Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Documentación del proyecto PetClinic

Profiling

Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2019 – 2020

Fecha	Versión
03/06/2020	v2r00

Grupo de prácticas	**G2-15 B.F.C.N**
Autores	**Rol**
Franco Sánchez, Pablo	Desarrollador
Gutiérrez Prieto, Gabriel	Desarrollador
Lopez, Thibaut	Desarrollador
Rojas Gutiérrez, Rodrigo	Desarrollador
Romero Cáceres, Antonio	Desarrollador
Vidal Pérez, Antonio	Desarrollador

Control de Versiones

Fecha	Versión	Descripción
28/05/2020	v1r00	Elaboración del documento.
03/06/2020	v2r00	Documento terminado.

Índice de contenido

Índice de figuras

1. Profiling Disease

Tras las pruebas de rendimiento realizadas, hemos visto que el peor rendimiento estaba en el listar de enfermedades, entonces decidimos optimizar el rendimiento de esta entidad.

Para empezar, se puso en su list **@Transactional(readOnly=true)**, haciendo que estos datos sólo sean de lecturas y mejorase algo su rendimiento.

La segunda cuestión era que nada más listar ya mostrábamos sus pets, decidimos entonces eliminar de la vista esos pets, y que solo se viesen en su mostrar (se comprobó que este cambio no afectase a ningún otro tests).

Por último, se decidió cachear el listar enfermedades, al cachearlas vimos que las enfermedades que cacheaba no mostraban sus atributos adecuadamente, ante la duda de que esto afectase a la caché, se creó el **toString** de la entidad solucionando este problema, para que efectivamente cuando se guardaba en caché se viese cada uno de los elementos que guardaba.

Por último, se probaron todos los tests para saber si esto afectase a algún tests en concreto. Se vio que esto afectaba a los tests de integración de enfermedad, ya que intentaban coger los datos de caché y estos datos estaban vacíos, la solución encontrada fue añadir una variable de entorno -> **spring.cache.type = NONE** (que hace que no se lean los datos de caché para el tests en el que se ponga).

2. Resultados obtenidos del profiling Disease

A continuación, mostramos el rendimiento que teníamos sin caché con **Glowroot:**

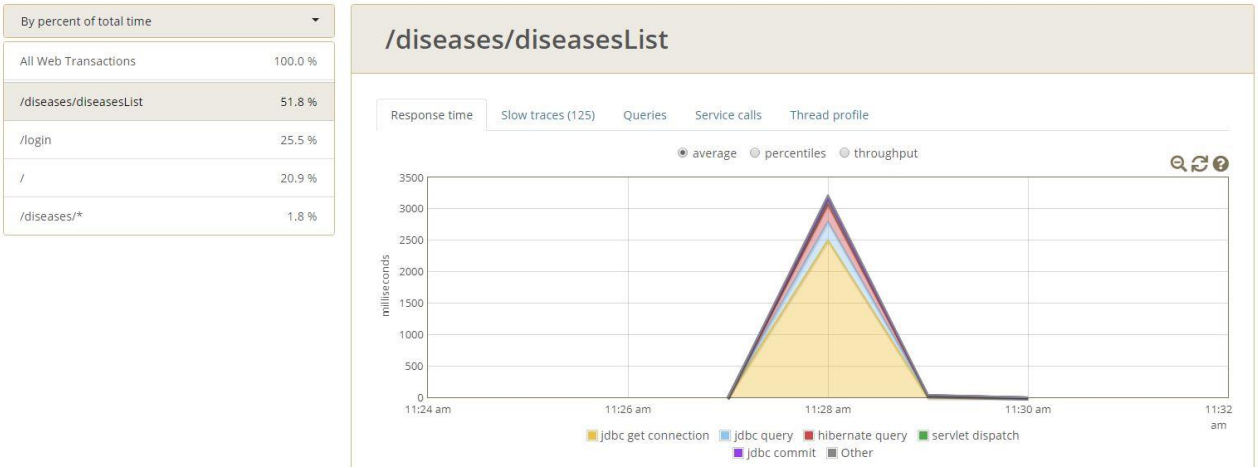

Figura 1. DiseaseList sin caché

Aqui mostramos a 500 usuarios realizando peticiones al sistema, el tiempo que tardan en entrar y su tiempo de respuesta.

Figura 2. Queries DiseaseList sin caché

Aquí vemos las peticiones que realiza en el listar y mostrar, y vemos una medida bastante alta.

Ahora tendremos en cuenta nuestro rendimiento con caché, en el que se ve que para el mismo número de usuarios muestra unos mejores datos:

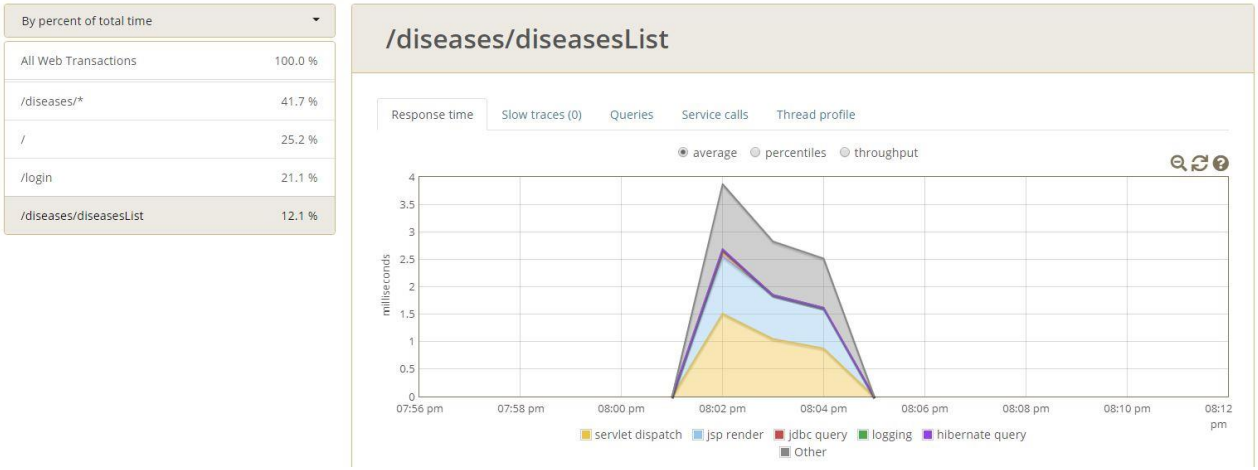

Figura 3. DiseaseList con caché

Como se puede ver, se ha reducido considerablemente el tiempo, al principio es más alto porque los datos todavía no se encuentran en caché, pero después de la primera petición los tiempos se reducen ya que estos datos están almacenados en caché y se mantendrá así hasta cumplir el tiempo que se puso que durase o se modifique algún dato de enfermedad.

Figura 4. Queries DiseaseList con caché

El tiempo medio por query también se ha reducido, con lo que podemos comprobar que la optimización ha sido un éxito.

3. Profiling Booking

Después de hacer las pruebas de rendimiento pudimos ver que el list de booking tenía un mal rendimiento.

Para intentar mejorar el rendimiento pusimos etiquetas **@Transactional** y **@ReadOnlyProperty.** Además, optamos por añadirle una cache a la lista de bookings para mejorar el rendimiento de este.

4. Resultados del Profiling

Pasaremos a analizar la aplicación con **Glowroot**.

Antes de añadir cache:

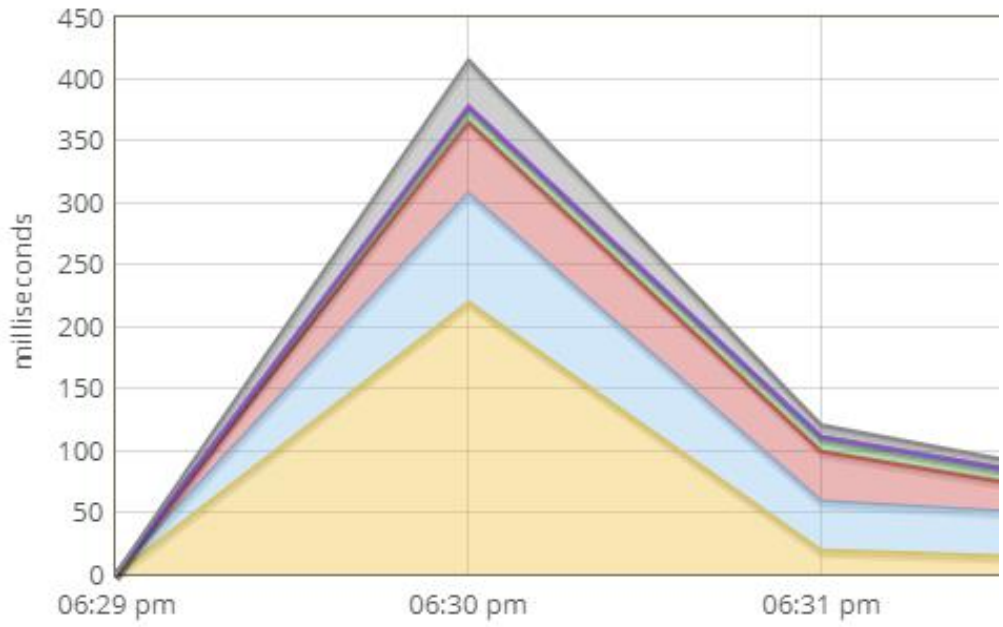

Figura 5. Booking sin caché

Querys que ha realizado:

	Total time (ms)	Total count	Avg time (ms)	Avg rows
`select pet0_.id as id1_7_0_, pet0_.name as name2_7_0_, pet0_.birth_date as birth_da3_7_...`	51,7	78	0.66	1,0
`select vet0_.id as id1_15_0_, vet0_.first_name as first_na2_15_0_, vet0_.last_name as l...`	41,7	78	0.53	1,2
`select owner0_.id as id1_5_0_, owner0_.first_name as first_na2_5_0_, owner0_.last_name ...`	28,0	65	0.43	1,0
`select room0_.id as id1_10_0_, room0_.name as name2_10_0_, room0_.floor as floor3_10_0_...`	20,1	52	0.39	1,0
`select booking0_.id as id1_1_, booking0_.date as date2_1_, booking0_.owner_id as owner_...`	4,5	13	0.34	6,0

Figura 6. Querys sin caché

Observamos como después de hacer varias querys seguidas, todas estas piden a la base de datos la información de booking (13 veces) y esto hace que la aplicación tarde mucho y administre recursos a estas operaciones.

Después de añadir la cache:

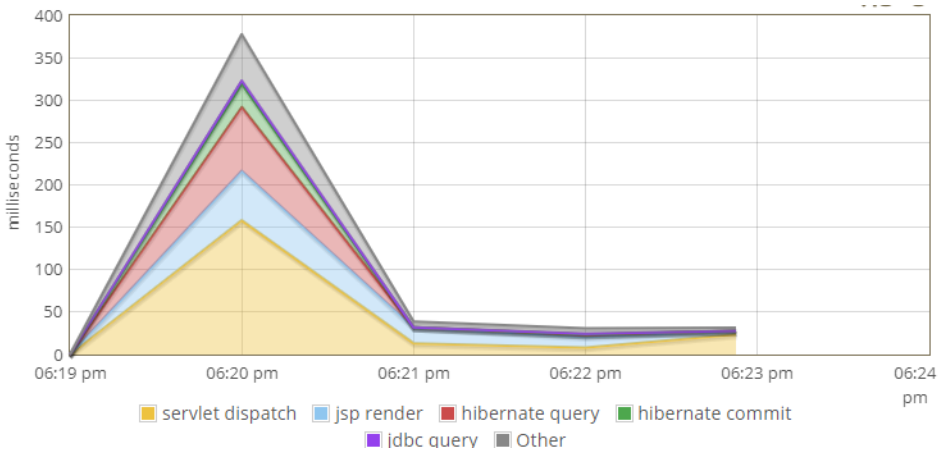

Figura 7. Booking con caché

Querys que ha realizado:

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select booking0_.id as id1_1_, booking0_.date as date2_1_, booking0_.owner_id as owner_id...	4,9	1	4,9	6,0
select pet0_.id as id1_7_0_, pet0_.name as name2_7_0_, pet0_.birth_date as birth_da3_7_0_...	3,6	6	0.60	1,0
select vet0_.id as id1_15_0_, vet0_.first_name as first_na2_15_0_, vet0_.last_name as las...	2,8	6	0.46	1,2
select owner0_.id as id1_5_0_, owner0_.first_name as first_na2_5_0_, owner0_.last_name as...	2,2	5	0.43	1,0
select room0_.id as id1_10_0_, room0_.name as name2_10_0_, room0_.floor as floor3_10_0_ f...	1,9	4	0.47	1,0

Figura 8. Querys con caché

Observamos como el tiempo de respuesta es menor e incluso haciéndole varias peticiones seguidas solo llama a la base de datos una única vez (hasta que se tenga que renovar la cache por la modificación de datos o la caducidad de estos)

5. Profiling Owners

Tras realizar las pruebas de rendimiento de Chips, se puede ver que la aplicación tarda mucho tiempo en listar los Owners, lo que reducia mucho el rendimiento de las pruebas. La forma que se ha elegido para arreglar esto es añadir una caché, que se usará durante estas consultas.

6. Resultados del Profiling

Antes de añadir la caché, esto es lo que se puede observar al usar Glowroot sobre la aplicación:

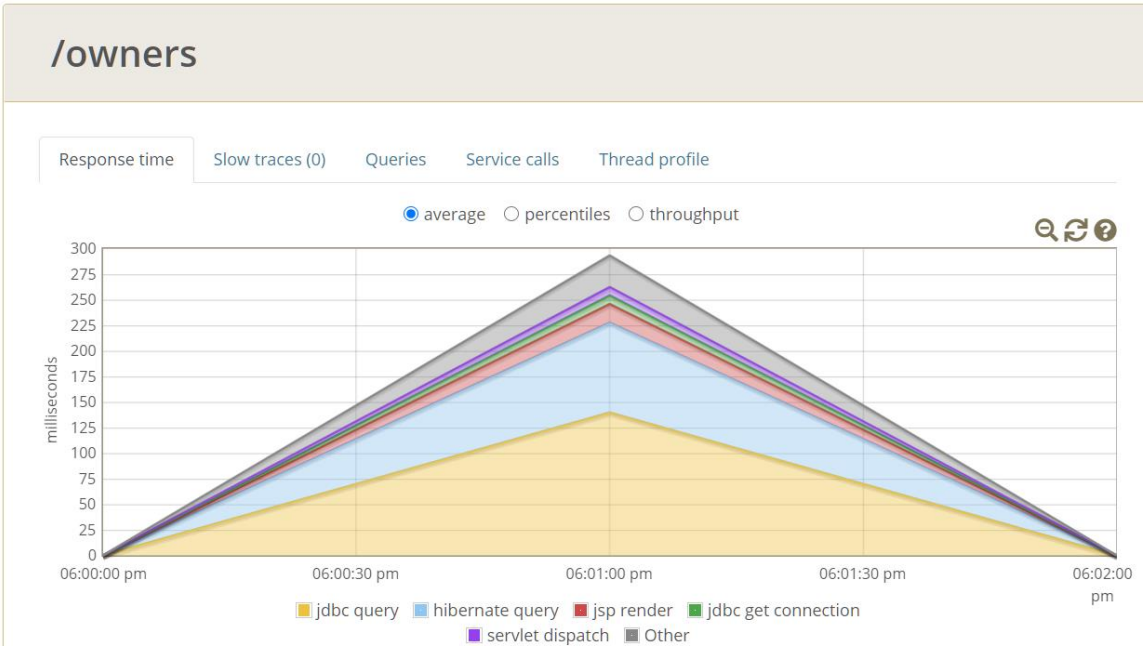

Figura 9. Owners sin caché

Figura 10. Querys sin caché

A continuación, después de añadir la caché:

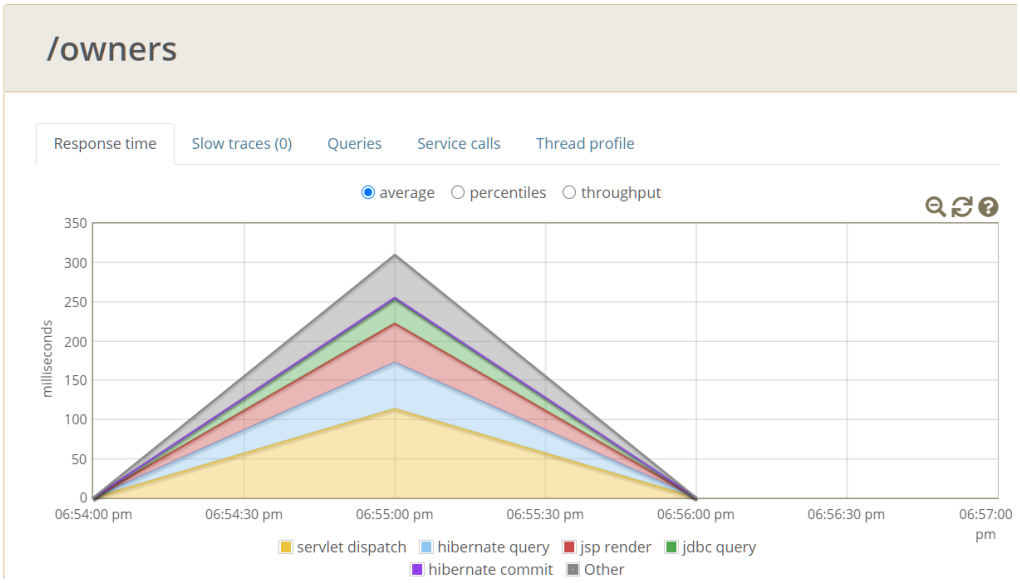

Figura 11. Owners con caché

/owners

Response time | Slow traces (0) | Queries | Service calls | Thread profile

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select chip0_.id as id1_2_4_, chip0_.geolocatable as geolocat2_2_4_, chip0_.model as mo...	46,3	13	3,6	0,2
select pettype0_.id as id1_12_0_, pettype0_.name as name2_12_0_ from types pettype0_ wh...	35,1	6	5,9	1,0
select visits0_.pet_id as pet_id4_17_0_, visits0_.id as id1_17_0_, visits0_.id as id1_1...	34,1	10	3,4	0,4
select user0_.username as username1_13_0_, user0_.enabled as enabled2_13_0_, user0_.pas...	6,8	2	3,4	1,0
select distinct owner0_.id as id1_5_0_, pets1_.id as id1_7_1_, owner0_.first_name as fi...	4,2	1	4,2	13,0

Figura 12. Querys con caché

Como se puede ver, mediante el uso de caché se logra la mitad de tiempo y la mitad de las consultas en casi cada query.