Комментарии документации

Комментарии документации

Java поддерживает тип комментария который называется **комментарием документации**. Такой комментарий начинается с последовательности символов `/**` и заканчивается последовательностью `*/`. Комментарии документации позволяют добавлять в программу информацию о ней самой. Позже с помощью утилиты `javadoc` (входящей в состав JDK) эту информацию можно будет извлекать и помещать в HTML файл.

Комментарии документации делают удобным процесс написания документации к разрабатываемым программам. Корпорация Sun использовала утилиту `javadoc` для документирования библиотеки `Java API`.

Комментарии документации

Дескрипторы утилиты `javadoc`

Дескрипторы `javadoc`, начинающиеся со знака `@`, называются **автономными** и должны использоваться в своей собственной строке. Дескрипторы, начинающиеся с фигурной скобки, например `{@code}`, называются встроенными и могут применяться внутри большего описания. В комментариях документации можно использовать и другие, стандартные HTML дескрипторы. Однако некоторые дескрипторы, такие как заголовки, нельзя использовать, потому что они нарушают вид HTML файла, сформированный утилитой `javadoc`.

Комментарии документации можно применять для документирования классов, интерфейсов, полей, конструкторов и методов. В каждом из случаев комментарий документации должен стоять перед документируемым элементом.

Комментарии документации

Для документирования переменной можно использовать следующие дескрипторы: @see, @serial, @serialField, {@value} и @deprecated. Для классов и интерфейсов можно использовать дескрипторы @see, @author, @deprecated, @param и @version. Методы можно документировать с помощью дескрипторов @see, @return, @param, @deprecated, @throws, @serialData, {@inheritDoc} и @exceptioп. Дескрипторы {@link}, {@docRoot}, {@code}, {@literal}, @since или {@linkplain} могут применяться где yродно. Рассмотрим каждый из этих дескрипторов.

Комментарии документации

@author

Дескриптор @author документирует автора класса. Он имеет следующий синтаксис:

```
@author описание
```

Здесь описание обычно представляет фамилию человека, написавшего класс. При выполнении утилиты `javadoc` вам нужно будет задать опцию author, чтобы поле `@author` включить в HTML документацию.

Комментарии документации

{@code}

Дескриптор {@code} позволяет встраивать в комментарий текст (например, фрагмент кода). Этот текст будет отображаться с помощью шрифта кода без последующей обработки (например, без HTML визуализации). Он имеет следующий синтаксис:

```
{@code фрагмент_кода}
```

Комментарии документации

`@deprecated`

Дескриптор `@deprecated` определяет, что класс, интерфейс или член класса является устаревшим. Рекомендуется включать дескрипторы `@see` или `{@link}` для toro, чтобы информировать программиста о доступных альтернативных вариантах. Синтаксис этого дескриптора выглядит следующим образом:

```
@deprecated описание
```

Здесь описание это сообщение, описывающее исключение. Дескриптор `@deprecated` может использоваться для документирования переменных, методов и классов.

Комментарии документации

`{@docRoot}`

Дескриптор `{@docRoot}` определяет путь к корневому каталогу текущей документации.

Комментарии документации

`@exception`

Дескриптор `@exception` описывает исключение для данного метода. Он имеет следующий синтаксис:

```
@exception имя_исключения пояснение
```

Здесь имя_исключения указывает полное имя исключения, а пояснение представляет строку, которая описывает, в каких случаях может возникнуть данное исключение. Дескриптор `@exception` может использоваться только для документирования методов.

Комментарии документации

`{@inheritDoc}`

Этот дескриптор наследует комментарий от непосредственного суперкласса.

Комментарии документации

{@link}

Дескриптор `{@link}` предлагает встроенную ссылку на дополнительную информацию. Он имеет следующий синтаксис:

```
{@link пакет.класс#член текст}
```

Здесь пакет.класс#член определяет имя класса или метода, на который добавляется ссылка, а текст представляет отображаемую строку.

Комментарии документации

`{@linkplain}`

Вставляет встроенную ссылку на другую тему. Ссылка отображается в открытой форме шрифта. В противном случае дескриптор аналогичен `{@link}`.

Комментарии документации

`{@literal}`

Дескриптор `{@literal}` позволяет встраивать текст в комментарий. Этот текст отображается "как есть" без последующей обработки (например, без HTML визуализации).

Он имеет следующий синтаксис:

```
{@literal описание}
```

Здесь описание представляет встраиваемый текст.

Комментарии документации

@param

Дескриптор `@param` документирует параметр для метода или параметр типа для класса или интерфейса. Он имеет следующий синтаксис:

```
@param имя_параметра пояснение
```

Здесь имя_параметра представляет имя параметра. Назначение этого параметра определяется соответствующим пояснением. Дескриптор `@param` может использоваться только для документирования метода, конструктора или обобщенного класса или интерфейса.

Комментарии документации

`@return`

Дескриптор `@return` описывает возвращаемое значение метода. Он имеет следующий синтаксис:

```
@return пояснение
```

Здесь пояснение описывает тип и смысл значения, возвращаемого методом. Дескриптор `@return` может использоваться только для документирования метода.

Комментарии документации

@see

Дескриптор @see обеспечивает ссылку на дополнительную информацию. Далее показаны наиболее распространенные его формы:

```
@see привязка
@see пакет.класс#член текст
```

в первой форме привязка представляет ссылку на абсолютный или относительный URL адрес. Во второй форме пакет.класс#член определяет имя элемента, а текст представляет отображаемый для данного элемента текст. Текстовый параметр необязателен и если не используется, то отображается элемент, указанный в параметре пакет.класс#член. Имя члена тоже является необязательным.

Комментарии документации

Таким образом, вы можете определить ссылку на модуль, класс или интерфейс в дополнение к ссылке на определенный метод или поле. Имя может быть определено полностью или частично. Однако точку, стоящую перед именем члена (если таковой существует), необходимо заменить символом #.

`@serial`

Дескриптор `@serial` определяет комментарий для поля, сериализируемого по умолчанию. Он имеет следующий синтаксис:

```
@serial описание
```

Здесь описание определяет комментарий для данного поля.

Комментарии документации

`@serialData`

Дескриптор `@serialData` документирует данные, записанные с помощью методов `writeObject()` и `writeExternal()`. Он имеет следующий синтаксис:

```
@serialData описание
```

Здесь описание определяет комментарий для этих данных.

Комментарии документации

`@serialField`

Для класса, реализующего `Serializable`, дескриптор `@serialField` предлагает комментарии для компонента `ObjectStreamField`. Он имеет следующий синтаксис:

```
@serialField имя тип описание
```

Здесь имя представляет имя поля, тип представляет его тип, а описание комментарий для данного поля.

Комментарии документации

`@since`

Дескриптор `@since` показывает, что класс или член класса был впервые представлен в определенном выпуске. Он имеет следующий синтаксис:

```
@since выпуск
```

Здесь выпуск представляет строку, в которой указан выпуск или версия, начиная с которого эта особенность стала доступной.

Комментарии документации

`@throws`

Дескриптор `@throws` имеет то же назначение, что и дескриптор `@exception`.

Комментарии документации

{@value}

Дескриптор `{@value}` имеет две формы. Первая отображает значение следующей за ним константы, которой должно являться поле `static`. Его форма показана ниже:

```
{@value}
```

Вторая форма отображает значение определенного поля `static`. В этом случае дескриптор имеет следующую форму:

```
{@value пакет.класс#поле}
```

Здесь пакет.класс#поле определяет имя `static` поля.

Комментарии документации

@version

Дескриптор `@version` определяет версию класса. Он имеет следующий синтаксис:

```
@version информация
```

Здесь информация представляет строку, содержащую информацию о версии (как правило, номер версии, например 2.2). При выполнении утилиты `javadoc` вам нужно будет указать опцию `version`, чтобы поле `@version` включить в HTML документацию.

Комментарии документации

Общая форма комментариев документации

После начальной комбинации символов /** первая строка или строки становятся главным описанием вашего класса, переменной или метода.

После них можно включать один или более различных дескрипторов @. Каждый дескриптор @ должен стоять в начале новой строки или следовать за одним или несколькими символами звездочки (*), находящимися в начале строки. Несколько дескрипторов одного и того же типа необходимо группировать вместе. Например, если у вас имеется три дескриптора @see, их следует поместить друг за другом. Встроенные дескрипторы (они начинаются с фигурной скобки) можно помещать внутри любого описания.

Комментарии документации

Вывод утилиты `javadoc`

Утилита `javadoc` в качестве входных данных принимает файл с исходным кодом вашей java-программы и выводит несколько HTML-файлов, содержащих документацию по этой программе. Информация о каждом классе будет содержаться в его собственном HTML-файле. Утилита `javadoc` выводит также дерево индексов и иерархии. Могут быть сгенерированы и другие HTML файлы.

Комментарии документации

Пример:

```
package csdev;

import java.io.*;
import javax.xml.bind.*;

/**
 * <p>Этот класс демонстрирует применение комментариев документации.
 *
 * <p>Будем использовать этот класс как базовый для клиент/сервер
 * взаимодействия в формате XML.
 *
 * @author Sergey Gutnikov
 * @version 1.0
 */
public class MessageXml implements Serializable {

    private static final long serialVersionUID = 1L;
```

Комментарии документации

```
/**
  *  Создание сообщения <code>MessageXml</code>.
  *
  */
public MessageXml() {
}

 /**
  * Этот статический метод преобразует объект <code>MessageXml</code> в
  * Xml-формат и записывает результат в поток <code>OutputStream</code>.
  *
  * @param msg Объект для преобразования в Xml-формат
  * @param os Выходной поток
  * @exception В случае ошибки генерируется исключение:
  * @return Метод ничего не возвращает.
  * @see JAXBException
  */
public static void toXml( MessageXml msg, OutputStream os )
            throws JAXBException {
```

Комментарии документации

```
        JAXBContext context =
                        JAXBContext.newInstance(msg.getClass());
        Marshaller m = context.createMarshaller();
        m.marshal( msg, os );
}

 /**
  * Этот статический метод читает объект известного
  * класса <code>Class<? extends MessageXml></code> в Xml-формате из
  * потока <code>InputStream</code>.
  *
  * @param what Класс ожидаемого объекта во входном потоке
  * @param is Входной поток
  * @exception В случае ошибки генерируется исключение:
  * @return <code>MessageXml</code> - прочитанный объект.
  * @see JAXBException
  */
public static MessageXml fromXml( Class<? extends MessageXml> what,
                    InputStream is ) throws JAXBException {
```

Комментарии документации

```
            JAXBContext context =
                            JAXBContext.newInstance(what);
            Unmarshaller u = context.createUnmarshaller();
            return ( MessageXml ) u.unmarshal( is );
    }
}
```