Sri Lanka Institute of Information Technology

APPLICATION FRAMEWORKS

JAVASCRIPT, AND NOSQL

LECTURE 03

Faculty of Computing
Department of Software Engineering
Module Code: SE3040

Agenda

1. JavaScript
2. NoSQL

JAVASCRIPT

- Introduction
- Classes, objects
- How ‘this’ acts
- Function closure
- Callbacks and promises

JAVASCRIPT

- By default, JavaScript programs run using a single thread. Though there are ways to create new threads JavaScript is considered as Single threaded language.
- JavaScript does not wait for I/O operations to get completed, instead it continues the execution of the program. This is called as non-blocking I/O.
- JavaScript is asynchronous because of the NIO nature.
- JavaScript is dynamically typed. It determines variable types, ordering etc. in runtime.
- JavaScript support OOP as well as functional programming (Multi-paradigm).
- JavaScript has an eventing system which manages it’s asynchronous operations.

https://www.youtube.com/watch?v=wB9tIg209-8&t=261s

CLASSES AND OBJECTS

- In JavaScript, a constructor function is used with the ‘new’ key keyword when creating a Object.
- Constructor function is just another function.
- When function is used with ‘new’ keyword that function acts as a Class.
- Recently JavaScript introduced ‘class’ keyword, but it is not yet adopted by all JavaScript engines.
- Another way of creating a object is using object literals (‘{}’). These objects are considered to be singleton.
- JavaScript supports static methods and variables.
- When ‘new’ keyword is used, new object is created and it assigned as ‘this’ for the duration of call to the constructor function.

"This" IN JAVASCRIPT

- Unlikely other languages in JavaScript ‘this’ keyword acts differently.
- Inside a object ‘this’ refers to object itself.
- In global context ‘this’ refers to global object (in browser it is the window object). This behaviour will get changed in strict mode.
- If a function which is using ‘this’ keyword is being passed to another object then ‘this’ will refers to that object, but not to the original object where function was declared at first place.
- This behaviour is very noticeable in callback and closures.

CLOSURE

- JavaScript closure is a function which returns another function. In other words, a closure gives you access to an outer function's scope from an inner function.
- In JavaScript closure is used to encapsulate variables into a function and restrict access to it from the outside.
- JavaScript creates an environment with all the local variables from the outer function when the inner function is created. Closure is the combination of this environment and the inner function.

CALLBACK AND PROMISES

- JavaScript is asynchronous. All I/O operations in JavaScript is implemented to be asynchronous by nature.
- Reason for this is JavaScript being a single threaded language if an I/O operations holds the thread till it get completed JavaScript won’t perform well as a programming language.
- But asynchronous operation introduce difficulty when we need to do synchronous processing using data.
- This is solved by using callbacks and promises.
- Callback is a function that is being passed to an async task and on completion the function will be executed.
- Promise is an object that is being returned from async tasks. Promise have properties to deal with async operations synchronously.

CALLBACK AND PROMISES... (CNT)

- Nested callbacks passed into sequence of async tasks is referred to a ‘callback hell’.
- Promise object was introduced to solve this problem.
- Promise object has a set of properties, methods and mechanism of chaining to handle complex async tasks nicely.

ACTIVITY

Objective:

Learn to work with JavaScript Promises to handle asynchronous operations.

Tasks:

1. Create a Promise: Write a function that returns a Promise which resolves if a condition (e.g., a number comparison) is true, and rejects if false.

index.js > ...

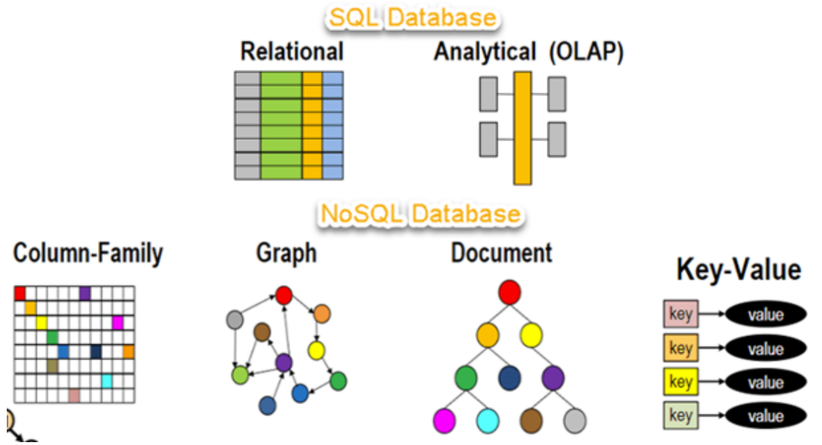
```
const waitPromise = new Promise((resolve, reject) => {
    const success = Math.random() >= 0.5; // randomly determine whether to fulfill or reject the promise
    setTimeout(() => {
        if (success) {
            resolve('Waited for 3 seconds!');
        } else {
            reject('An error occurred while waiting for 3 seconds!');
        }
    }, 3000);
});

waitPromise.then(msg => {
    console.log(msg);
}).catch(err => {
    console.error('Error:', err);
});
```

```
Formatter  Formatting completed in 7319ms.                    7s on 10:50:05, 02/24 ✓
Run  Waited for 3 seconds!                                    3s on 10:52:45, 02/24 ✓
Run  Error: An error occurred while waiting for 3 seconds!    3s on 10:53:14, 02/24 ✓
Run  Error: An error occurred while waiting for 3 seconds!    3s on 10:54:58, 02/24 ✓
Run  Waited for 3 seconds!                                    3s on 10:55:07, 02/24 ✓
Run  Waited for 3 seconds!                                    3s on 10:55:16, 02/24 ✓
Run  Waited for 3 seconds!                                    3s on 10:55:21, 02/24 ✓
```

NOSQL

- What?
- Why not SQL?
- Strengths and weaknesses
- MongoDB



WHAT?

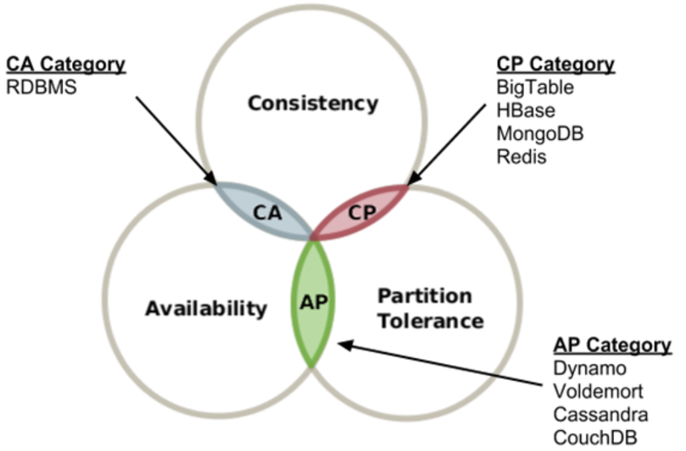
- Non relational (mostly), Schema free.
- Distributed.
- Horizontally scalable.
- Easy replication.
- Eventually consistent.
- Open source (mostly).
- No transaction support.

WHY?

- Remove the burden of data structures mismatch between application in-memory and relational databases.
- Integrate databases using services. (ElasticSearch).
- Relational databases not designed to run efficiently on clusters.
- Aggregate oriented databases, are easier to manage inside clusters and based on the domain driven design. (Order details inside the order).
- But inter aggregate relationships are harder to manage.

CAP THEOREM

- **Consistency** - Every read receives the most recent write.
- **Availability** - Every request receives a response (no guarantee that it is the most recent write).
- **Partition tolerance** - System will continue to operate despite number of messages lost among network nodes.



TYPES

- Key-Value - Stores data as key value pairs.
  - Ex: Redis, Riak, Memcached.
- Document - Stores data as documents (JSON, BSON, XML) in maps or collections.
  - Ex: MongoDB
- Column Family - Store data in column families as rows that have many columns associated with.
  - Ex: Cassandra
- Graph - Store entities(nodes) and relationships(edges) between them and represent it in a graph.
  - Ex: Neo4j

MongoDB

- NoSQL document database.
- Strong query capabilities with aggregations using JavaScript.
- Use SpiderMonkey JavaScript engine.
- High availability with replica sets.
- Reads and writes on primary by default.
- Eventually consistent on secondary instances.
- In built file storage called Grid File System.

MongoDB queries

- Insert
- Find
- Update
- Remove

THAT'S ALL FOLKS !

ANY QUESTIONS ?