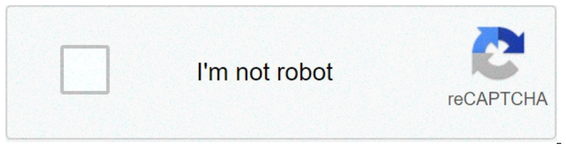

Continue

Java json reader example

The GSON JsonParser class can parse a JSON string or stream into a tree structure of Java objects. GSON also has two other parsers. The Gson JSON parser which can parse JSON into Java objects, and the JsonReader which can parse a JSON string or stream into tokens (a pull parser). This tutorial focuses on the JsonParser though - GSON's tree parser. Creating a JsonParser Before you can use the GSON JsonParser you must create a JsonParser instance. Here is an example of creating a JsonParser instance: JsonParser jsonParser = new JsonParser(); Parsing JSON Into a Tree Structure Once you have created a JsonParser you can parse JSON into a tree structure with it. Here is an example of parsing a JSON string into a tree structure of GSON objects with the JsonParser: JsonParser parser = new JsonParser(); String json = "{ \"f1\":\"Hello\",\"f2\":{\"f3\":\"World\"}}"; JsonElement jsonTree = parser.parse(json); The parsing of the JSON happens in the third line of code, by calling parse() on the JsonParser, passing as parameter a reference to the JSON string (or stream) to parse. Iterating the JSON Tree Structure The parsed JSON tree structure consists of objects from the GSON API. The root of a JSON tree structure is a JsonElement object. You can find out what type of JSON element it represents using one of the type checking methods: jsonTree.isJsonObject(); jsonTree.isJsonArray(); jsonTree.isJsonNull(); jsonTree.isJsonPrimitive(); The JSON string parsed above is a JSON object. Thus, we will expect the JsonElement to represent a JSON object. If it does, we will do something with it. Here is how that looks: if(jsonTree.isJsonObject()) { JsonObject jsonObject = jsonTree.getAsJsonObject(); } Once you have a JsonObject instance you can extract fields from it using its get() method. Here is an example: JsonObject jsonObject = jsonTree.getAsJsonObject(); JsonElement f1 = jsonObject.get("f1"); JsonElement f2 = jsonObject.get("f1"); You can inspect the type of each of these fields too, just like with the first JsonElement obtained from the JsonParser parse() method. Here is an example showing how: if(f2.isJsonObject()){ JsonObject f2Obj = f2.getAsJsonObject(); JsonElement f3 = f2Obj.get("f3"); } Here is a full example showing how to iterate the JsonElement obtained from the JsonReader : JsonParser parser = new JsonParser(); String json = "{ \"f1\":\"Hello\",\"f2\":{\"f3\":\"World\"}}"; JsonElement jsonTree = parser.parse(json); if(jsonTree.isJsonObject()){ JsonObject jsonObject = jsonTree.getAsJsonObject(); JsonElement f1 = jsonObject.get("f1"); JsonElement f2 = jsonObject.get("f2"); if(f2.isJsonObject()){ JsonObject f2Obj = f2.getAsJsonObject(); JsonElement f3 = f2Obj.get("f3"); } } Creating a GSON JsonReaderIterating the JSON Tokens of a JsonReader The GSON JsonReader is the GSON streaming JSON parser. The GSON JsonReader enables you to read a JSON string or file as a stream of JSON tokens. Iterating JSON token for token is also referred to as streaming through the JSON tokens. That is why the GSON JsonReader is also sometimes referred to as a streaming JSON parser. Streaming parsers typically come in two versions: Pull parsers and push parsers. A pull parser is a parser where the code using it pulls the tokens out of the parser when the code is ready to handle the next token. A push parser parses through the JSON tokens and pushes them into an event handler. The GSON JsonReader is a pull parser. In this GSON JsonReader tutorial we will take a closer look at what that means, and how the JsonReader works. Creating a GSON JsonReader You create a GSON JsonReader like this: String json = "{\"brand\" : \"Toyota\", \"doors\" : 5}"; JsonReader jsonReader = new JsonReader(new StringReader(json)); Notice how the JsonReader constructor takes a Java Reader as parameter. In the example above we pass a Java StringReader to the JsonReader's constructor. The StringReader is capable of "converting" a Java string into a character stream (a Reader in other words). Iterating the JSON Tokens of a JsonReader Once you have created a JsonReader instance you can iterate through the JSON tokens it reads from the Reader passed to the JsonReader's constructor. The main look for iterating the JSON tokens of a JsonReader looks like this: while(jsonReader.hasNext() { } The hasNext() method of the JsonReader returns true if the JsonReader has more tokens. To access the tokens of the JsonReader you use a loop similar to the following: String json = "{\"brand\" : \"Toyota\", \"doors\" : 5}"; JsonReader jsonReader = new JsonReader(new StringReader(json)); try { while(jsonReader.hasNext()){ JsonToken nextToken = jsonReader.peek(); System.out.println(nextToken); if(JsonToken.BEGIN_OBJECT.equals(nextToken)){ jsonReader.beginObject(); } else if(JsonToken.NAME.equals(nextToken)){ String name = jsonReader.nextName(); System.out.println(name); } else if(JsonToken.STRING.equals(nextToken)){ String value = jsonReader.nextString(); System.out.println(value); } else if(JsonToken.NUMBER.equals(nextToken)){ long value = jsonReader.nextLong(); System.out.println(value); } } } catch (IOException e) { e.printStackTrace(); } The JsonReader peek() method returns the next JSON token, but without moving over it. Multiple calls to peek() subsequently will return the same JSON token. The JsonToken returned by peek() can be compared to constants in the JsonToken class to find out what type of token it is. You can see that done in the loop above. Inside each if statement a JsonReader method is called which moves the JsonReader over the current token, and on to the next token. All of beginObject(), nextString() and nextLong() return the value of the current token and moves the internal pointer to the next. public final class JsonReader extends Object implements Closeable java.lang.Object ↳ android.util.JsonReader Reads a JSON (RFC 4627) encoded value as a stream of tokens. This stream includes both literal values (strings, numbers, booleans, and nulls) as well as the begin and end delimiters of objects and arrays. The tokens are traversed in depth-first order, the same order that they appear in the JSON document. Within JSON objects, name/value pairs are represented by a single token. Parsing JSON To create a recursive descent parser for your own JSON streams, first create an entry point method that creates a JsonReader. Next, create handler methods for each structure in your JSON text. You'll need a method for each object type and for each array type. Within array handling methods, first call beginArray() to consume the array's opening bracket. Then create a while loop that accumulates values, terminating when hasNext() is false. Finally, read the array's closing bracket by calling endArray(). Within object handling methods, first call beginObject() to consume the object's opening brace. Then create a while loop that assigns values to local variables based on their name. This loop should terminate when hasNext() is false. Finally, read the object's closing brace by calling endObject(). When a nested object or array is encountered, delegate to the corresponding handler method. When an unknown name is encountered, strict parsers should fail with an exception. Lenient parsers should call skipValue() to recursively skip the value's nested tokens, which may otherwise conflict. If a value may be null, you should first check using peek(). Null literals can be consumed using either nextNull() or skipValue(). Example Suppose we'd like to parse a stream of messages such as the following: [{ "id": 912345678901, "text": "How do I read JSON on Android?", "geo": null, "user": { "name": "android_newb", "followers_count": 41 } }, { "id": 912345678902, "text": "@android_newb just use android.util.JsonReader!", "geo": [50.454722, -104.606667], "user": { "name": "jesse", "followers_count": 2 } }] This code implements the parser for the above structure: public List readJsonStream(InputStream in) throws IOException { JsonReader reader = new JsonReader(new InputStreamReader(in, "UTF-8")); try { return readMessagesArray(reader); } finally { reader.close(); } } public List readMessagesArray(JsonReader reader) throws IOException { List messages = new ArrayList(); reader.beginArray(); while (reader.hasNext()) { messages.add(readMessage(reader)); } reader.endArray(); return messages; } public Message readMessage(JsonReader reader) throws IOException { long id = -1; String text = null; User user = null; List geo = null; reader.beginObject(); while (reader.hasNext()) { String name = reader.nextName(); if (name.equals("id")) { id = reader.nextLong(); } else if (name.equals("text")) { text = reader.nextString(); } else if (name.equals("geo") && reader.peek() != JsonToken.NULL) { geo = readDoublesArray(reader); } else if (name.equals("user")) { user = readUser(reader); } else { reader.skipValue(); } } reader.endObject(); return new Message(id, text, user, geo); } public List readDoublesArray(JsonReader reader) throws IOException { List doubles = new ArrayList(); reader.beginArray(); while (reader.hasNext()) { doubles.add(reader.nextDouble()); } reader.endArray(); return doubles; } public User readUser(JsonReader reader) throws IOException { String username = null; int followersCount = -1; reader.beginObject(); while (reader.hasNext()) { String name = reader.nextName(); if (name.equals("name")) { username = reader.nextString(); } else if (name.equals("followers_count")) { followersCount = reader.nextInt(); } else { reader.skipValue(); } } reader.endObject(); return new User(username, followersCount); } Number Handling This reader permits numeric values to be read as strings and string values to be read as numbers. For example, both elements of the JSON array [1, "1"] may be read using either nextInt() or nextString(). This behavior is intended to prevent lossy numeric conversions: double is JavaScript's only numeric type and very large values like 9007199254740993 cannot be represented exactly on that platform. To minimize precision loss, extremely large values should be written and read as strings in JSON. Each JsonReader may be used to read a single JSON stream. Instances of this class are not thread safe. JsonReader(Reader in) Creates a new instance that reads a JSON-encoded stream from in. void beginArray() Consumes the next token from the JSON stream and asserts that it is the beginning of a new array. void beginObject() Consumes the next token from the JSON stream and asserts that it is the beginning of a new object. void close() Closes this JSON reader and the underlying Reader. void endArray() Consumes the next token from the JSON stream and asserts that it is the end of the current array. void endObject() Consumes the next token from the JSON stream and asserts that it is the end of the current object. boolean hasNext() Returns true if the current array or object has another element. boolean isLenient() Returns true if this parser is liberal in what it accepts. boolean nextBoolean() Returns the boolean value of the next token, consuming it. double nextDouble() Returns the double value of the next token, consuming it. int nextInt() Returns the int value of the next token, consuming it. long nextLong() Returns the long value of the next token, consuming it. String nextName() Returns the next token, a property name, and consumes it. void nextNull() Consumes the next token from the JSON stream and asserts that it is a literal null. String nextString() Returns the string value of the next token, consuming it. JsonToken peek() Returns the type of the next token without consuming it. void setLenient(boolean lenient) Configure this parser to be be liberal in what it accepts. void skipValue() Skips the next value recursively. String toString() Returns a string representation of the object. From class java.lang.Object Object clone() Creates and returns a copy of this object. boolean equals(Object obj) Indicates whether some other object is "equal to" this one. void finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object. final Class getClass() Returns the runtime class of this Object. int hashCode() Returns a hash code value for the object. final void notify() Wakes up a single thread that is waiting on this object's monitor. final void notifyAll() Wakes up all threads that are waiting on this object's monitor. String toString() Returns a string representation of the object. final void wait(long timeout, int nanos) Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed. final void wait(long timeout) Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed. final void wait() Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object. From interface java.io.Closeable abstract void close() Closes this stream and releases any system resources associated with it. Public constructors public JsonReader (Reader in) Creates a new instance that reads a JSON-encoded stream from in. Public methods public void beginArray () Consumes the next token from the JSON stream and asserts that it is the beginning of a new array. public void beginObject () Consumes the next token from the JSON stream and asserts that it is the beginning of a new object. public void endArray () Consumes the next token from the JSON stream and asserts that it is the end of the current array. public void endObject () Consumes the next token from the JSON stream and asserts that it is the end of the current object. public boolean hasNext () Returns true if the current array or object has another element. public boolean isLenient () Returns true if this parser is liberal in what it accepts. public int nextInt () Returns the int value of the next token, consuming it. If the next token is a string, this method will attempt to parse it as an int. If the next token's numeric value cannot be exactly represented by a Java int, this method throws. public long nextLong () Returns the long value of the next token, consuming it. If the next token is a string, this method will attempt to parse it as a long. If the next token's numeric value cannot be exactly represented by a Java long, this method throws. public void nextNull () Consumes the next token from the JSON stream and asserts that it is a literal null. public void setLenient (boolean lenient) Configure this parser to be be liberal in what it accepts. By default, this parser is strict and only accepts JSON as specified by RFC 4627. Setting the parser to lenient causes it to ignore the following syntax errors: End of line comments starting with // or # and ending with a newline character. C-style comments starting with /* and ending with */. Such comments may not be nested. Names that are unquoted or 'single quoted'. Strings that are unquoted or 'single quoted'. Array elements separated by ; instead of ,. Unnecessary array separators. These are interpreted as if null was the omitted value. Names and values separated by = or => instead of :. Name/value pairs separated by ; instead of ,. Parameters lenient boolean public void skipValue () Skips the next value recursively. If it is an object or array, all nested elements are skipped. This method is intended for use when the JSON token stream contains unrecognized or unhandled values. public String toString () Returns a string representation of the object. In general, the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method. The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of: getClass().getName() + '@' + Integer.toHexString(hashCode()) Returns String a string representation of the object.