CS 240: JSON DOM Parser Transcript

[00:00:00] The next kind of Jason parser that we're going to talk about is a Dom parser.

[00:00:04] Dom stands for document object model or sometimes called a tree parser.

[00:00:09] If you think about a Jason file, it's really um just a textual syntax for representing a tree of data or a hierarchy.

Start visual description. The professor demonstrates the structure of a JSON file, explaining that it represents a tree of data or a hierarchy. The professor shows a diagram of a tree with an object or an array at the root, and nested values inside. End visual description.

[00:00:19] So at the root of your tree, you would have either an object or an array.

[00:00:25] And then of course, um if it were an array nested inside the array, you could have other values that could be any Json primitive or other objects and arrays.

[00:00:35] Um If the root level value is an object, of course, you can on that object, you can have properties that have values that are any primitive type or objects or other arrays.

[00:00:45] So essentially with JSON arrays and objects, you can arbitrarily nest um your data as needed.

[00:00:52] And so you can build up very complex data structures just by nesting arrays and objects inside of each other.

[00:00:57] And, and, and if you really think about that, all you're really doing is describing a tree of, of nested objects or nodes um in a textual format.

[00:01:08] So uh a Dom parser or a tree parser is a parser that when you ask it to parse your JSON file, what it does is it parses the whole file in one method call.

Start visual description. The professor demonstrates how a DOM parser works by parsing the entire JSON file in one method call and returning a data structure that mimics the data in the JSON file. The professor shows an image of a tree structure with an object at the root and properties like "query" and "results." End visual description.

[00:01:20] And what it does is it returns to you a data structure that looks like this, this image that I made.

[00:01:27] And so what you can see here is that this tree structure.

[00:01:32] Um Now this is not the CD catalog example, but the structure of this this in memory data structure directly mimics the data that’s in the JSON file that was parsed.

[00:01:43] And so you can see in this particular file, there was an object at the root and that object had two properties query and results.

[00:01:51] And the query property has a string value but the results property has an array value and this array has two objects inside of it.

[00:02:01] And of course, those objects can have their own properties and so forth and so on.

[00:02:06] And so in the Dom parser library, they actually have classes with names very similar to these.

[00:02:13] So there’s a Jason object class, a JSON array class A JSON value, class A JSON string A JSON number.

[00:02:21] There’s, there’s all kinds of classes that just represent the different kinds of, of values that can appear in a JSON file.

[00:02:29] And so the, the, the convenient thing about a tree parser or a dam parser is that you just call one method on the, the parser and it, it parses the whole file and then it just returns the whole file in tree form to you. And then in your program, you can write algorithms that traverse this tree to extract whatever data that you care about.

[00:02:51] And so you don't have to do any parsing at the token level.

[00:02:54] This is a much higher-level parcel that just gives you back the whole tree in a very convenient form that you can then process in your code.

[00:03:02] And so that's what a Dom parser does.

[00:03:05] So this would be much more commonly used than a streaming parser.

[00:03:09] Now, as a code example, let's go look um back at the sample code for the lecture and we'll look at the um the Dom parser example.

[00:03:26] So in this case, we are still parsing our CD catalog.

Start visual description. The professor demonstrates the process of parsing a CD catalog JSON file using a DOM parser. The professor shows the sample code that returns a list of CDs, highlighting the shorter and more efficient code compared to a streaming parser. End visual description.

[00:03:32] And again, our goal is to parse through this file and create a list of java CD objects and return it.

[00:03:41] And so if we go to our sample code here, you'll see it essentially, it looks similar to the previous example where we return a list of CDs and the input is the JSON file.

[00:03:52] But you'll notice here that the code is much shorter than it was with the streaming parser.

[00:03:56] And so that reflects the fact that it’s just easier to do it this way.

[00:04:00] So what we do to use a streaming parser is we open the JSON file again and we wrap that file in a buffer reader for efficiency.

[00:04:10] And then what we do is we create something called a token.

[00:04:14] Now, in this library, the token is essentially the same thing as the streaming parser.

[00:04:19] And so we’re going to create a, a streaming parser object and pass it our file.

[00:04:26] So you can imagine the a tree parser is built on top of a streaming parser.

[00:04:32] If you think about trying to build a tree data structure from a Jason file underneath, you could probably make use of the streaming parser to parse the tokens out of the file.

[00:04:41] And that’s typically how this works.

[00:04:42] So even though we’re using a tree parser, we still need to create a, a streaming parser object.

[00:04:47] And then when, when we parse the file, we pass in the streaming parser.

[00:04:52] And uh that’s how the, the tokens get parsed.

[00:04:55] And so in this particular library that we’re using, which um is yet another um parser library org dot JSON.

[00:05:09] So you can see what we do is to parse that whole file. We, we create a new Json object.

Start visual description. The professor demonstrates the creation of a new JSON object at the root of the CD catalog file. The professor shows the code that *creates a JSON object and passes it to the tokenizer, running the DOM parser on the entire file. End visual description.*

[00:05:15] And so we happen to know that at the root of our CD catalog file, we have adjacent object.

[00:05:20] And so that's why we're creating adjacent object here.

[00:05:22] If it were an array at the root of the file, we would create a Jason array.

[00:05:27] But in this case, we know it's an object.

[00:05:28] So we create a Jason object and then we just pass it to tokenizer, and this runs the, the tree parser or Dom parser on the whole file.

[00:05:37] And so what I get back um is a pointer to the Jason object which is at the root.

[00:05:42] So essentially the constructor on this Jason object class runs the, the Dom parser on the file and gives you back a pointer to the whole data structure or the whole tree.

[00:05:53] So once I get back that, that root object, which we call it here, then I can use methods to extract the properties of an object. So, in this case, I can, I can go to the root object, and I can say, OK, get the catalog property which I happen to know is an array.

[00:06:10] And so I get back a Jason array object and that has my array of CD wrapper objects.

[00:06:17] And then I can just iterate over the length of that array. And for each element in that array, I can go ahead and extract the wrapper object that just has the CD property.

[00:06:29] And then from that wrapper object, I can extract the actual Jason object that contains the CD properties.

[00:06:35] Then once I have the CD object, then I can extract the title and artist, et cetera and get all those properties.

[00:06:40] And I can just create a new java CD object and add it to my list.

[00:06:45] So this is far easier than using a, a, a streaming parser.

[00:06:50] The disadvantage of a tree parser is it does parse the whole file at once.

[00:06:55] And so if you have a very large file, it’s going to create a very large tree and return it.

[00:07:02] But usually you do want all the data in the file or oftentimes.

[00:07:06] And so this is a good way to parse it.

[00:07:08] If you only wanted to extract a few selected pieces of information from the file, the streaming parcel would probably be a better choice because it wouldn’t spend you all the extra work and uh expense needed to build that tree.

[00:07:21] So you just have to think about what am I trying to do with the data? And what would the right kind of parc be to use on that data? So, tree parcels are great.

[00:07:31] Now, the other thing is if you need to generate some JSON, let’s say I want to take a CD catalog and um I want to convert it to JSON.

Start visual description. The professor demonstrates how to generate JSON data by building a tree structure in RAM. The professor shows the code that creates a new JSON object with nested properties and calls the "toString" method to convert it to JSON. End visual description.

[00:07:44] Well, one thing you can do to create Jason is you can, you can actually in your code, you can build the tree.

[00:07:54] So this is the tree that the parser returns. But if you want to create JSON data, you can actually build a tree that looks like this.

[00:08:01] So you can create a new adjacent object that has these properties on it.

[00:08:06] And you can nest the Jason objects and Jason arrays and just build up a data structure in RA M that represents the data that you want to write. And then on the route, you basically just call two string and the two string method would convert it to Jason.

[00:08:22] And so that's an easy way to generate some JSON data if you want, if you don't mind building a tree structure first and then just call two string on that tree and it'll, it'll save it to JSON.

[00:08:36] And so if we go back to our code here, um in this example, I've got the root object that I originally got back from the, the parser.

[00:08:49] But if I wanted to turn around and actually save that, that tree back out to, to Jason, all I have to do is call the two-string method on it.

[00:08:56] I can pass the indentation level that I want to use. So, in this case, I want it to indent indents to be four spaces. If you don't pass in an indent factor, then it doesn't indent it at all.

[00:09:08] And I get back a Jason string and I could take that string and save it to a file or, or whatever.

[00:09:15] And so not only can we parse using a tree parser, but we can also generate JSON data using a, a tree as well.