EBook Gratis

APRENDIZAJE couchdb

Free unaffiliated eBook created from
Stack Overflow contributors.

#couchdb

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: couchdb

It is an unofficial and free couchdb ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official couchdb.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con couchdb

Observaciones

¿Por qué CouchDB?

CouchDB tiene un modelo de almacenamiento de documentos JSON con un potente motor de consulta basado en una API HTTP.

Al utilizar JavaScript en *los* documentos de *diseño* , tiene control sobre las formas de consultar, asignar, combinar y filtrar sus datos. Al proporcionar tolerancia a fallos, escalabilidad extrema y replicación incremental, CouchDB es una buena opción para una base de datos de documentos no relacional.

> Si bien una base de datos relacional tradicional requiere que modele sus datos por adelantado, el diseño sin esquemas de CouchDB le ofrece una forma poderosa de agregar sus datos después del hecho, tal como lo hacemos con los documentos del mundo real. Estudiaremos en profundidad cómo diseñar aplicaciones con este paradigma de almacenamiento subyacente.

Versiones

Versión	Fecha de lanzamiento
2.0.0	2016-09-20
1.6.1	2014-09-03

Examples

Instalación y configuración

Ubuntu

En las versiones recientes de Ubuntu, puede instalar una versión actualizada de CouchDB con `sudo apt-get install couchdb` . Para versiones anteriores, como Ubuntu 14.04, debe ejecutar:

```
sudo add-apt-repository ppa:couchdb/stable -y
sudo apt-get update
sudo apt-get install couchdb -y
```

Fedora

Para instalar couchdb en fedora ryou puede hacer `sudo dnf install couchdb`

Mac OS X

Para instalar CouchDB en Mac OS X, puede instalar la aplicación Mac desde la sección de descargas de CouchDB .

Windows

Para instalar CouchDB en Windows, simplemente puede descargar el ejecutable desde la sección de descargas de CouchDB .

Hola Mundo

De forma predeterminada, CouchDB escucha en el puerto 5984. Visitar `http://127.0.0.1:5984` producirá una respuesta como esta:

```
{"couchdb":"Welcome","version":"1.6.1"}
```

CouchDB viene de fábrica con una GUI llamada *Futon* . Puede encontrar esta interfaz en `http://127.0.0.1:5984/_utils` . Aquí, puede configurar fácilmente una cuenta de administrador y configurar otros ajustes importantes.

Lea Empezando con couchdb en línea: https://riptutorial.com/es/couchdb/topic/2649/empezando-con-couchdb

Capítulo 2: Documentos de diseño

Observaciones

Los documentos de diseño se comportan como todos los documentos en términos de revisiones, replicación y conflictos. También puede agregar archivos adjuntos para diseñar documentos.

Examples

_design / ejemplo

Los documentos de diseño contienen lógica de aplicación. Cualquier documento en una base de datos que tenga un _id que comience con "_design /" se puede usar como documento de diseño. Por lo general, hay un documento de diseño para cada aplicación.

```
{
    "_id": "_design/example",
    "view": {
        "foo": {
            "map": "function(doc){...};",
            "reduce": "function(keys, values, rereduce){...};"
        }
    }
}
```

El ejemplo anterior define una **vista** llamada *foo* , que se puede solicitar desde la siguiente ruta, asumiendo que la base de datos se llama *db* :

> http: // localhost: 5984 / db / _design / example / _view / foo

Lea Documentos de diseño en línea: https://riptutorial.com/es/couchdb/topic/6958/documentos-de-diseno

Capítulo 3: Ektorp java client

Observaciones

Prueba

Examples

Abrir una conexión a CouchDB

```
HttpClient httpClient = new StdHttpClient.Builder().
    url("http://yourcouchdbhost:5984").
    username("admin").
    password("password").
    build();

CouchDbInstance dbInstance = new StdCouchDbInstance(httpClient);
```

Conectando a una base de datos

Dado que tiene una instancia de CouchDbInstance válida, se conecta a una base de datos dentro de CouchDB de la siguiente manera

```
CouchDbConnector connector = dbInstance.createConnector("databaseName", true);
```

CRUD simple con POJOs

Una de las grandes cosas de Ektorp, es que proporciona funcionalidad similar a ORM, de manera inmediata. Este ejemplo lo guiará a través de la creación de un POJO simple y de realizar una operación CRUD estándar en él.

Creando un simple POJO

En primer lugar, definimos un POJO de la siguiente manera

```
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;

@JsonInclude(JsonInclude.Include.NON_NULL)
public class Person {
    @JsonProperty("_id") private String id;
    @JsonProperty("_rev") private String revision;
    private String name;

    public String getId() {
        return id;
```

```
    }

    public String getRevision() {
        return revision;
    }

    public String getName() {
        return name;
    }


    public void setId(String id) {
        this.id = id;
    }

    public void setRevision(String revision) {
        this.revision = revision;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Entonces, ¿qué está pasando aquí? La anotación `@JsonInclude(JsonInclude.Include.NON_NULL)` le dice a jackson que no `@JsonInclude(JsonInclude.Include.NON_NULL)` campos nulos en JSON. Entonces, por ejemplo, si la propiedad id es nula, no tendrá la propiedad id en el JSON resultante.

Además, las `@JsonProperty("_id")` y `@JsonProperty("_rev")` son directivas, que informan al serializador / unserializer a qué propiedades JSON asignan estos valores. Los documentos en CouchDB deben tener tanto un campo _id como un campo _rev, por lo tanto, todos los POJO que intenta conservar en CouchDB, deben incluir una identificación y propiedades de revisión como las anteriores. Usted es libre de nombrar sus propiedades de manera diferente en el POJO, siempre y cuando no cambie las anotaciones. Por ejemplo,

```
@JsonProperty("_id") private String identity;
@JsonProperty("_rev") private String version;
```

Persistiendo nuevas instancias a CouchDB.

Ahora, la creación de un documento nuevo, en la base de datos se realiza de la siguiente manera, suponiendo que tiene una instancia de CouchDbInstance válida y que desea conservar el documento en una base de datos llamada *person*

```
CouchDbConnector connector = dbInstance.createConnector("person", true);

Person person = new Person();
person.setName("John Doe");
connector.create(person);
```

Ahora, en este escenario, CouchDB creará automáticamente una nueva ID y Revisión para usted. Si no quieres esto, puedes usar

```
connector.create("MyID", person);
```

Cargando, actualizando y borrando documentos.

Suponiendo que tiene una instancia de CouchDBConnector lista, podemos cargar las instancias de nuestro POJO de la siguiente manera

```
Person person = connector.get(Person.class, "id");
```

entonces podemos manipularlo, y actualizarlo de la siguiente manera

```
person.setName("Mr Bean");
connector.update(person);
```

Tenga en cuenta que, si la revisión del documento en el sofá no coincide con la revisión del documento que está enviando, la actualización fallará, y debe cargar la última versión de la base de datos y combinar las instancias en consecuencia.

y finalmente, si deseamos eliminar la instancia, es tan simple como

```
connector.delete(person);
```

Lea Ektorp java client en línea: https://riptutorial.com/es/couchdb/topic/6417/ektorp-java-client

Capítulo 4: Puntos de vista

Examples

Vistas para personas

Para mostrarle cómo funcionan las vistas, asumiremos que queremos consultar el documento del tipo de *personas* . Para hacerlo, primero necesitaremos un documento de diseño que contendrá nuestras opiniones.

Nota: para el propósito del ejemplo, usaremos muchas vistas dentro de 1 documento de diseño. Por lo tanto, en un entorno de producción, es posible que prefiera tener 1 vista por documento de diseño. El motivo es que cada vez que actualiza el documento de diseño, todas las vistas se ***vuelven a*** *ejecutar (* ***al menos para Cloudant*** *) .*

En este punto, asumo que sabes qué es un documento de diseño y cómo funciona. Nuestro documento de diseño se verá así:

```
{
    "_id":"_design/people",
    "language":"javascript"
}
```

Luego, dentro de este documento, tendrá una propiedad de vistas. Esta propiedad contiene un objeto que contiene las vistas. Cada vista tiene su propio objeto que contiene una función de **mapa** y, opcionalmente, una función de **reducción** . Aquí es cómo se ve si tenemos una vista que recupera a todas las personas de la base de datos:

```
{
    "_id":"_design/people",
    "language":"javascript",
    "views":{
        "all":{
            "map":"function(doc){if(doc.type ===\"people\")emit(doc._id);}"
        }
    }
}
```

Tal visión devolvería algo como esto:

```
{
    "total_rows": 2,
    "offset": 0,
    "rows": [
        { "id": "people_23929319009123", "key": "people_23929319009123", "value": null },
        { "id": "people_11482871000723", "key": "people_11482871000723", "value": null }
    ]
}
```

Lo que hemos hecho hasta ahora es la visión que nos da toda la gente. El equivalente en SQL sería: `SELECT * FROM table WHERE type="people"` . Voy a explicar en detalle cómo funciona la función de mapa.

Función de mapa: todos

```
function(doc) {
    if (doc.type === "people") emit(doc._id);
}
```

Primero, necesita saber que la función de mapa se ejecutará para cada documento. Ahora para la función de mapa, necesita saber que toma un parámetro: **doc** . Dentro de su función de mapa, su lógica determinará si el documento necesita ser mapeado o no. En caso afirmativo, utilizará la función **emit ()** para indexarla. La función de emisión toma 2 parámetros.

1. La clave para indexar.
2. El valor a emitir.

Al final, creará una matriz con 3 columnas: **id** , **clave** , **valor** .

*Nota: NUNCA PERO NUNCA emita el documento como el valor. Esto es totalmente inútil ya que el uso del parámetro **include_docs** buscará los documentos asociados a la identificación.*

Llaves complejas

Ahora digamos que queremos atraer a las personas de acuerdo con diferentes parámetros. Digamos que quiero consultar a los usuarios sobre su nombre, su género y sus hijos.

En este caso, tendríamos una vista como esta:

```
function(doc) {
    if (doc.type === "people") {
        emit([doc.name,doc.gender,doc.childrenCount]);
    }
}
```

Para el ejemplo, no validé que los objetos tuvieran el parámetro requerido ya que no me causaré ningún problema. Por lo tanto, puede variar de su contexto. Es posible que desee comprobar si tienen el parámetro birthDate, por ejemplo.

Así que ahora, como puedes ver, todavía tenemos **una** clave, pero es compleja. El truco aquí es que nuestra clave es una matriz, por lo que podemos tener varias claves.

Ahora, puedes preguntarte, pero hey, ¿cómo uso esto? ¡Es raro! Mantén la calma, te mostraré cómo!

Cuando consulta varias claves, es una buena idea saber cómo funciona la comparación en CouchDB. Para más información, mira esto . Lo más importante que debes saber es que, si estás usando rangos y quieres consultar todos los elementos en una clave, necesitas usar

`starkey=[null]&endkey=[\ufff0]` . Dado que nulo es el valor más bajo y \ ufff0 es el carácter más alto, obtendrá todo entre esto.

Entonces, si quiero conseguir a todas las mujeres llamadas Julia, haría lo siguiente:

```
http://localhost:5984/db/_design/people/_view/byNameGenderChildren?starkey=["Julia","Female"]&endkey=["J
```

Básicamente, tomamos todas las filas con la tecla [Julia, Female] y para la tercera parte, tomamos cualquier cosa entre el valor más bajo (nulo) y el más alto (\ ufff0) que significa todo.

A continuación, si quiero ir a buscar a todos los varones con 3 hijos?

```
http://localhost:5984/db/_design/people/_view/byNameGenderchildren?startkey[null,"Male",3]&endkey=[\ufff
```

fácil como esto:

```
http://localhost:5984/db/_design/people/_view/byNameGenderchildren?startkey[null,"Male",3]&endkey=[\ufff
```

Lea Puntos de vista en línea: https://riptutorial.com/es/couchdb/topic/7007/puntos-de-vista

Creditos

S. No	Capítulos	Contributors
1	Empezando con couchdb	Alexis Côté, Bernhard Gschwantner, Community, Flimzy, Giancarlo Corzo, Luke Taylor, pwagner
2	Documentos de diseño	pwagner
3	Ektorp java client	JustDanyul
4	Puntos de vista	Alexis Côté