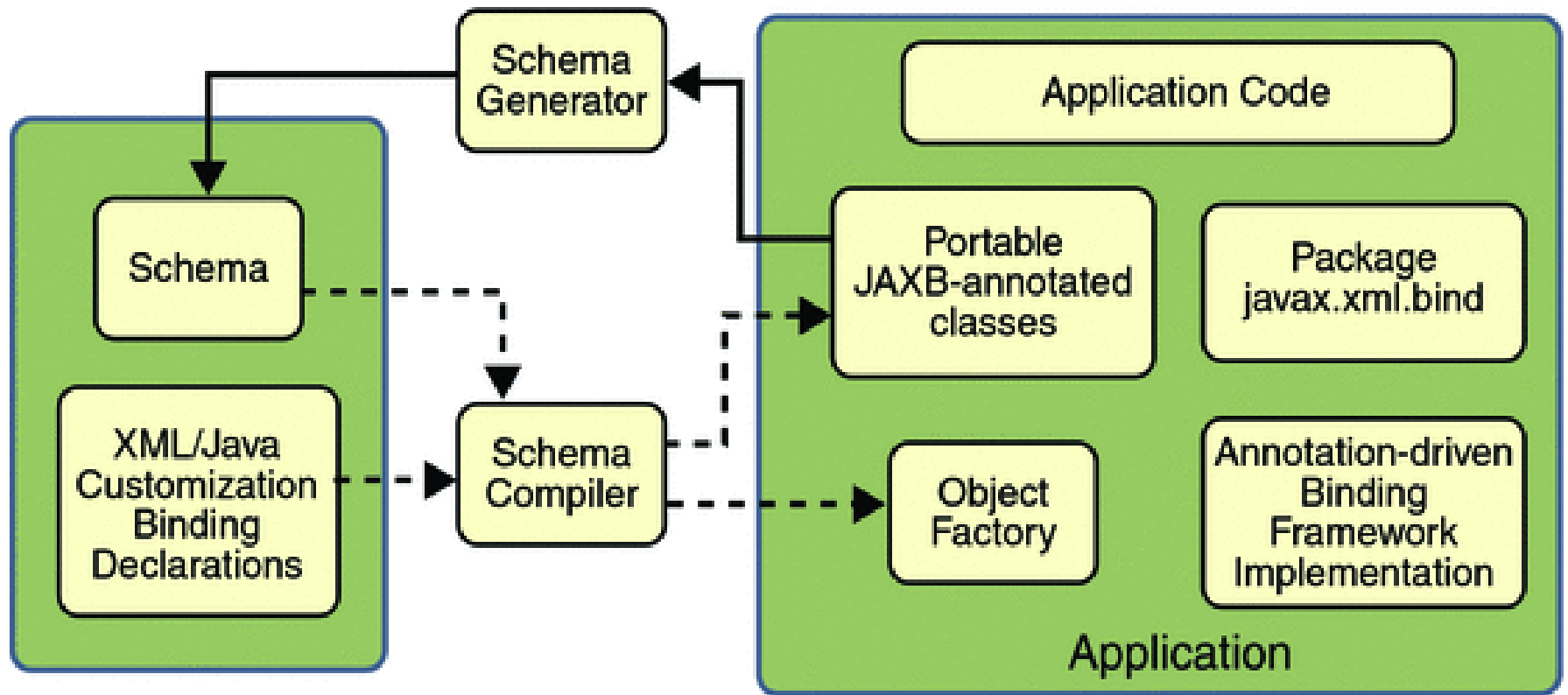
- Java Architecture for XML Binding (JAXB) разрешава на Java проектантите да съпоставят Java класове на XML съдържание.
- JAXB адресира два въпроса: възможността да се разполагат (marshal) Java обекти в XML документи и обратното - (unmarshal) XML обратно до Java обекти.
- Т.е., чрез JAXB, данни могат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

JAXB

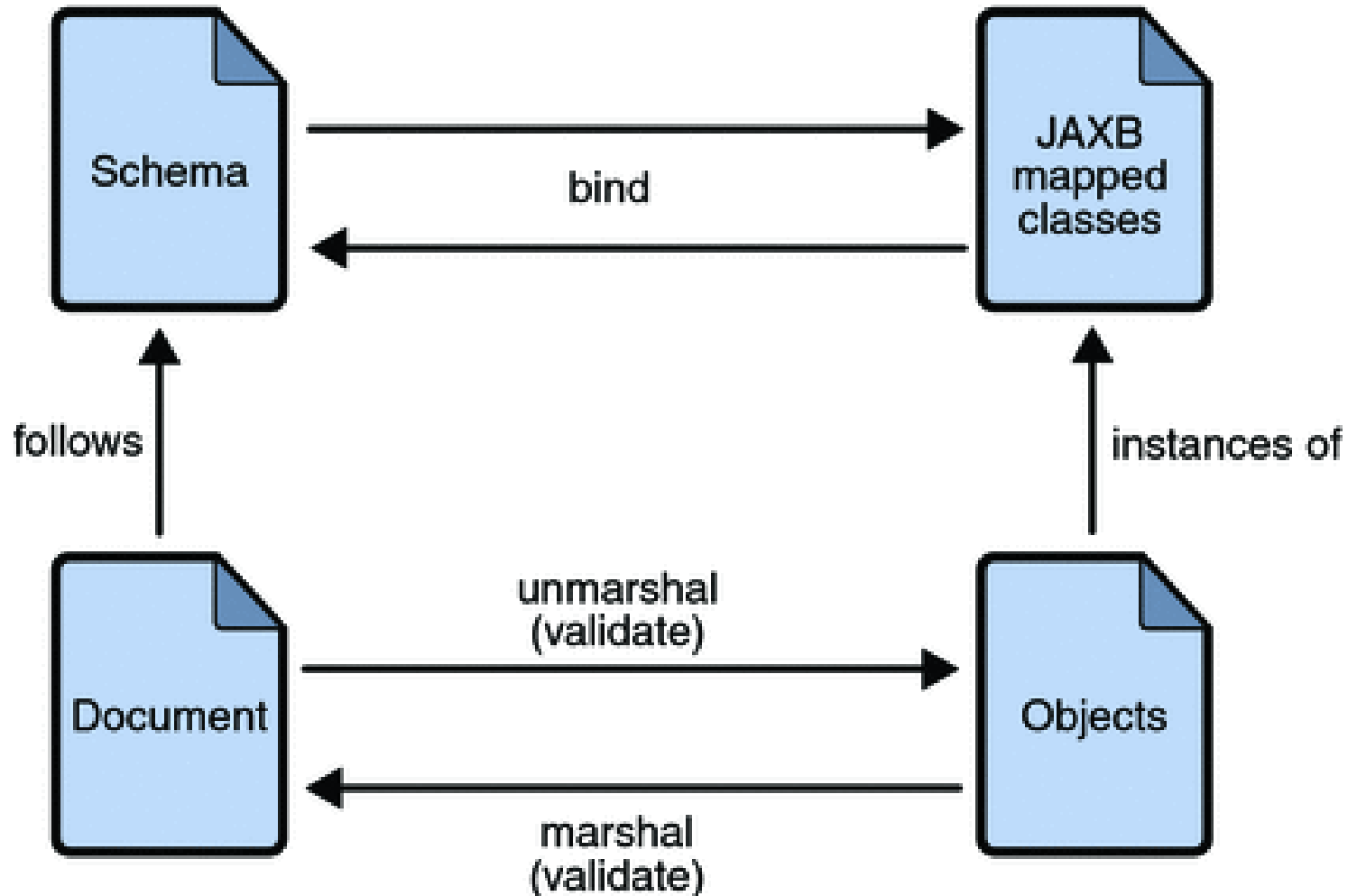
- **Schema compiler:** Свързва изходна схема към набор от програмни елементи, получени от схемата; свързването се описва от XML-базиран свързващ език.
- **Schema generator:** Съпоставя набор от съществуващи програмни елементи към производна схема (описва се от програмните анотации).
- **Binding runtime framework:** Осигурява *unmarshalling* (XML четене) и *marshalling* (XML write) операции за достъп, манипулиране и валидиране на XML съдържание.



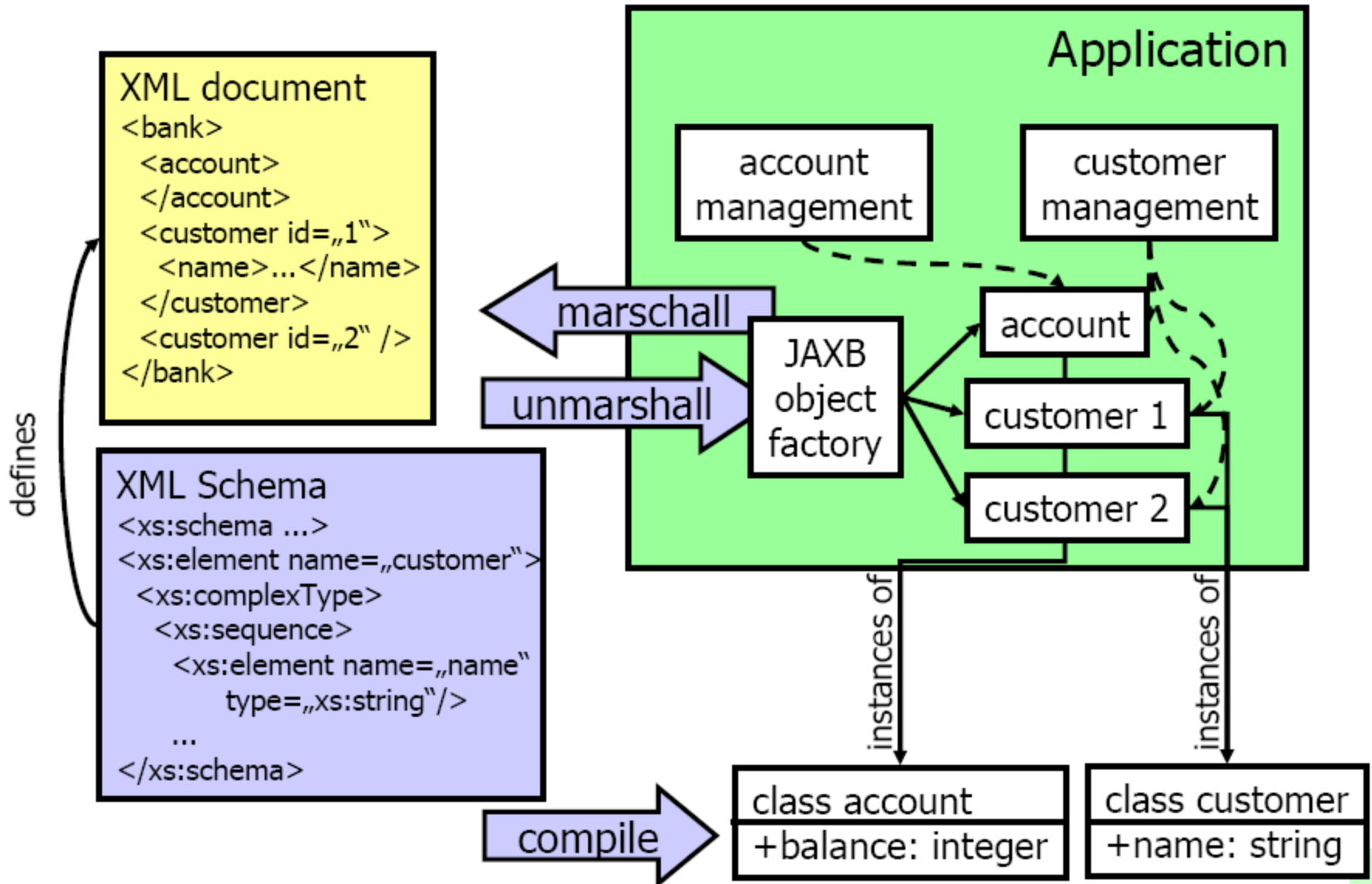
JAXB архитектура



Работен процес на JAXB - Java Architecture for XML Binding (JAXB) 1/2



Работен процес на JAXB - Java Architecture for XML Binding 2/2



Повече за JAXB

- [*https://docs.oracle.com/javase/tutorial/jaxb/intro/arch.html*](https://docs.oracle.com/javase/tutorial/jaxb/intro/arch.html)
- [*http://www.oracle.com/technetwork/articles/javase/index-140168.html*](http://www.oracle.com/technetwork/articles/javase/index-140168.html)

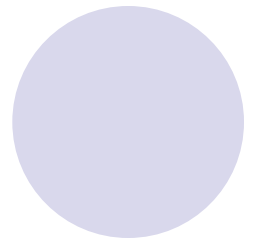
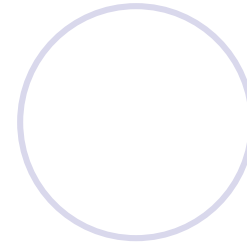
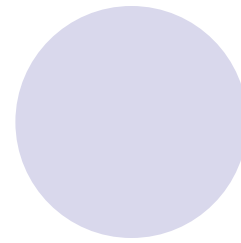
DOM спрямо StAX

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

