

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Documentación de las pruebas de profiling

Disease




Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2019 – 2020


Fecha	Versión
28/05/2020	v1r00

Grupo de prácticas	G2-15
Autores	Rol
Rojas Gutiérrez, Rodrigo	Desarrollador
Romero Cáceres, Antonio	Desarrollador

	Proceso de Software y Gestión 1 Documentación de la práctica <nn>
	Control de Versiones


Control de Versiones

Fecha	Versión	Descripción
28/05/2020	v1r00	Elaboración del documento.

	<p>Diseño y Pruebas II</p> <p>Documentación de las pruebas de profiling</p>
---	---


Índice de contenido

1. Cambios realizados	4
2. Resultados obtenidos	5

	<p>Diseño y Pruebas II Documentación de las pruebas de profiling</p>


Índice de figuras

<i>Figura 1. DiseaseList sin caché.....</i>	<i>5</i>
<i>Figura 2. Queries DiseaseList sin caché.....</i>	<i>5</i>
<i>Figura 3. DiseaseList con caché.....</i>	<i>6</i>
<i>Figura 4. Queries DiseaseList con caché.....</i>	<i>6</i>

	<p>Diseño y Pruebas II Documentación de las pruebas de profiling</p>

Índice de tablas

[No se encuentran elementos de tabla de ilustraciones.](#)

	<p>Diseño y Pruebas II</p> <p>Documentación de las pruebas de profiling</p>
---	---

1. Cambios realizados

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 mostrabamos 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), posteriormente se le puso a esta propiedad un **FetchType=LAZY**, para leer estos datos solo cuando se mostrasen.

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

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

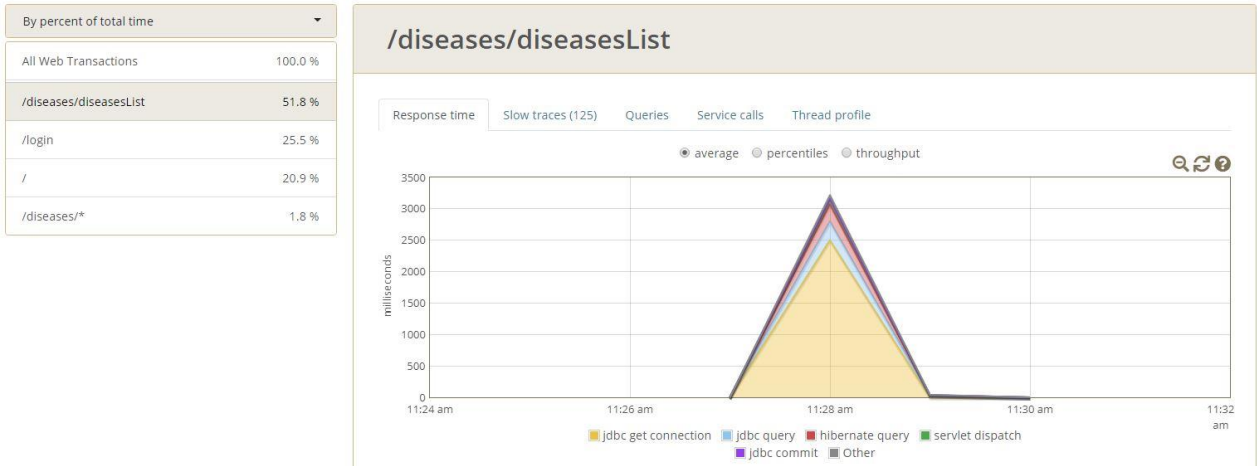


Figura 1. DiseaseList sin caché

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

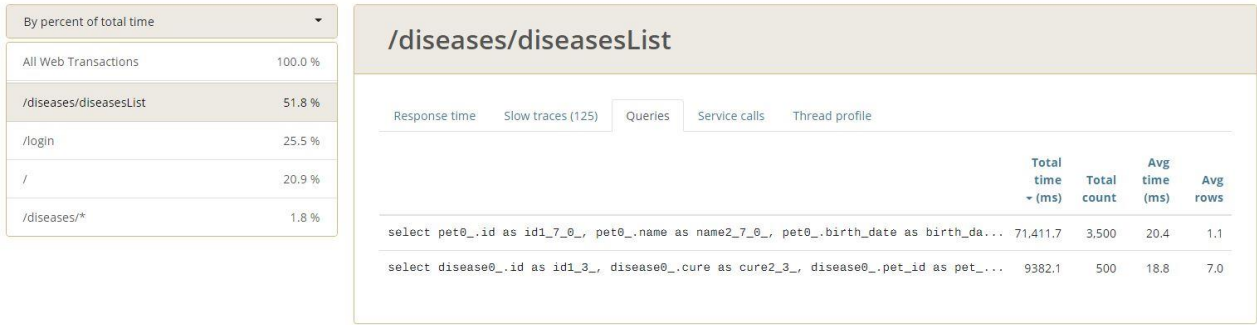


Figura 2. Queries DiseaseList sin caché

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:

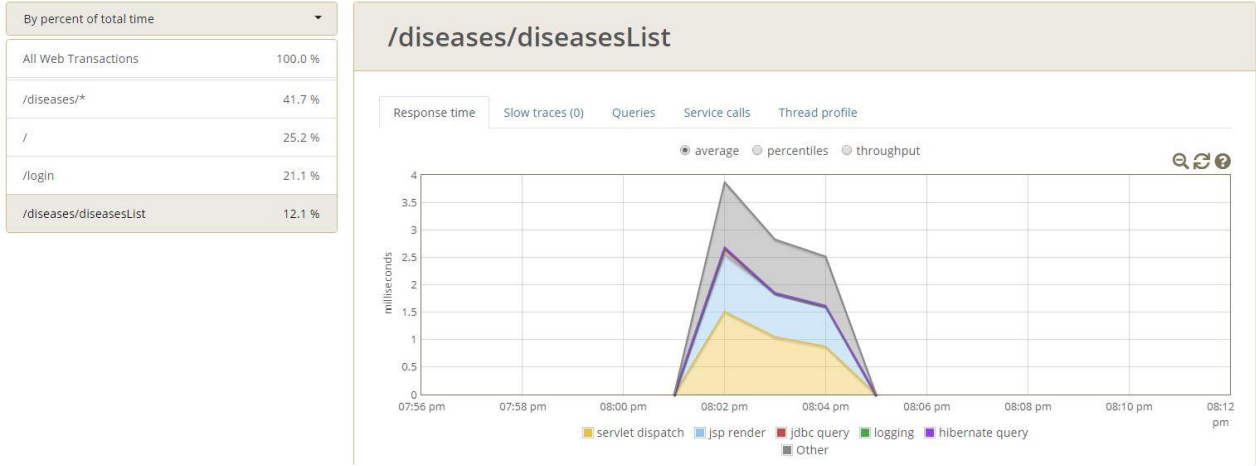


Figura 3. DiseaseList con caché

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.