Сериализация

Croc Java School

Сериализация/десериализация

Сериализация - перевод данных в последовательность символов или байтов. Обратный процесс - десериализация.

Применяется для обмена данными между приложениями (интеграции) и для сохранения состояния между запусками.

В Java есть встроенный механизм сериализации, но мы его не используем

Java Serialization

Mark Reinhold: Serialization was a “horrible mistake” made in 1997.

Первая попытка отказаться от встроенной сериализации:

JEP 154: Remove Serialization (endorsed by Brian Goetz)

Что использовать вместо стандартной сериализации

Текстовые форматы:

- JSON
- XML

Бинарные форматы:

- Protobuf
- Thrift

Java

```
public class User {

  private Long id;
  private String userName;
  private String email;
}
```

JSON

```
{
  "id": 12,
  "name": "Petr",
  "email": "petr@croc.ru"
}
```

XML

```
<User>
  <Id>12</Id>
  <Name>Petr</Name>
  <Email>petr@croc.ru</Email>
</User>
```

Protobuf (схема)

```
message DtoUser {
  required int64 id = 1;
  optional string name = 2;
  required string email = 3;
}
```

Парсеры

Для поддержки всех этих форматов потребуется подключить внешние библиотеки:

JSON
Gson или Jackson

XML
JAXB

Protobuf
protobuf-java

Jackson

Jackson Project

https://github.com/FasterXML/jackson

> Jackson has been known as "the Java JSON library" or "the best JSON parser for Java". Or simply as "JSON for Java".

Jackson. Разметка класса

```
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonIgnoreProperties(ignoreUnknown = true)
public class User {

  private Long id;

  @JsonProperty("name")
  private String userName;

  private String email;

  private User boss;
}
```

Jackson. Сериализация

```
User olga = new User();
olga.setUserName("Olga");
olga.setEmail("olga@croc.ru");

User petr = new User();
petr.setUserName("Petr");
petr.setEmail("petr@croc.ru");
petr.setBoss(olga);

ObjectMapper mapper = new ObjectMapper()
    .enable(SerializationFeature.INDENT_OUTPUT);
String serialized = mapper.writeValueAsString(petr);
```

```
{
  "email" : "petr@croc.ru",
  "boss" : {
    "email" : "olga@croc.ru",
    "name" : "Olga"
  },
  "name" : "Petr"
}
```

Jackson. Десериализация

```
ObjectMapper mapper = new ObjectMapper();
User user = mapper.readValue(serialized, User.class);
```

Структура данных для десериализации может быть не определена, с ней можно работать в динамическом виде:

```
ObjectMapper mapper = new ObjectMapper();
JsonNode json = mapper.readTree(serialized);
```

JAXB

JAXB. Разметка класса

```
@XmlRootElement(name = "user")
public class User {
  private Long id;
  private String userName;
  private String email;
  private User boss;

  @XmlAttribute
  public void setId(Long id) { this.id = id;}

  @XmlElement(name = "name")
  public void setUserName(String userName) { this.userName = userName; }

  @XmlElement
  public void setEmail(String email) { this.email = email; }

  @XmlElement
  public void setBoss(User boss) { this.boss = boss; }
}
```

JAXB. Сериализация

Те же действующие лица, только теперь с заполненными идентификаторами.

```
User olga = new User();
olga.setId(1L);
olga.setUserName("Olga");
olga.setEmail("olga@croc.ru");

User petr = new User();
petr.setId(2L);
petr.setUserName("Petr");
petr.setEmail("petr@croc.ru");
petr.setBoss(olga);
```

```
JAXBContext context = JAXBContext.newInstance(User.class);
Marshaller marshaller = context.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

StringWriter w = new StringWriter();
marshaller.marshal(petr, w);
String serialized = w.toString();
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user id="2">
  <boss id="1">
    <email>olga@croc.ru</email>
    <name>Olga</name>
  </boss>
  <email>petr@croc.ru</email>
  <name>Petr</name>
</user>
```

JAXB. Десериализация

```
Unmarshaller unmarshaller = context.createUnmarshaller();
User user = (User)unmarshaller
    .unmarshal(new StringReader(serialized));
```

XML скорее мертв, чем жив, но в легаси-проектах его все еще много

Итого

Не используем стандартную Java-сериализацию.

Для сериализации данных в текстовом виде выбираем JSON или XML.

Всегда подключаем библиотеки: не парсим JSON и XML вручную (в том числе регулярными выражениями).

Если важно более компактное представление, вместо текстовых форматов используем бинарные. В этом случае подойдет protobuf, но есть и альтернативы.

You can't parse [X]HTML with regex. Because HTML can't be parsed by regex. Regex is not a tool that can be used to correctly parse HTML. As I have answered in HTML-and-regex questions here so many times before, the use of regex will not allow you to consume HTML. Regular expressions are a tool that is insufficiently sophisticated to understand the constructs employed by HTML. HTML is not a regular language and hence cannot be parsed by regular expressions. Regex queries are not equipped to break down HTML into its meaningful parts. so many times but it is not getting to me. Even enhanced irregular regular expressions as used by Perl are not up to the task of parsing HTML. You will never make me crack. HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow *it is too late it is too late we cannot be saved* the transgression of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) *dear lord help us how can anyone survive this scourge* using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes *using regex* as a tool to process HTML establishes a breach *between this world* and the dread realm of corrupt entities (like SGML entities, but *more corrupt) a mere glimpse* of the world of reg**ex parsers for HTML will ins**tantly transport a *programmer's consciousness* into a *world* of ceaseless screaming, he comes~~, the pestilent sl~~ithy regex-infection wil**l devour your HT**ML parser, application and existence for all time like Visual Basic only worse *he comes he comes do not fi*ght h**e comes, hi**s unholy radiance destroying *all enlightenment, HTML tags* ***leaking from your eyes like liq****uid* pain, the song of regular expre~~ssion parsing~~ will ext*inguish the voices of mor**tal man from the sp**here I can see it can you see it it is beautiful* the f`inal snuf`fing *of the lie**s of Man ALL IS LOST*** **ALL IS L**OST the *pony he comes* he comes ~~he comes~~ ***the ich***or permeates *all* MY FACE *MY FACE* h *godno* ***NO NO*** **OOO N**O stop *the an*gles `gre` n**ot real ZALGO IS** TONY THE **PONY HE COMES**

Have you tried using an XML parser instead?