Database & REST Service & Postman

Question: **what is endpoint(API)** ?

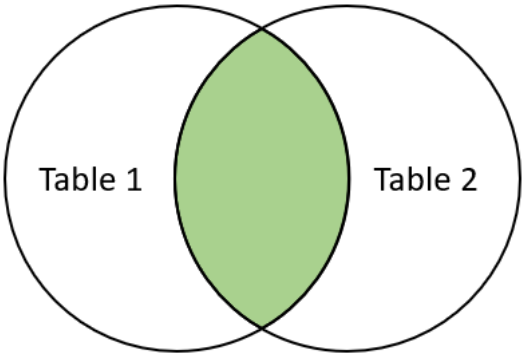
Database(CRUD Operation)

MySQL + (MySQL Workbench)

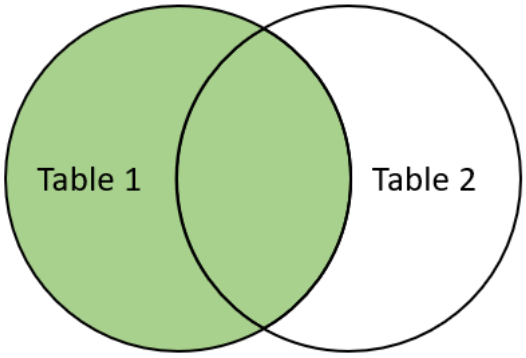
```
--Created DB Schema
CREATE SCHEMA `ChuwaTest` DEFAULT CHARACTER SET utf8 ;

--Created DB Table
```

```sql
CREATE TABLE `Chuwa`.`User` (
  `id` INT NOT NULL,
  `firstName` VARCHAR(100) NULL,
  `lastName` VARCHAR(100) NULL,
  `Address` VARCHAR(200) NULL,
  PRIMARY KEY (`id`));

--Insert
INSERT INTO `Chuwa`.`User` (`id`, `firstName`, `lastName`, `Address`) VALUES ('1', 'Charles', 'C', '123 Fake Address');

--Select
SELECT * FROM Chuwa.User;

```

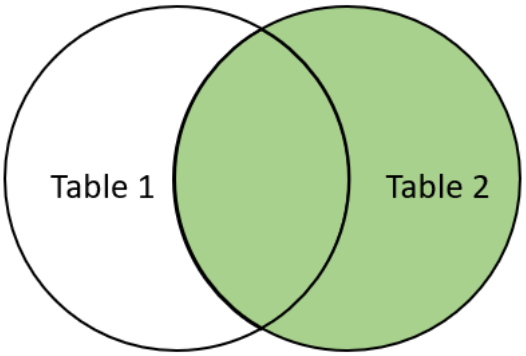
4 types of Joins



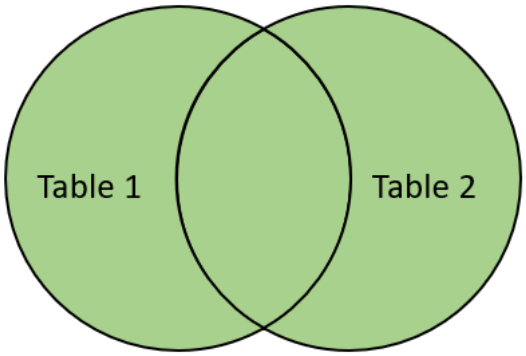
INNER JOIN



LEFT JOIN



RIGHT JOIN



CROSS JOIN

NoSQL - MongoDB/Cassandra + Compass

```
{ //e.g. Employee collection/table in JSON - Format
  _id: ,
  Emp_ID: "10025AE336"
  Personal_details:{
    First_Name: "Radhika",
    Last_Name: "Sharma",
    Date_Of_Birth: "1995-09-26"
    Maiden_name: "C" //No need to alter DB/Table structure
    Address: {
      city: "Hyderabad",
      Area: "Madapur",
      State: "Telangana"
    }
  },
  Contact: {
    e-mail: "radhika_sharma.123@gmail.com",
    phone: "9848022338"
  },
  Address: {
    city: "Hyderabad",
    Area: "Madapur",
    State: "Telangana",
  }
}

--GrahQL
{ //e.g. Employee collection/table in JSON - Format
  _id: ,
  Emp_ID: "10025AE336"
  Personal_details:{
    First_Name: "Radhika",
    Last_Name: "Sharma",
  }
}
```

```
--Creaet DB
use my_database

--show list of DBs
show dbs

```

```
--delete DB
db.dropDatabase()
{ "dropped": "my_database", "ok": 1}


--create collection (aka Table)
db.createCollection(collection_name, options)
db.createCollection(<collection_name>, { capped: <boolean>,
                              autoIndexId: <boolean>,
                              size: <number>,
                              max: <number>,
                              storageEngine: <document>,
                              validator: <document>,
                              validationLevel: <string>,
                              validationAction: <string>,
                              indexOptionDefaults: <document>,
                              viewOn: <string>,
                              pipeline: <pipeline>,
                              collation: <document>,
                              writeConcern: <document>} )

e.g.
db.createCollection("MyCollection")

--this will automatically create `movie` collection
db.movie.insert({"name":"Avengers: Endgame"})

```

More Syntax learning

SQL vs. NoSQL** (interview common question)

SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (**RDB-MS**)	Non-relational or **distributed** database system.
These databases have fixed or **static** or predefined **schema** (有网格的本子)	They have **dynamic schema** (无网格的本子)
These databases are **NOT** suited for **hierarchical** data storage.	These databases are best suited for **hierarchical data** storage. (JSON style)
These databases are best suited for **complex queries**	These databases are **NOT so good** for complex queries(很痛苦)
Vertically Scalable(扩容**单台服务器**. e.g 32g -> 128G) is easier fo SQL	**Horizontally Scalable** (多台服务器扩容) (e.g. 一台--> 多台)
Follows **ACID** property	Follows **CAP**(consistency, availability, partition tolerance)
Examples: MySQL, PostgreSQL, Oracle, MS-SQL Server,SQLite(In memory DB) etc.	**Examples:** MongoDB, HBase, Neo4j, Cassandra etc.

Question: **In your project, which one did you use ? And why ?**

- **QPS** (Query Per Second) (e.g. 10k QPS)

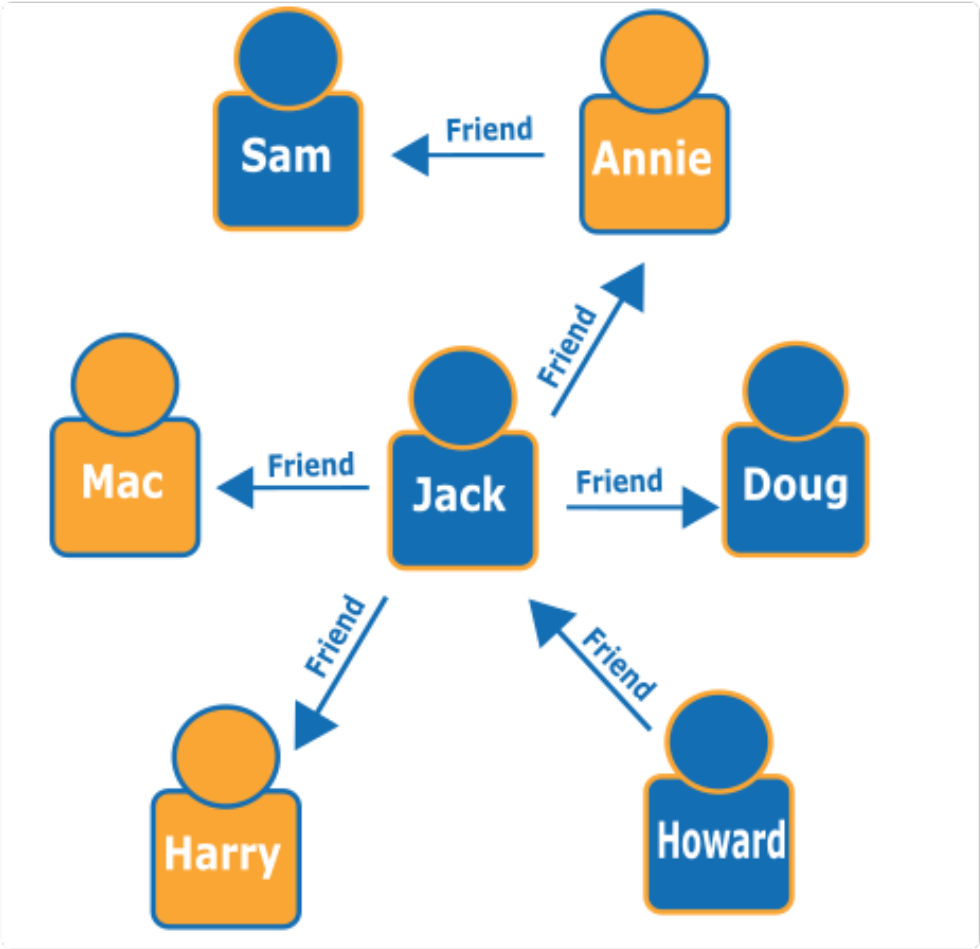
No-SQL becomes more and more popular.

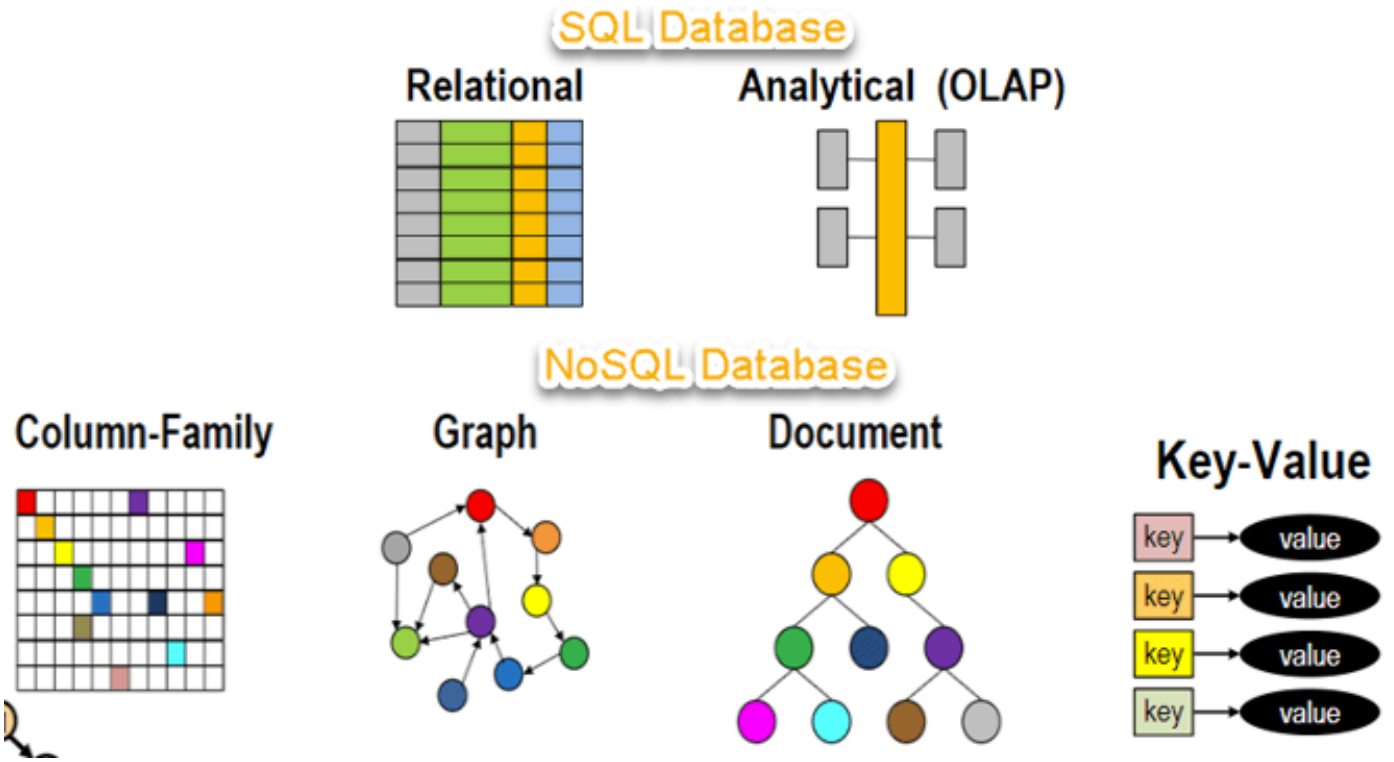
Graph Database

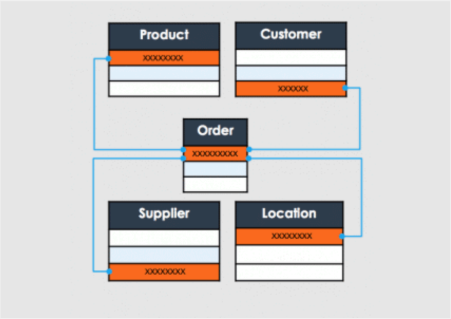
Key concept: Node + Edge

- **Neo4j**
- TigerGraph
- Amazon Neptune

(DFS + BFS)


Sam
Friend
Annie
Friend
Mac
Friend
Jack
Friend
Doug
Friend
Friend
Harry
Howard

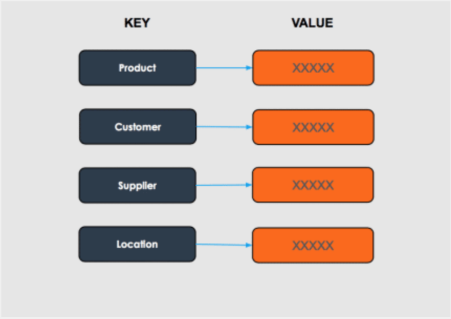




Relational Database

COMPLEX, SLOW, TABLE JOINS REQUIRED

- Rigid schema
- High performance for transactions
- Poor performance for deep analytics



Key-Value Database

MULTIPLE SCANS OF MASSIVE TABLE REQUIRED

- High fluid schema/no schema
- High performance for simple transactions
- Poor performance for deep analytics

Graph Database

PRE-CONNECTED BUSINESS ENTITIES – NO JOINS NEEDED

- Flexible schema
- High performance for complex transactions
- High performance for deep analytics

NoSQL vs SQL

	Funct. Req.								Non-Funct. Req.										Techniques																			
	Scan Queries	ACID Transactions	Conditional Writes	Joins	Sorting	Filter Queries	Full-Text Search	Analytics	Data Scalability	Write Scalability	Read Scalability	Elasticity	Consistency	Write Latency	Read Latency	Write Throughput	Read Availability	Write Availability	Durability	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared-Disk	Transaction Protocol	Sync. Replication	Async. Replication	Primary Copy	Update Anywhere	Logging	Update-in-Place	Caching	In-Memory Storage	Append-Only Storage	Global Indexing	Local Indexing	Query Planning	Analytics Framework	Materialized Views
MongoDB	x		x		x	x	x	x	x	x	x		x	x	x	x	x		x	x	x					x	x		x			x	x			x	x	x	
Redis	x	x	x								x		x	x	x	x	x		x								x	x		x		x							
HBase	x		x		x			x	x	x	x	x	x	x		x			x	x						x		x		x		x		x					
Riak							x	x	x	x	x	x		x	x	x	x	x	x		x		x				x		x	x	x	x			x	x		x	
Cassandra	x		x		x		x	x	x	x	x	x		x		x	x	x	x		x		x				x		x	x		x		x	x	x			x
MySQL	x	x	x	x	x	x	x	x			x		x						x					x	x		x	x		x	x	x				x	x		

Table 1: A direct comparison of functional requirements, non-functional requirements and techniques among **MongoDB**, **Redis**, HBase, Riak, **Cassandra** and **MySQL** according to our NoSQL Toolbox.

Recap Reasons for SQL:

- **Structured data**
- **Strict schema**
- **Relational data**
- Need for **complex joins**
- Transactions
- Clear patterns for scaling (?)
- More established: developers, community, code, tools, etc
- **Lookups by index** are very fast;

Recap Reasons for NoSQL: (easier for Scale)

- Semi-structured data
- **Dynamic or flexible schema(implicit schema)**
- Non-relational data
- No need for complex joins
- **Store many TB (or PB) of data**
- Very data intensive workload
- **Very high throughput for IOPS** (**QPS**)

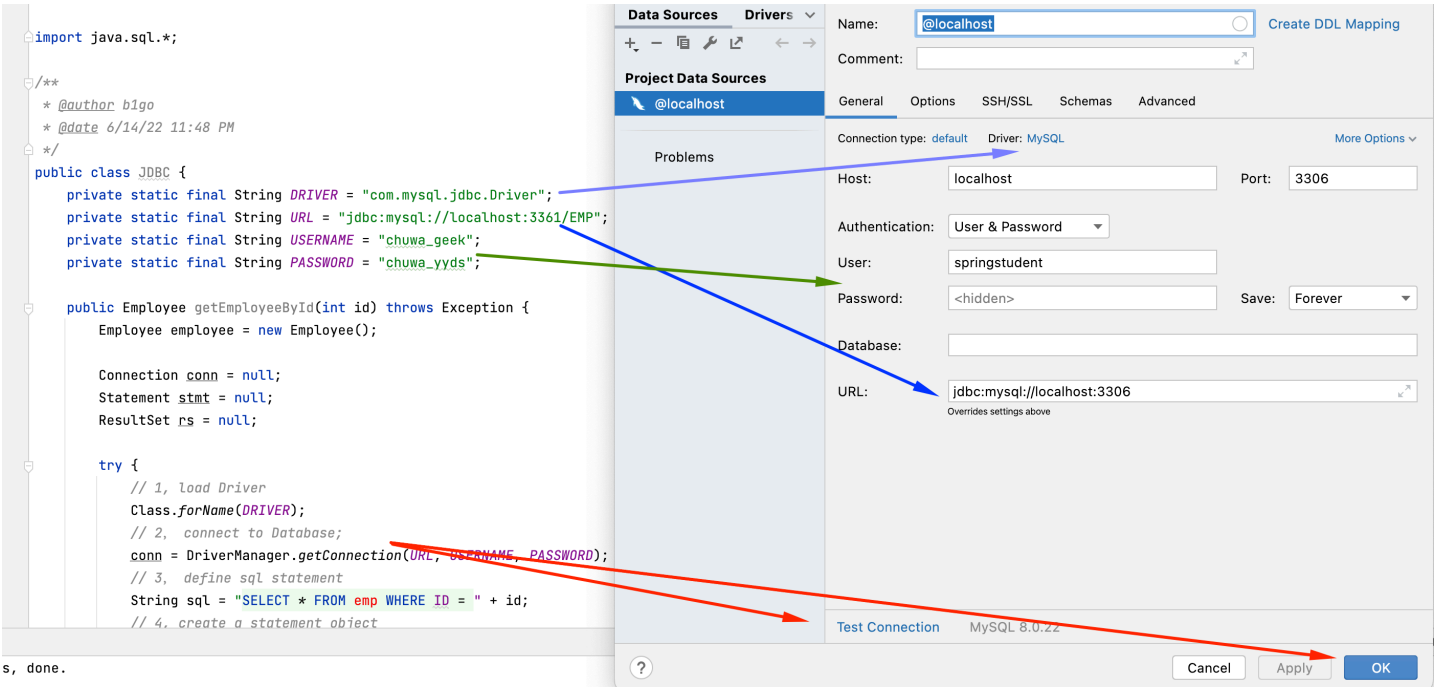
JDBC(Java Database Connectivity)

DataSource:

- Driver
- URL,
- Username,
- Password

跟手动连接数据库和获取数据的步骤差不多，只是变成了Java语句。

Connecting database using Java and Database client



Basic flow:

1. load Driver
2. connect to Database using username, password and url
3. define SQL statement
4. use stmt object to execute sql statement and use stmt object to execute sql statement;
5. get ResultSet's data and set them to java object(employee)
6. close conections and other resource.

Notice `try... catch...catch...finally`

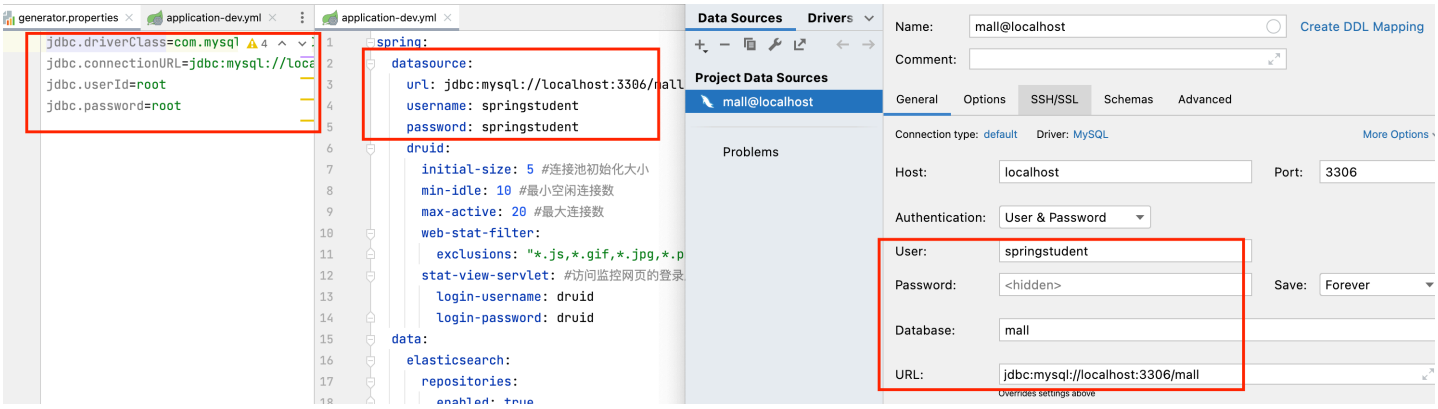
Catch 范围从小到大。finally关闭已打开的资源。

```
public class JDBC {
```

```java
public class JDBC {
    private static final String DRIVER = "com.mysql.jdbc.Driver";
    private static final String URL = "jdbc:mysql://localhost:3361/EMP";
    private static final String USERNAME = "chuwa_geek";
    private static final String PASSWORD = "chuwa_yyds";

    public Employee getEmployeeById(int id) throws Exception {
        Employee employee = new Employee();

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // 1, load Driver
            Class.forName(DRIVER);
            // 2,  connect to Database;
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            // 3,  define sql statement
            String sql = "SELECT * FROM emp WHERE ID = " + id;
            // 4, create a statement object
            stmt = conn.createStatement();
            // 5, use stmt object to execute sql statement;
            rs = stmt.executeQuery(sql); // the result is return to ResultSet

            while(rs.next()) {
                // 6, get ResultSet's data to java object(employee)
                employee.setId(rs.getInt("id"));
                employee.setName(rs.getString("name"));
            }

            // 7, close conections and other resource.
            rs.close();
            stmt.close();
            conn.close();

            return employee;
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (rs != null) {
                rs.close();
                rs = null;
            }

            if (stmt != null) {
```

```
            if (stmt != null) {
                stmt.close();
                stmt = null;
            }
            if (conn != null) {
                conn.close();
                conn = null;
            }
        }

        return null;
    }
}
```

Connecting to database in SpringBoot




```
List<User> users = new ArrayLsit<>();
user1 <-  database;
users.add(user1)
```

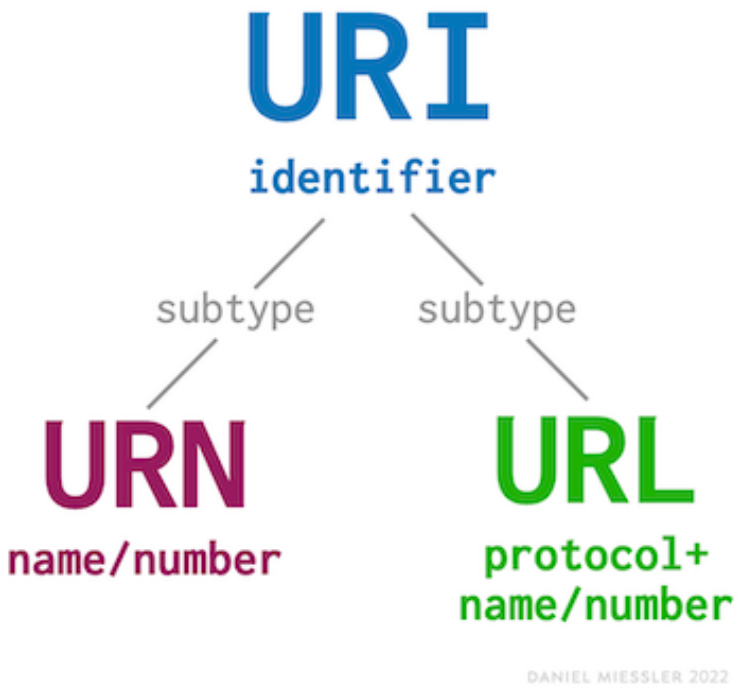
REST API(End-Point)

"Java Interface"

End-Point --> End to End Point

URI, URL, URN (Unique Resource)

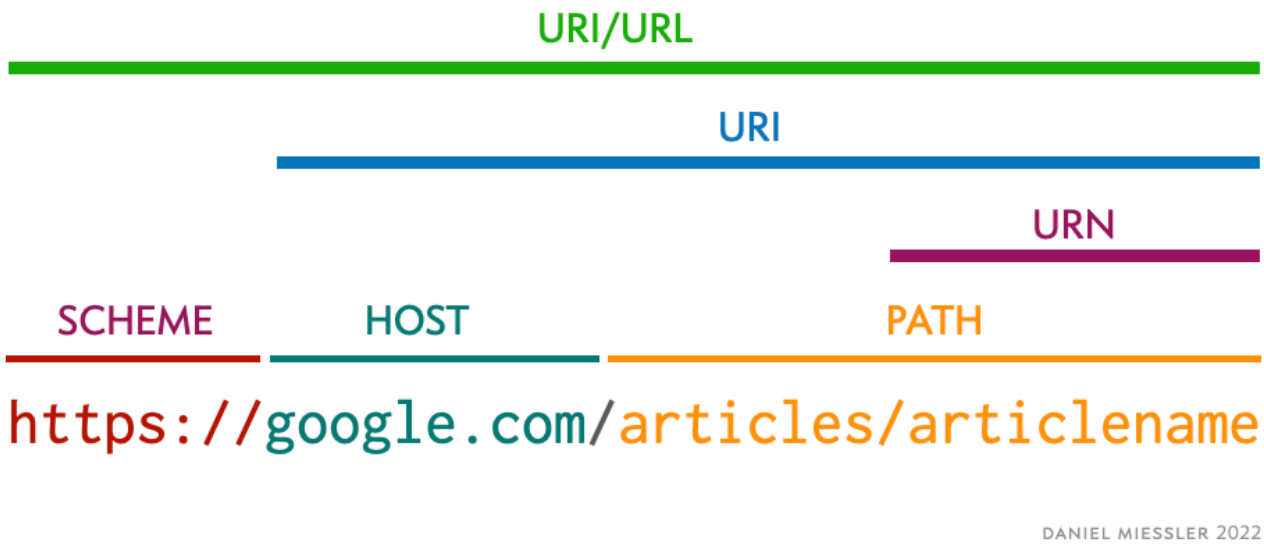
> URN: **google.com**
>
> URL: https://google.com

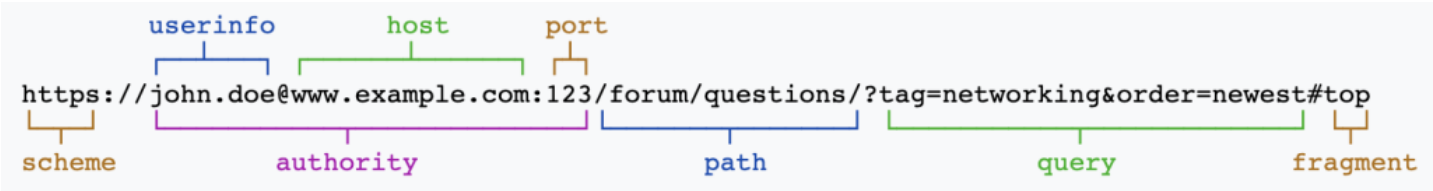


URI

- URN
- protocol + URN
  - **http://** (Web/APIs)
  - **Https://** (Secured Web/APIs)
  - ftp:// (for file transfer)
  - mailto:// (for email transfer)

URL Structure





Path Variblae and Request Parameter

```
http://localhost:8080/userapp/users/{id}/load?minAge=20&lastName=Stark;
http://localhost:8080/userapp/users/1234/load?minAge=20&lastName=Stark;

http://localhost:8080/userapp/users/load?userid=1234&minAge=20&lastName=Stark;

http://localhost:8080/hotel/filter?location=???&price=???&nearAiprot=?
```

Path Varibale: **2**

Request Parameter: ?**key1**=value1&**key2**=value2

Question: when do we use **path variable**? when do we use **request parameter**?

Path Variable examples

> https://drive.google.com/drive/u/0/my-drive?fileName=Chuwa
>
> https://drive.google.com:443/drive/u/1/my-drive

> https://drive.google.com/drive/u/0/computers
>
> https://drive.google.com/drive/u/0/my-drive
>
> https://drive.google.com/drive/u/0/shared-with-me
>
> https://drive.google.com/drive/u/0/recent

> https://drive.google.com/drive/u/0/recent
>
> https://drive.google.com/drive/u/0/starred
>
> https://drive.google.com/drive/u/0/trash

Question: **what is the path varibale in the below url**?

> https://docs.google.com/document/d/1m71_YP-V6E4232323-KMVNgkiLr72d6qPID1aWjI9Y28/edit
>
> /api/v1/document/d/{id}/edit
>
> /api/v1/document/d/{id}/view

Request Parameter (there is a lenght limit, SD:"TinyURL")

> https://www.google.com/search?q=chuwa&oq=chuwa&aqs=chrome..69i57j69i60l4.1261j0j4&sourceid=chrome&ie=UTF-8

HTTP Methods and Status Code

HTTP Verb/Type	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	**Create** & Save	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	**404 (Not Found)**, 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/**Replace** (e.g. replace first name)	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/**Modify** (e.g. modifiy address)	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Question:

- what is the difference between PUT and POST ?
- what is the difference between PUT and PATCH ?

STATUS CODE	

STATUS CODE RANGE	MEANING
100 – 299	1XX Informational Responses. For example, 102 indicates the resource is being processed. 2XX **success** status. E.g. 200/204 etc. (**Mostly okay**)
300 – 399	Redirects For example, 301 means **Moved permanently** (e.g. redirect)
400 – 499	Client-side errors 400 means bad request and 404 means **resource** not found (**applicaiton error**)
500 – 599	Server-side errors For example, 500 means an internal **server error** (e.g. 503)

REST Resource Naming Guide/Convention

独孤九剑： 并不一定能懂原理或专业词汇，但是见多了，用到的时候自然而然就会了。

原理不懂就当多看例子，多见例子很有用

1. Use **JSON** as the Format for Sending and Receiving Data
2. Use **Nouns** Instead of **Verbs** in Endpoints
    1. `POST  /api/Users` `//NOT /api/createUser`
    2. `GET  /api/users/{id}` - `/api/getUser WRONG`
    3. `DELETE /api/users/{id}` - `/api/deleteUser  WRONG`
    4. `UPDATE/PATCH/PUT /api/users/{id}` -> `UPDATE /api/users/101`
3. Use **Status Codes** in Error Handling
4. Use **nouns** to represent resources
    1. > http://api.example.com/device-management/managed-devices
       >
       > http://api.example.com/device-management/managed-devices/{device-id}
       >
       > http://api.example.com/user-management/users
       > http://api.example.com/user-management/users/{id}
    2. document
        1. > http://api.example.com/device-management/managed-devices/{device-id}
           >
           > http://api.example.com/user-management/users/{id}
           >
           > http://api.example.com/user-management/users/admin
        2. What is the document in the above google drive url example?
    3. collection
        1. > http://api.example.com/device-management/managed-devices

> http://api.example.com/user-management/users
>
> http://api.example.com/user-management/users/{id}/accounts

4. store
    1. > http://api.example.com/song-management/users/{id}/playlists
5. controller
    1. > http://api.example.com/cart-management/users/{id}/cart/checkout
       >
       > http://api.example.com/song-management/users/{id}/playlist/play

5. **Consistency is the key**
    1. Use forward **slash (/)** to indicate hierarchical relationships

```
http://api.example.com/device-management
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices/{id}
http://api.example.com/device-management/managed-devices/{id}/scripts
http://api.example.com/device-management/managed-devices/{id}/scripts/{id}
```

    2. Do NOT use **trailing forward slash (/)** in URIs

```
http://api.example.com/device-management/managed-devices/
http://api.example.com/device-management/managed-devices  /*This is much better version*/
```

    3. Use **hyphens (-)** to improve the readability of URIs

```
http://api.example.com/device-management/managed-devices/
http://api.example.com/device-management/managed-devices
/*This is much better version  then `manageddevices`*/
```

    4. Do not use underscores (_)

```
http://api.example.com/inventory-management/managed-entities/{id}/install-script-location  //More readable

http://api.example.com/inventory-management/managedEntities/{id}/install_Script_Location  //Less readable
```

    5. Use lowercase letters in URIs

```
http://api.example.org/my-folder/my-doc       //1
HTTP://API.EXAMPLE.ORG/my-folder/my-doc     //2
http://api.example.org/My-Folder/my-doc       //3
```

6. Do not use file extensions

```
http://api.example.com/device-management/managed-devices.xml  /*Do not use it*/

http://api.example.com/device-management/managed-devices  /*This is correct URI*/
```

7. Never use CRUD function names(or Verbs) in URIs

```
HTTP GET http://api.example.com/device-management/managed-devices  //Get all devices
HTTP POST http://api.example.com/device-management/managed-devices  //Create new Device

HTTP GET http://api.example.com/device-management/managed-devices/{id}  //Get device for given Id
HTTP PUT http://api.example.com/device-management/managed-devices/{id} //Update device for given Id
HTTP DELETE http://api.example.com/device-management/managed-devices/{id} //Delete device for given Id
```

8. Use query component to filter URI collection

```
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices?region=USA
http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ
http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ&sort=installation-date
```

   2. Paging

```
https://api.spotify.com/v1/artists/12vsllerkjdsasjc/albums?album_type=SINGLE&offset=20&limit=10
```

9. Be Clear with **Versioning** (mobile app -- **forward/backward compatibale** APIs)

```
https://mysite.com/api/v1/posts (user without address)
https://mysite.com/api/v2/posts (user with address, creditCard info)
```

Design

Design

design 3 APIs (思考： path variable 怎么用？ 有sub resources, 哪些地方该用复数)

1. find the customer's payments, like credit card 1, credit card 2, paypal, Apple Pay.
2. Find the customer's history orders from 10/10/2022 to 10/24/2022
3. find the customer's delievery addresses
4. If I also want to get customer's default payment and default delievery address, what kind of the API (URL) should be?

```
/api/v1/customers/amy/orders
/api/v1/customers/amy/payments
/api/v1/customers/amy/payments/{paymentTypeId}
  /api/v1/customers/chenyu/addresses

default payment:
/api/v1/customers/{user name/id}/default-payment
/api/v1/customers/{user}/payments/default (improved version)

default devlierry:
/api/v1/customers/{user name/id}/default-delivery
/api/v1/customers/{user}/addresses/default (improved version)

history orders from 10/10/2022 to 10/24/2022
/api/v1/orders/{userId}/daterange?start-date >= {} & end-date <= {}

/api/v1/customers/{user}/orders?start-date="10/10/2022"&end-date="10-24-2022"
(improved version)
```

Exercise (super useful)

Practice Request site: https://reqres.in/

Payload variables: https://developers.tackle.io/reference/payload-variables

Reference

- https://danielmiessler.com/study/difference-between-uri-url/
- https://restfulapi.net/resource-naming/
- https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/

GraphQL (RoR tutorial)

GraphQL (RoR tutorial)

面试几乎不不怎么考，但是现在越来越多的大厂开始采用, 简单介绍一下,如果有兴趣深入了解的可以自学。

(e.g. Github, Facebook/Meta, Airbnb, Shopify)

Understanding RPC, REST and GraphQL

RPC (Remote Procedure Call - with Http/AMQP) → Web APIs;

- **SOAP** is one of them; (using **WSDL** + **XML** vs. **JSON**(in REST))
- **gRPC** (modern) better SOAP, uses **ProtoBuff** (schema + data) kind like **WSDL**;

REST

- **JSON Schema** → inspired by XML-Schema;
- Can handle multiple content types **json/text**, **image/png** etc.
    - More **versatile** on this aspect;
- **Caching** is easier;

GraphQL (vs.Rest)

- RPC with good part from REST/HTTP;
- **Kind like SQL**;
- defaults to providing the very smallest response from an API
- You can only speak in terms of "**Fields**"
- **Better API evolution** support; (deprecation easy)

e .g.

`rails server` --> `http://localhost:3000/graphiql`

```
query {
  allFood{
    id
    name
  }
}

query{
  nutrition{
    id
    calories
    foodId
```

```
  }
}


Create:
mutation {
  foodCreate(input: {
    name: "Apple Pie Pie",
    placeOfOrigin: "US",
    image: "apple-pie.png"
  }) {
    id
    name
    placeOfOrigin
    image
  }
}

Update:
mutation {
  foodUpdate(input:{
    id:4,
    name:"super pumpkin pie !!!"
  }){
    id
    name
  }
}

Delete:
mutation {
  foodDelete(input: {
    id: 4
  }) {
    id
  }
}
```

Postman*

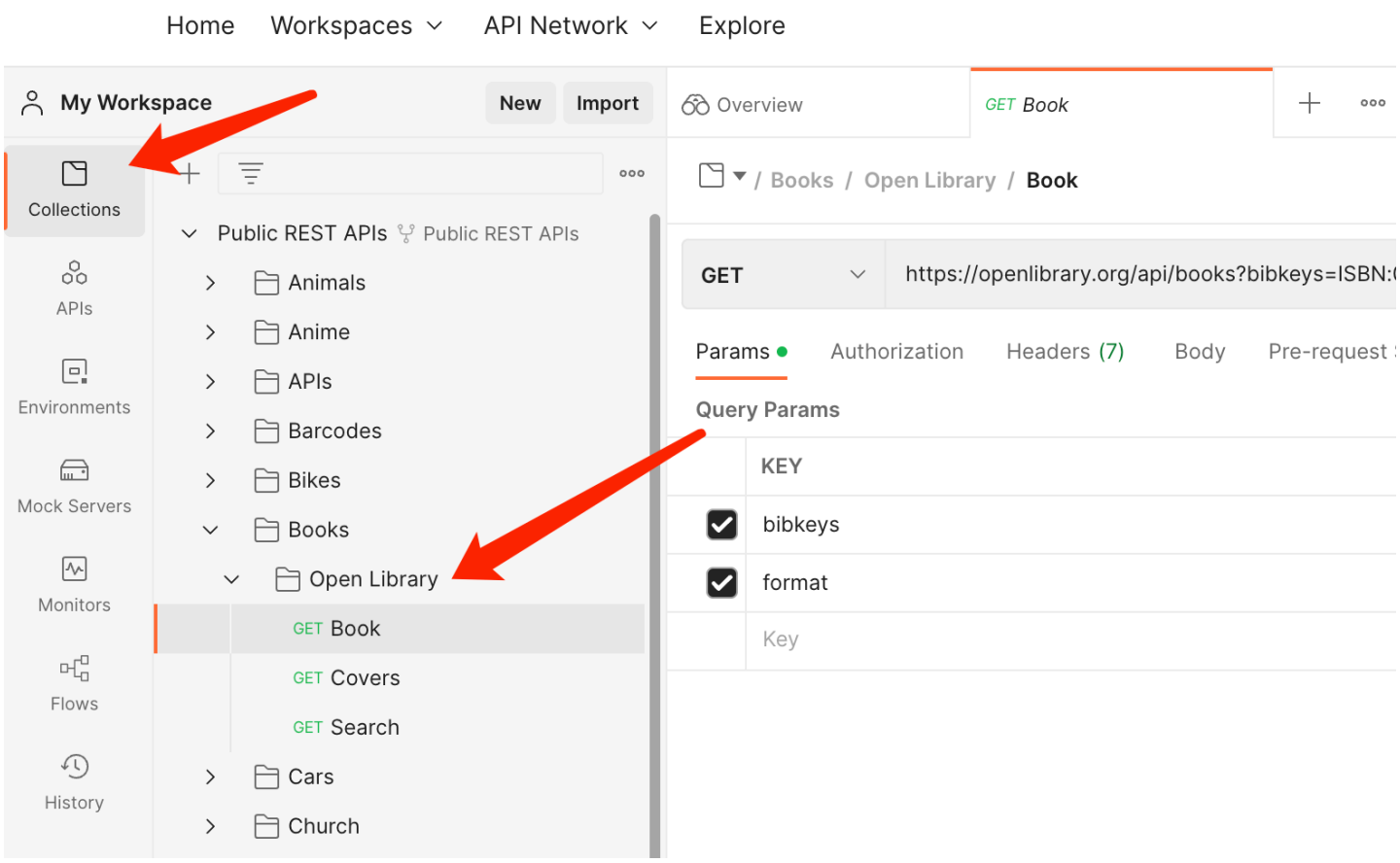
Public APIs

[Postman public API]

[Explore the World of APIs]

API 练习网站： https://reqres.in/

API Collection



Environments