

redis

Ein Key/Value In-Memory Datenspeicher

Wintersemester 2017
adelheid.knodel@hs-karlsruhe.de

- Historie
- Was ist Redis?
- Getting Started
- Redis Datentypen
- Datenmodellierung
- Spring Data Redis
- Sessionverwaltung mit Redis
- Links



redis

- Redis – „REmote DIctionary Server“
- Projekt gestartet 2009 von Salvatore Sanfilippo
aktuelle Version 4.0.2 (26. Oktober 2017)
- Entwickelt für Realtime Web Log Analyzer
Gründe:
 - bessere Skalierbarkeit
 - Geringe Kosten
 - Wenig Hardware
 - Möglichkeit für schnelle Operationen
- Implementierung unter Unix in C
- Open Source
- Unterstützt von VMWare, Pivotal und Redis Labs



redis

Was ist Redis?



- NoSQL-Datenbank
- Key /Value Datenspeicher
- Datensätze im Arbeitsspeicher gespeichert
- Datentypen: String, Hash, List, Set, Sorted Set
- Persistenz möglich
- Publish/ Subscribe Messaging System



redis

Key

Key1

page:index.html

user:123:session

login_count

user:100:last_login_time

Value

→ value1

→ <html><head>[...]

→ xDrSdEwd4dSlZkEkj+

→ "7464"

→ "102736485756"



redis

- Interaktives Tutorial

- <http://try.redis.io/>

- Redis Downloads

- Linux

- <http://download.redis.io/>

- Windows

- <https://github.com/MicrosoftArchive/redis>

- <https://github.com/MicrosoftArchive/redis/releases>

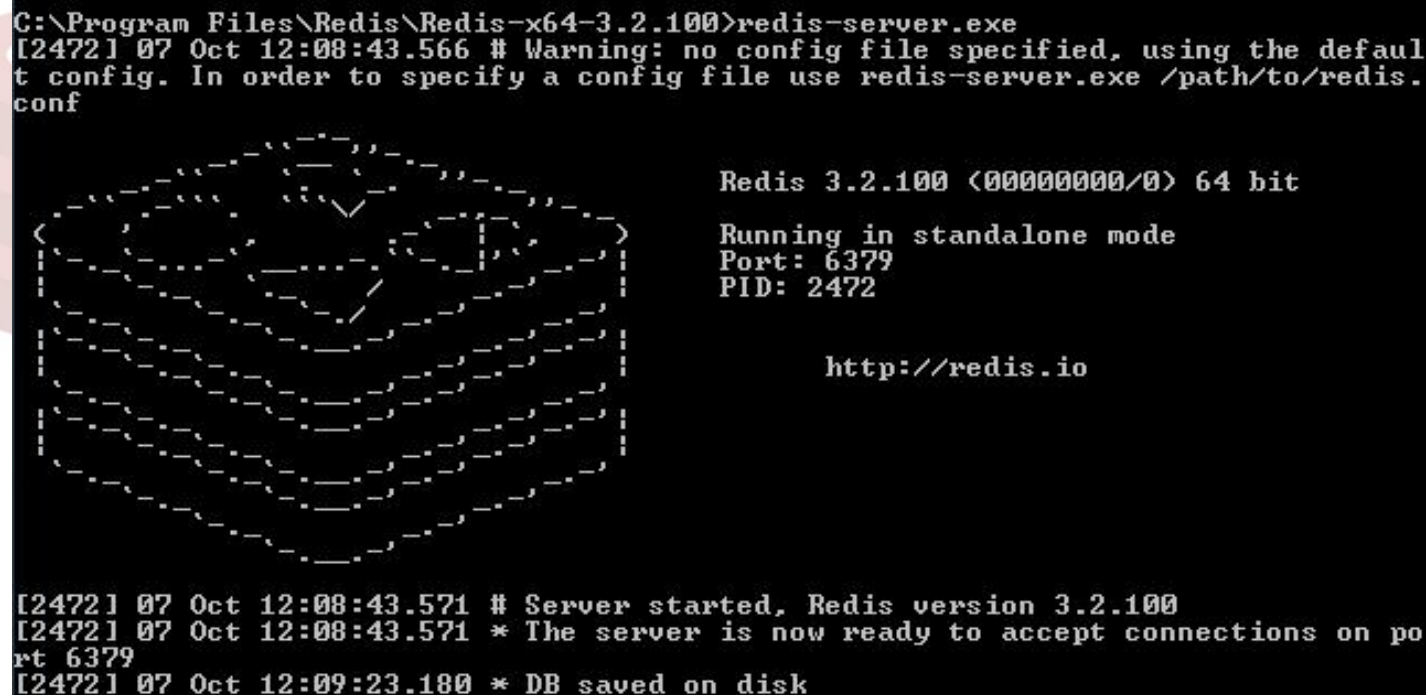
- <http://redis.io/topics/quickstart>



starten des Redis-Servers

Windows: redis-server.exe (evtl. als Administrator)

Unix: ./redis-server



```
C:\Program Files\Redis\Redis-x64-3.2.100>redis-server.exe
[2472] 07 Oct 12:08:43.566 # Warning: no config file specified, using the default
t config. In order to specify a config file use redis-server.exe /path/to/redis.
conf

Redis 3.2.100 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 2472

http://redis.io

[2472] 07 Oct 12:08:43.571 # Server started, Redis version 3.2.100
[2472] 07 Oct 12:08:43.571 * The server is now ready to accept connections on po
rt 6379
[2472] 07 Oct 12:09:23.180 * DB saved on disk
```

starten des Redis-Clients

Windows: redis-cli.exe

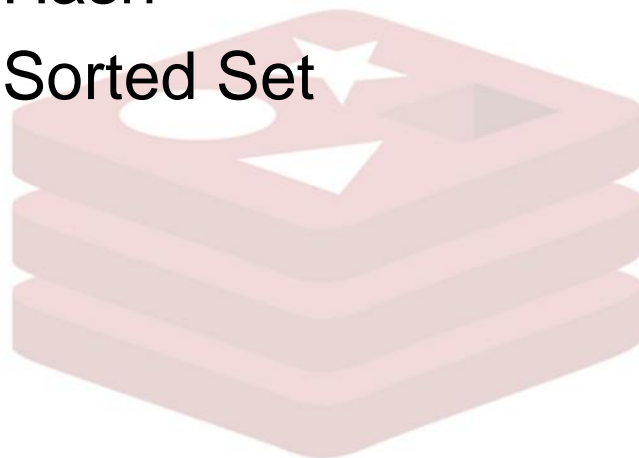
Unix: ./redis-cli

```
C:\Program Files\Redis\Redis-x64-2.8.2400>redis-cli.exe
127.0.0.1:6379> set first-key hallo
OK
127.0.0.1:6379> get first-key
"hallo"
127.0.0.1:6379> _
```

Monitoring der Anfragen in neuem Fenster

```
127.0.0.1:6379> monitor
OK
1508929453.639751 [0 127.0.0.1:51803] "get" "firts_key"
1508929463.030968 [0 127.0.0.1:51803] "get" "first_key"
_
```


- String
- List
- Set
- Hash
- Sorted Set



redis

Keys

page:index.html

login_count

users_logged_in_today_1

users_logged_in_today_2

latest_post_ids

user:123:session

users_and_scores

Values

→ <html><head>[...]

→ 7464

→ { 1, 4, 2, 5, 3 }

→ [1, 4, 2, 5, 4, 3, 1]

→ [201,209,204]

→ time => 10927353

username => joe

→ "1912" "Alan Turing"

"1914" "Hedy Lamarr"

"1916" "Claude Shannon"

"1940" "Alan Kay"

← **String**

← **String**

← **Set**

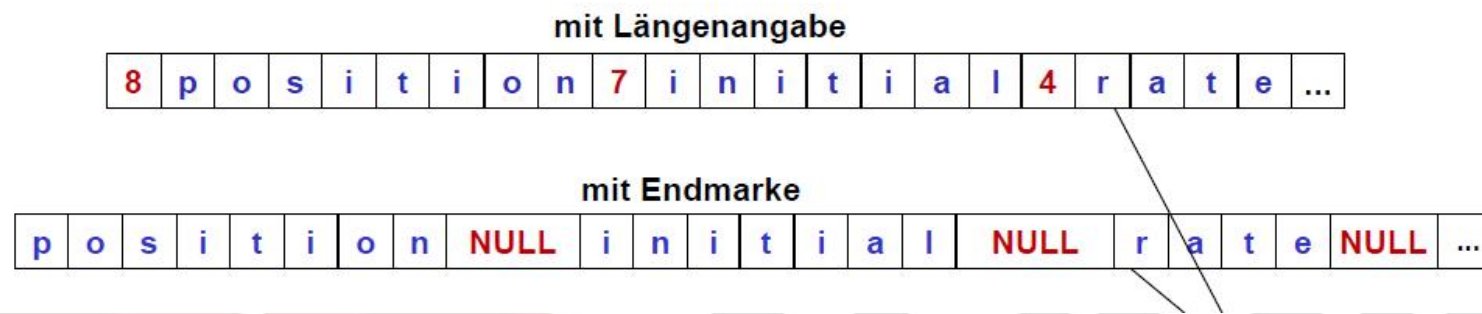
← **List**

← **List**

← **Hash**

← **Sorted Set**

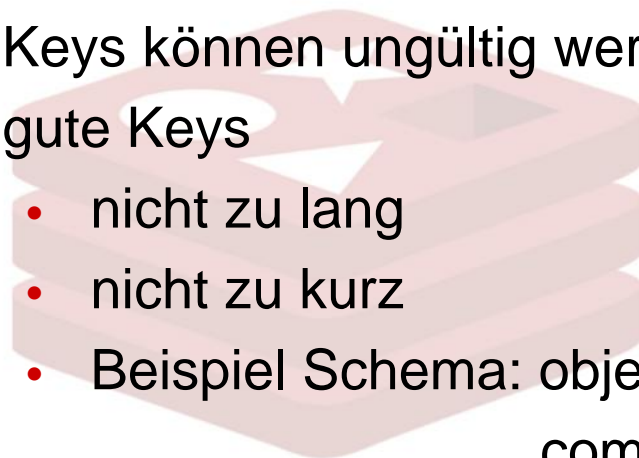
- Basisdatentyp
- String eine Sequenz von Bytes



aus Folien Systemnahes Programmieren Prof. Fuchß

- Besitzt eine bekannte Länge, nicht durch ein bestimmtes Symbol beendet
- Binary safe – alle Arten von Daten, z.B. jpeg, png, exe, pdf Datei
- Max. Größe 512 MB

- Keys sind Strings
- auch leerer String ist gültiger Schlüssel
- sind binary safe
- Keys können ungültig werden
- gute Keys
 - nicht zu lang
 - nicht zu kurz
 - Beispiel Schema: object-type:id:field
comment:1234:reply.to
user_Timeline:userId:54



redis

Befehle:

- del *key*
- exists *key*
- expire *key seconds*

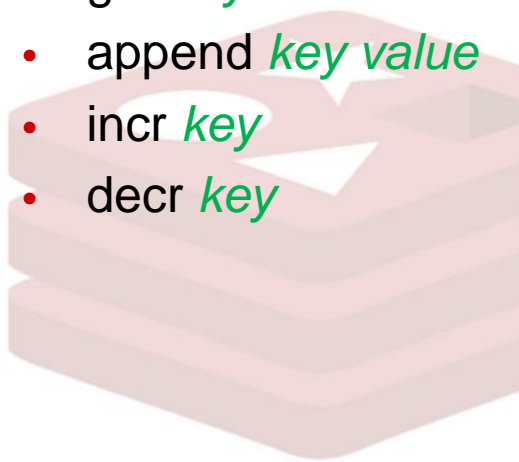


```
127.0.0.1:6379> exists my_first_key
(integer) 0
127.0.0.1:6379> set my_first_key 10
OK
127.0.0.1:6379> exists my_first_key
(integer) 1
127.0.0.1:6379> del my_first_key
(integer) 1
127.0.0.1:6379> exists my_first_key
(integer) 0
127.0.0.1:6379>
```

Vollständige Liste der Befehle unter <http://redis.io/commands#generic>

Befehle Beispiel:

- set *key value* [ex *seconds*]
- get *key*
- append *key value*
- incr *key*
- decr *key*



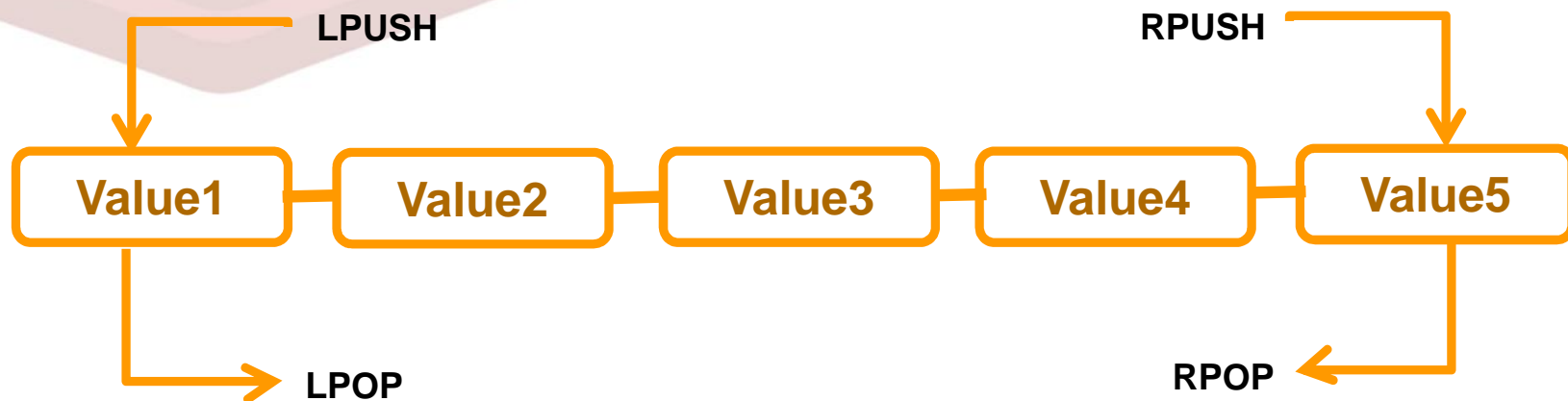
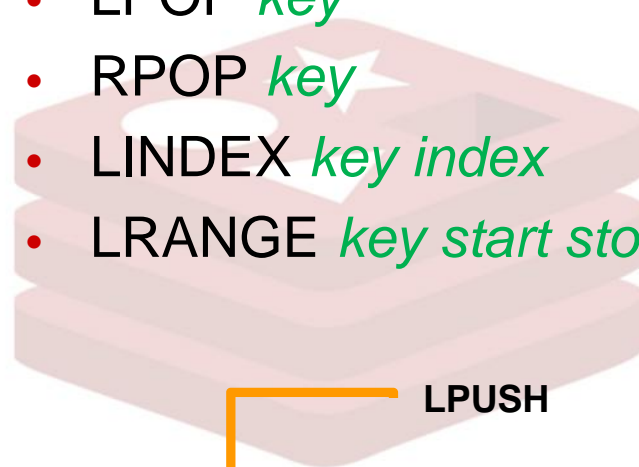
```
127.0.0.1:6379> set first_key hallo
OK
127.0.0.1:6379> get first_key
"hallo"
127.0.0.1:6379> append first_key welt
(integer) 9
127.0.0.1:6379> get first_key
"hallowelt"
127.0.0.1:6379> set first_value 10
OK
127.0.0.1:6379> incr first_value
(integer) 11
127.0.0.1:6379> get first_value
"11"
127.0.0.1:6379> decr first_value
(integer) 10
127.0.0.1:6379> _
```

Vollständige Liste der Befehle unter <http://redis.io/commands#string>

- Liste von Strings
- Implementiert als Linked List
- Schnelles Einfügen/Lesen der Elemente
 - am Anfang
 - am Ende
- sortiert durch die Reihenfolge des Einfügens
- Use Case Beispiel:
Modellieren einer Timeline in einem sozialen Netzwerk

■ Befehle:

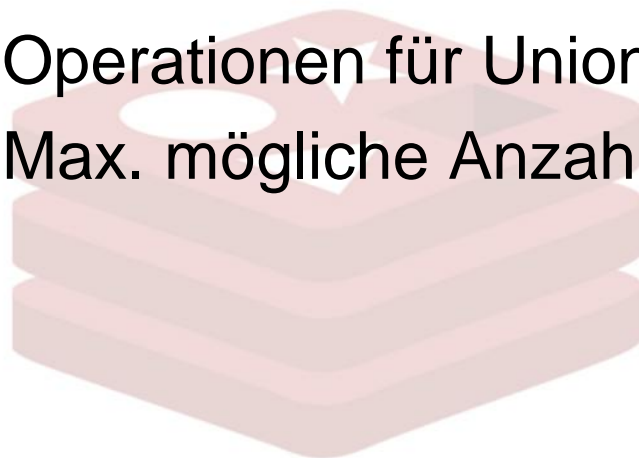
- LPUSH *key value [value..]* einfügen am Anfang der Liste
- RPUSH *key value [value..]* einfügen am Ende der Liste
- LPOP *key* lesen und löschen am Anfang der Liste
- RPOP *key* lesen und löschen am Ende der Liste
- LINDEX *key index* lesen des Elements mit dem Index
- LRANGE *key start stop* lesen eines Bereichs von Elementen




```
(integer) 0
127.0.0.1:6379> lpush listkey user1 user2 user3 user4
(integer) 4
127.0.0.1:6379> llen listkey
(integer) 4
127.0.0.1:6379> lrange listkey 0 5
1) "user4"
2) "user3"
3) "user2"
4) "user1"
127.0.0.1:6379> lrange listkey 0 10
1) "user4"
2) "user3"
3) "user2"
4) "user1"
127.0.0.1:6379> rpop listkey
"user1"
127.0.0.1:6379> lrange listkey 0 5
1) "user4"
2) "user3"
3) "user2"
127.0.0.1:6379> _
```

Vollständige Liste der Befehle unter <http://redis.io/commands#list>

- Eine unsortierte Sammlung von Strings
- jeder Wert ist nur einmal im Set vorhanden
- Reihenfolge der Werte kann bei jeder Abfrage unterschiedlich sein
- Operationen für Unions, Intersections, Differences möglich
- Max. mögliche Anzahl Einträge $2^{32} - 1$



redis

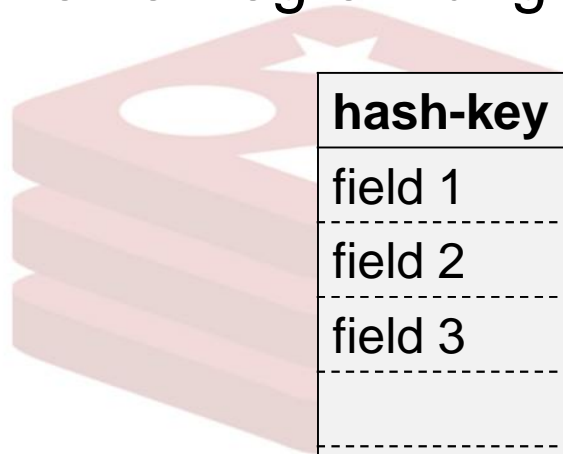
Befehle

- sadd *key member* [*member* ...]
Element(e) hinzufügen
- scard *key*
Anzahl Elemente anzeigen
- smembers *key*
alle Elemente lesen
- sismember *key value*
ist value Element von key
- sinter *key* [*key* ...]
Schnittmenge (intersection)
- sdiff *key* [*key* ...]
Unterschied (difference)
- sunion *key* [*key* ...]
Vereinigung (union)

Vollständige Liste der Befehle unter <http://redis.io/commands#set>

```
127.0.0.1:6379> sadd news:1000:tags 1 2 5 77
(integer) 4
127.0.0.1:6379> sadd tag:1:news 1000
(integer) 1
127.0.0.1:6379> sadd tag:2:news 1000
(integer) 1
127.0.0.1:6379> sadd tag:5:news 1000
(integer) 1
127.0.0.1:6379> sadd tag:77:news 1000
(integer) 1
127.0.0.1:6379> smembers news:1000:tags
1) "1"
2) "2"
3) "5"
4) "77"
127.0.0.1:6379> sadd key1 a b c d
(integer) 4
127.0.0.1:6379> sadd key2 c
(integer) 1
127.0.0.1:6379> sadd key3 a c e
(integer) 3
127.0.0.1:6379> sinter key1 key2 key3
1) "c"
127.0.0.1:6379>
```

- Sammlung von field/ value Paaren
- Besteht aus einzelnen Feldern und zugehörigen Werten
- Bester Datentyp, um Objekte zu speichern
- Keine Begrenzung der Anzahl der Felder



hash-key	
field 1	value 1
field 2	value 2
field 3	value 3

dis

Befehle:

- hmset *key field value* [*field value*...]

mehrere Felder mit Werten speichern

- hget *key field*

ein Feld lesen

- hmget *key field* [*field* ...]

mehrere Felder lesen

- hexists *key field*

prüfen, ob Feld existiert

- hkeys *key*

alle Feldnamen lesen

- hdel *key field* [*field* ...]

ein oder mehrere Felder löschen

- hincrby *key field increment*

integer Wert eines Feldes um
increment erhöhen

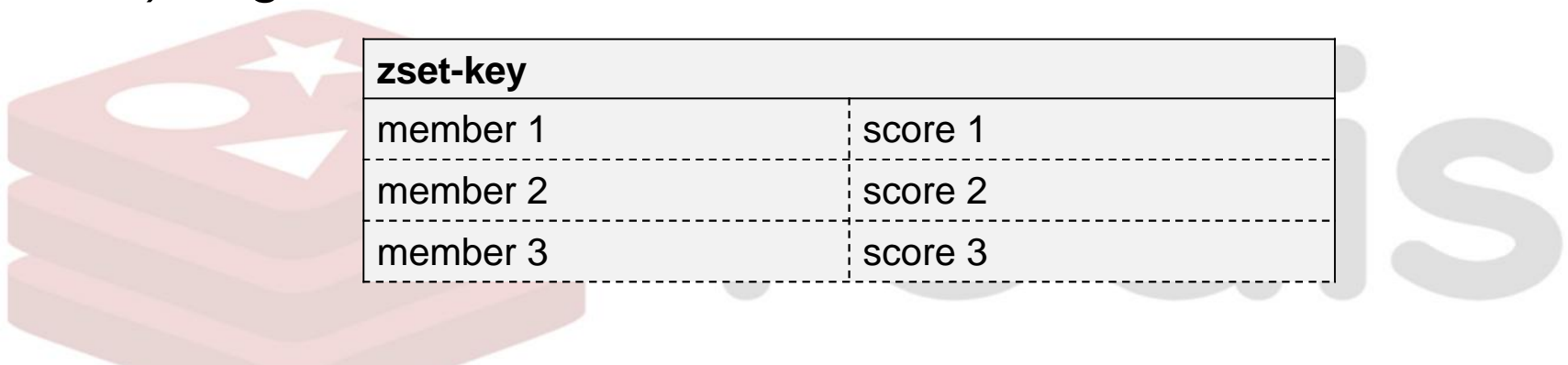
Datentypen Hash Beispiel



```
127.0.0.1:6379> hmset user:1000 username antirez birthyear 1977 verified 1
OK
127.0.0.1:6379> hget user:1000 username
"antirez"
127.0.0.1:6379> hget user:1000 birthyear
"1977"
127.0.0.1:6379> hget user:1000 verified
"1"
127.0.0.1:6379> hgetall user:1000
1) "username"
2) "antirez"
3) "birthyear"
4) "1977"
5) "verified"
6) "1"
127.0.0.1:6379>
```

Vollständige Liste der Befehle unter <http://redis.io/commands#hash>

- Sammlung von Strings
- Jedes Element (member) kann nur einmal vorkommen
- Jedes Element eines Sorted Sets hat einen Wert (*element score*) zugeordnet.



zset-key	
member 1	score 1
member 2	score 2
member 3	score 3

- Ordnung des Sets anhand der Scores nach folgenden Regeln:
 - A und B Elemente mit verschiedenen Scores,
→ $A > B$, falls $A.score > B.score$.
 - A und B Elemente mit dem selben Score,
→ $A > B$, falls A String lexikographisch größer als B String

Befehle

- `zadd` *key score member* [*score member ...*]
ein oder mehr Elemente hinzufügen
- `zcard` *key*
Anzahl Elemente
- `zrange` *key begin end* lesen eines Bereichs von Elementen
- `zscore` *key member* lesen des Scores eines Elements
- `zrangebylex` *key min max* [*LIMIT offset count*]
lesen der Elemente in lex.
Ordnung (nur bei selben Scores)

Vollständige Liste der Befehle unter http://redis.io/commands#sorted_set

- Ordnung des Sets anhand der Scores nach folgenden Regeln:

- A und B Elemente mit verschiedenen Scores,
→ $A > B$, falls $A.score > B.score$.
- A und B Elemente mit dem selben Score,
→ $A > B$, falls A String lexikographisch größer als B String

Beispiel1: key: zset_key
Elemente: (Alice 100) (Bert 15)
(Conni 125) (Xaver 110)

zset_key	
Bert	15
Alice	100
Xaver	110
Conni	125

Beispiel 2: key: zset_key_2
Elemente: (Alice 10) (Bert 10)
(Conni 10) (Xaver 10)

zset_key_2	
Alice	10
Bert	10
Conni	10
Xaver	10

Datentypen Sorted Set



```
127.0.0.1:6379> zadd zset_key 100 Alice 15 Bert 125 Conni 110 Xavier
(integer) 4
127.0.0.1:6379> zrange zset_key 0 -1
1) "Bert"
2) "Alice"
3) "Xaver"
4) "Conni"
127.0.0.1:6379> zrange zset_key 0 -1 withscores
1) "Bert"
2) "15"
3) "Alice"
4) "100"
5) "Xaver"
6) "110"
7) "Conni"
8) "125"
127.0.0.1:6379>
```

```
127.0.0.1:6379> zadd zset_key_1 10 Alice 10 Bert 10 Conni 10 Xavier
(integer) 4
127.0.0.1:6379> zrange zset_key_1 0 -1 withscores
1) "Alice"
2) "10"
3) "Bert"
4) "10"
5) "Conni"
6) "10"
7) "Xaver"
8) "10"
127.0.0.1:6379>
```

- Daten müssen so organisiert werden, dass alle Anfragen über den „Primärschlüssel“ erfragt werden können
- Modellierung typischerweise ausgehend von
 - welche Anfragen werden gestellt
 - wie werden die Daten gesucht
- Daten müssen evtl. mehrmals gespeichert werden
- Keine Relationen möglich, Beziehungen müssen manuell gepflegt werden

Datentyp	Beschreibung
String	Für Informationen, die in textueller Form vorliegen. Zum Beispiel um HTML, JSON, or XML abzuspeichern.
List	Daten am Anfang oder Ende bearbeiten, z.B. Queues or Stacks.
Set	Länge einer Collection wird benötigt, prüfen, ob bestimmtes Element enthalten Relationen umsetzen, wie z.B. "wer sind Johns Freunde"
Sorted set	Sorted Sets wie Sets, zusätzlich die Ordnung wichtig
Hash	Hash ist die beste Struktur, um Objekte zu repräsentieren.

Aus <https://www.packtpub.com/books/content/building-applications-spring-data-redis>

Auto:

- Hersteller
- Modell
- Farbe
- Höchstgeschwindigkeit
- Herstellungsjahr

Datenstruktur: String

key →	value
car:1:make	Opel
car:1:model	Adam
car:1:color	red
car:1:max-speed	200
car:1:date	2015

Car

id:	1
make:	Opel
model:	Adam
color:	red
max-speed:	200
date:	2015

Datenstruktur: Hash

key →	car:<id>
car:1	
make	Opel
model	Adam
color	red
max-speed	200
date	2015

Auto:

- Hersteller
- Modell
- Farbe
- Höchstgeschwindigkeit
- Herstellungsjahr

Car

id:	1
make:	Opel
model:	Adam
color:	red
max-speed:	200
date:	2015

Speichern des Objekts:

```
> hmset car:1 make Opel model Adam color red max-speed 200 date 2015
```

Lesen des Objekts über die Id:

```
> hgetall car:1
```

Frage: finde alle Modelle eines Herstellers

- Keine Möglichkeit nach Eigenschaft eines Objekts zu suchen

Lösung: manuelles Erstellen eines Index

Datenstruktur: Set

key:

make:<hersteller>

value:

Ids der Modelle des Herstellers

make:Opel

make:Porsche

make:Ford

Ids der Modelle von Opel

Ids der Modelle von Porsche

Ids der Modelle von Ford

Sobald ein Objekt hinzugefügt wird, muss Index aktualisiert werden

```
> hmset car:1 make Opel model Adam color red max-speed 200 date 2015  
> sadd make:opel 1  
> hmset car:5 make Opel model Vectra color blue max-speed 220 date 2016  
> sadd make:opel 5
```

Finde alle Modelle von Opel

```
> smembers make:opel
```

1) 1

2) 5



Frage: finde alle Modelle von Opel mit der Farbe rot

Datenstruktur: Hash für Objekte

Set für Modelle eines Herstellers

Set für Farben der Autos



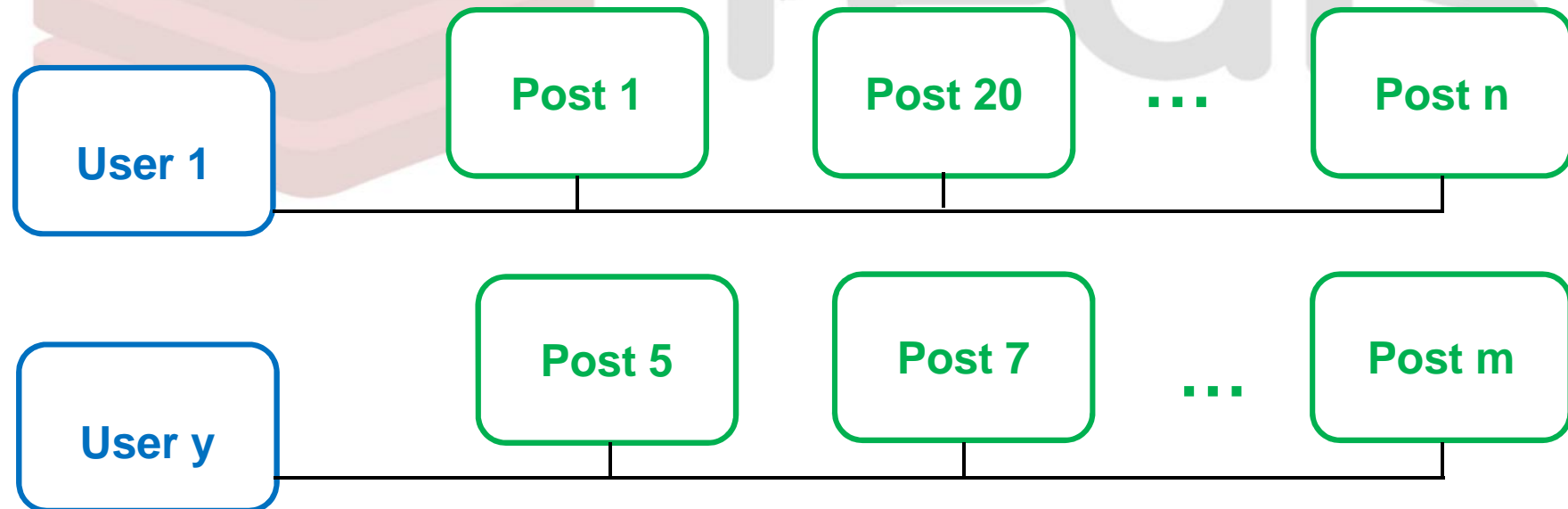
```
> hmset car:1 make Opel model Adam color red max-speed 200 date 2015
> sadd make:opel 1
> sadd color:red 1
> sinter make:opel color:red
```

Frage: finde alle Modelle von Opel und Porsche mit der Farbe rot

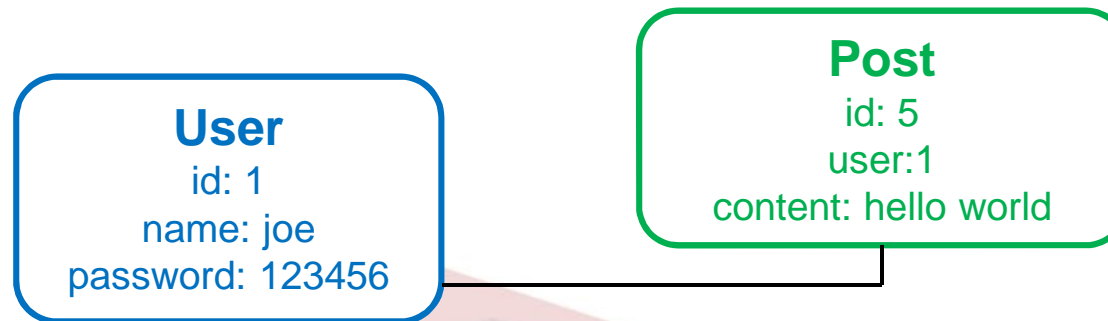
```
> sinterstore opel:rot make:opel color:red
> sinterstore porsche:rot make:porsche color:red
> sunion opel:rot porsche:rot
```

Social Network

- **User:** hat Namen
kann anderen folgen
kann gefolgt werden
- **Post:** enthält eine Nachricht



Modellierung Beispiel 2



key value

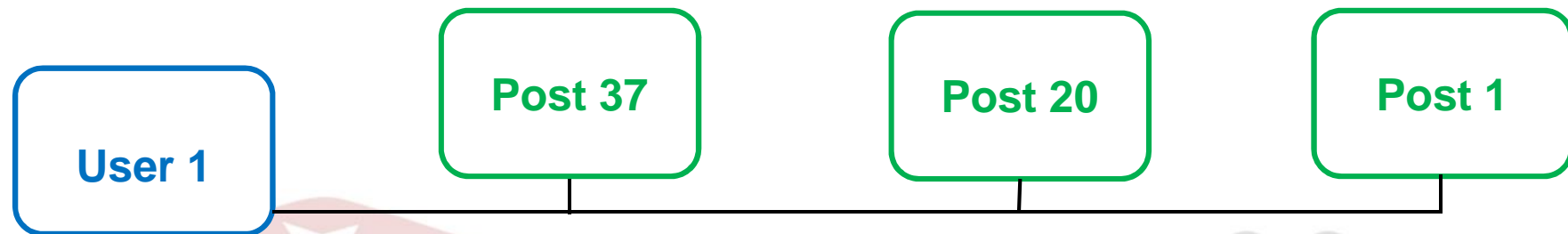
user:<id>name	user_name
username:<name>	user_id

set user:1: ~~name~~ joe
set username: joe 1

post:<id>:content post_content
post:<id>:user user_id

set post:5:content "hello world"
set post:5:user 1

Modellierung Beispiel 2



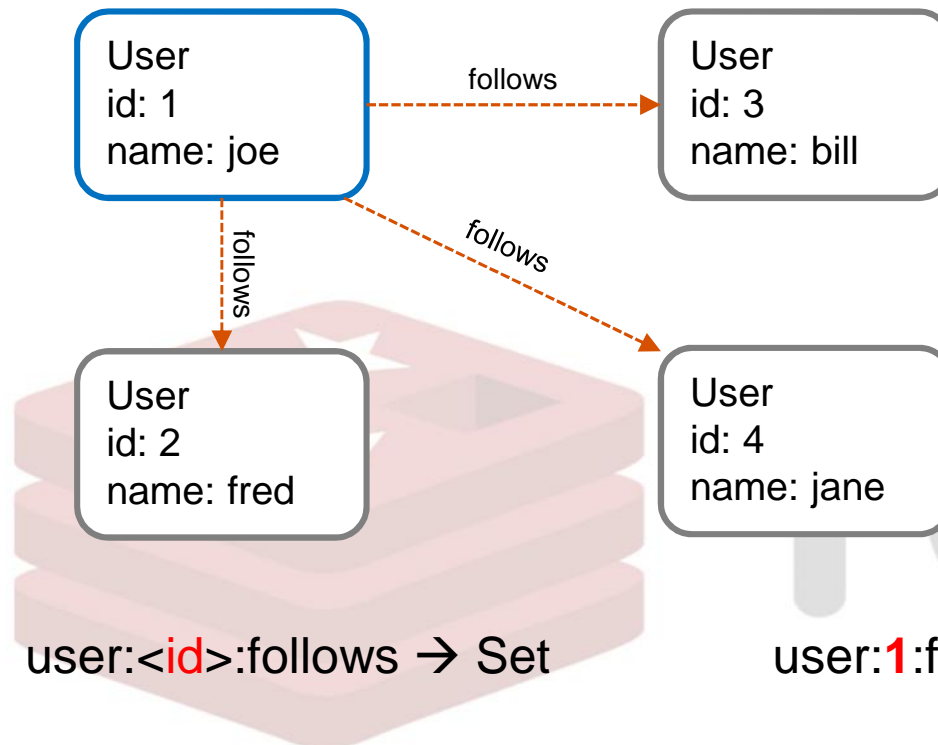
user:<id>:posts → List

user:1:posts

lpush user:1:posts **1**
lpush user:1:posts **20**
lpush user:1:posts **37**

user:1:posts → [37, 20, 1]

Modellierung Beispiel 2



user:1:follows

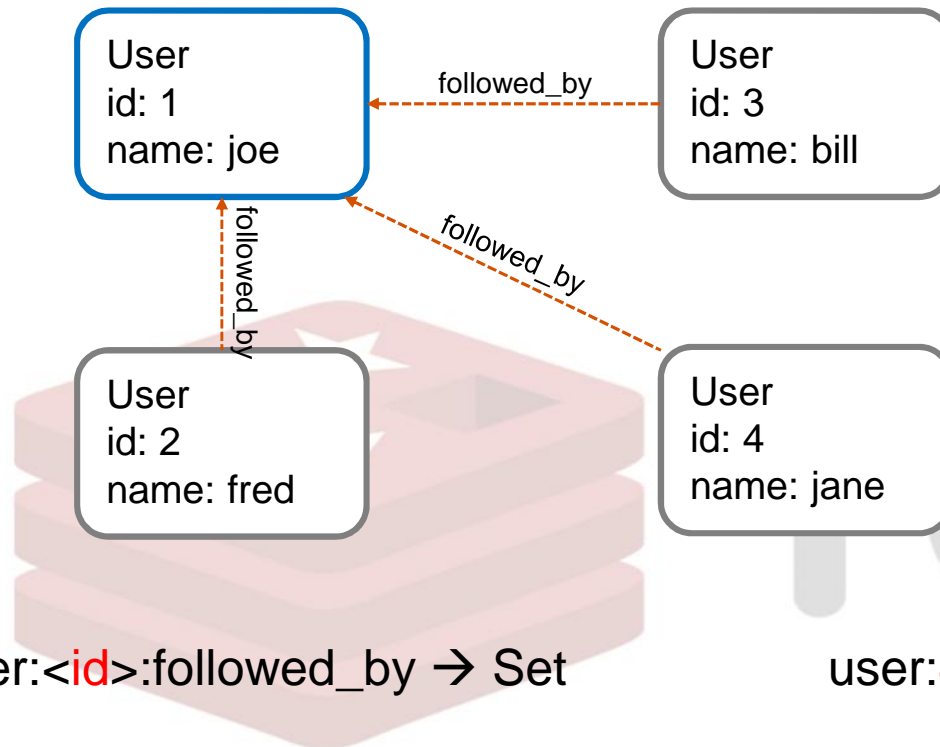
sadd user:1:follows 2

sadd user:1:follows 3

sadd user:1:follows 4

user:1:follows → {2, 3, 4}

Modellierung Beispiel 2



user:<id>:followed_by → Set

user:4:followed_by

sadd user:4:followed_by 1

user:4:followed_by → {1}

Beispiel aus: <http://no.gd/redis-presentation.pdf>

Modellierung Beispiel 2



Einfache Modellierung eines Social Networks

Keys

Values

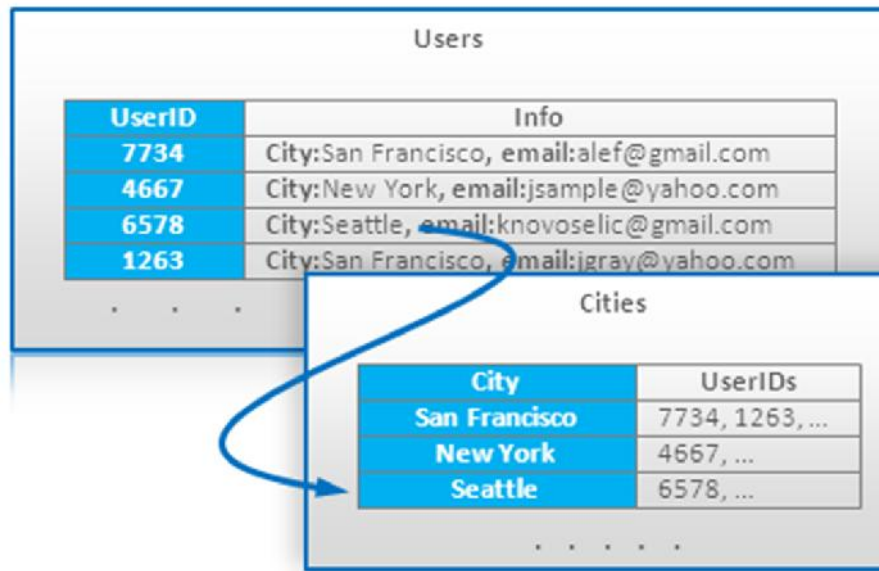
user:1:name	joe
user:2:name	fred
username:joe	1
username:fred	2
user:1:follows	{2,3,4}
user:2:follows	{1}
user:1:followed_by	{2}
user:2:followed_by	{1}
post:1:content	"Hello world"
post:1:user	2
post:2:content	"Good Morning"
post:2:user	1
user:1:posts	[2,3,4]
user:2:posts	[1,5,6]

← Set

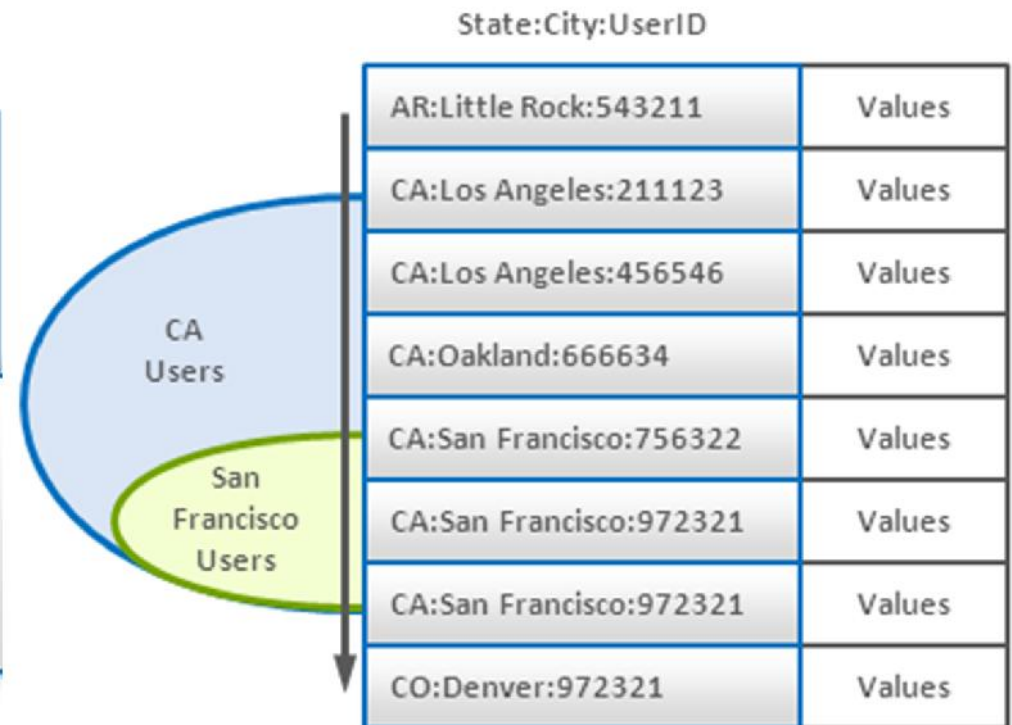
← List

redis

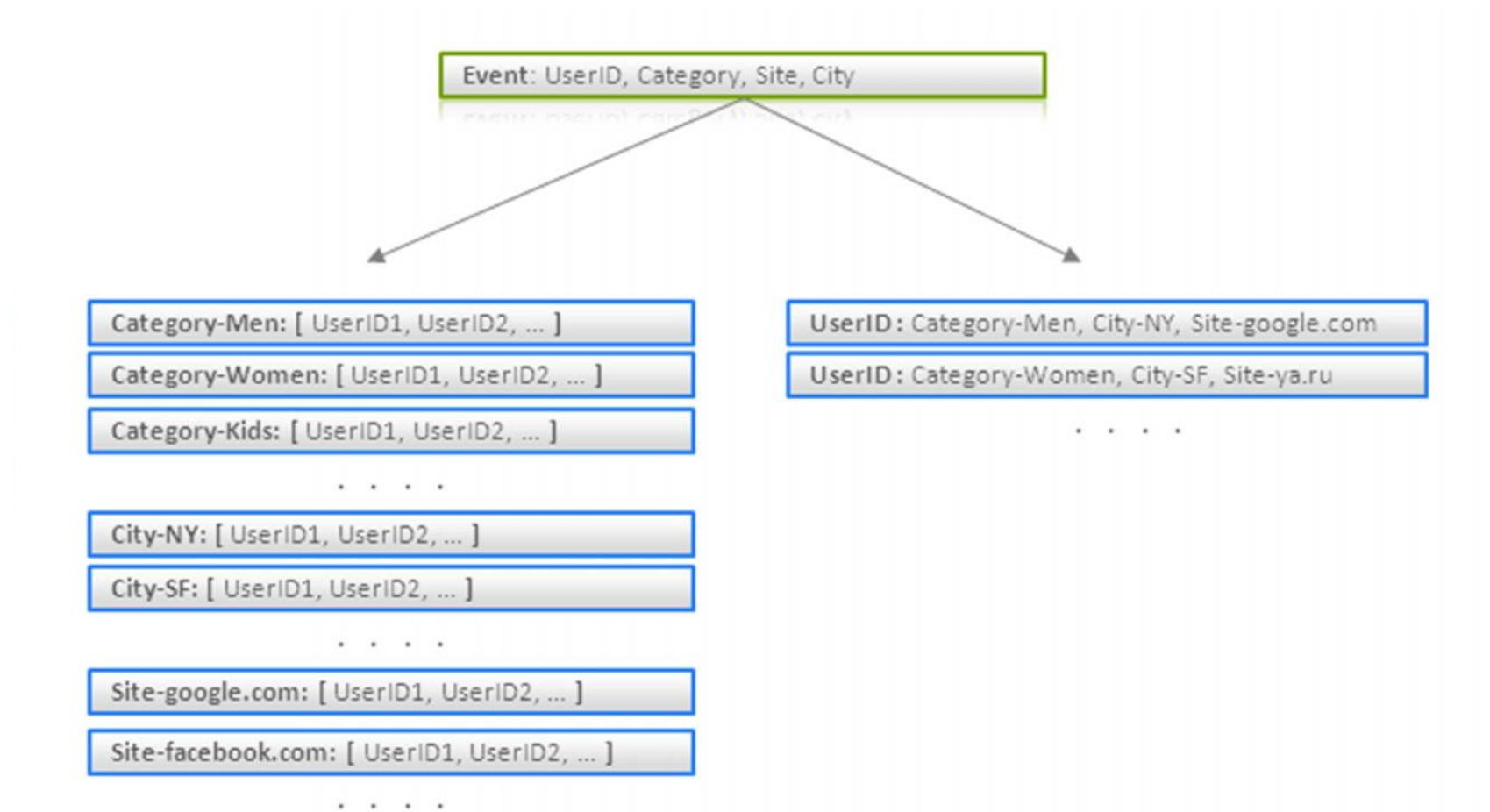
Index Tabelle



Composite Key



<https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/>



<https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/>

- Teil des Spring Data Projekts
- Integration von Redis in Spring Framework
- Abstraktionsebene für die Redis Client Bibliotheken
- Konfiguration für Verbindung zu Redis
- Template zur Nutzung von Redis in Spring
- Serialisierung der Schlüssel und Werte
- AtomicCounters (*RedisAtomicInteger*, *RedisAtomicLong*)
- Publish/Subscribe Unterstützung

Maven Dependency

```
<dependency>  
  <groupId>org.springframework.data</groupId>  
  <artifactId>spring-data-redis</artifactId>  
  <version>2.0.0.RELEASE</version>  
</dependency>
```

oder in einem Spring Boot Projekt

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-redis</artifactId>  
</dependency>
```

Verbindung zum Redis Server

```
@Configuration
public class RedisConfiguration {
    @Bean
    public JedisConnectionFactory getConnectionFactory() {
        JedisConnectionFactory jedisConnectionFactory =
            new JedisConnectionFactory();
        jedisConnectionFactory.setHostName("localhost");
        jedisConnectionFactory.setPort(6379);
        jedisConnectionFactory.setPassword("");
        return jedisConnectionFactory;
    }
}
```

RedisTemplate<k,v> - Klasse, Methoden für high level Datenzugriff
K - Redis Key Typ, normalerweise String
V - Redis Value Typ

```
@Bean(name= "redisTemplate")
RedisTemplate<String, Object> redisTemplate() {
    RedisTemplate<String, Object> template =
        new RedisTemplate<String, Object>();
    template.setConnectionFactory( getConnectionFactory() );
    return template;
}
```

StringRedisTemplate extends RedisTemplate<String,String>

```
@Bean(name= "stringRedisTemplate")
StringRedisTemplate stringRedisTemplate() {
    StringRedisTemplate stringTemplate = new StringRedisTemplate();
    stringTemplate.setConnectionFactory( getConnectionFactory() );
    return stringTemplate;
}
```

RedisTemplate Operationen

Bei diesen Methoden muss jedes Mal der Key angegeben werden

- *ValueOperations<K,V> opsForValue()*
`template.opsForValue().get("key1");`
- *ListOperations<K,V> opsForList()*
`template.opsForList().rightPush("key2", value);`
- *SetOperations<K,V> opsForSet()*
`template.opsForSet().intersect("k1", "k2");`
- *ZSetOperations<K,HK,HV> opsForZSet()*
`template.opsForZSet().rangeByLex("key3", range);`
- *HashOperations<K,HK,HV> opsForHash()*
`template.opsForHash().put("hkey", field, value);`

Für die vollständige Liste der Operationen siehe

<http://docs.spring.io/spring-data/data-redis/docs/current/api/org/springframework/data/redis/core/RedisTemplate.html>

RedisTemplate Operationen

Bei diesen Methoden werden die Operationen an einen Key gebunden

- *BoundHashOperations<K, HK, HV> boundHashOps(K key)*
- *BoundListOperations<K, V> boundListOps(K key)*
- *BoundSetOperations<K, V> boundSetOps(K key)*
- *BoundValueOperations<K, V> boundValueOps(K key)*
- *BoundZSetOperations<K, V> boundZSetOps(K key)*

Für die vollständige Liste der Operationen siehe

<http://docs.spring.io/spring-data/data-redis/docs/current/api/org/springframework/data/redis/core/RedisTemplate.html>


```
public class Example {  
    // inject the actual template  
    @Autowired  
    private RedisTemplate<String, String> template;  
  
    // inject the template for ListOperations  
    @Resource(name="redisTemplate")  
    private ListOperations<String, String> myListOps;  
  
    public void addValue(String userId, String value) {  
        // use template directly  
        template.opsForList().leftPush(userId, value);  
        template.boundListOps(userId).leftPush(value);  
        //or use operations with injected template  
        myListOps.leftPush(userId, value);  
    }  
}
```

Beispiel: Hash Datenstruktur

@Autowired

```
private RedisTemplate<String, String> template;
```

```
public User create(User user) {  
    String key = "user:"+user.getUsername();  
    template.opsForHash().put(key, "id", UUID.randomUUID().toString());  
    template.opsForHash().put(key, "firstName", user.getFirstName());  
    template.opsForHash().put(key, "lastName", user.getLastName());  
    template.opsForHash().put(key, "username", user.getUsername());  
    template.opsForHash().put(key, "password", user.getPassword());  
  
    template.opsForSet().add("user", key);  
    return user;  
}
```

Beispiel BoundOperations

```
private Contact buildContact(String key) {  
    Contact contact = new Contact();  
  
    BoundHashOps ops = template.boundHashOps(key);  
  
    contact.setId((Long) ops.get("id"));  
    contact.setEmailAddress((String) ops.get("emailAddress"));  
    contact.setFirstName((String) ops.get("firstName"));  
    contact.setLastName((String) ops.get("lastName"));  
    contact.setPhoneNumber((String) ops.get("phoneNumber"));  
    return contact;  
}
```

Cookies und Sessions

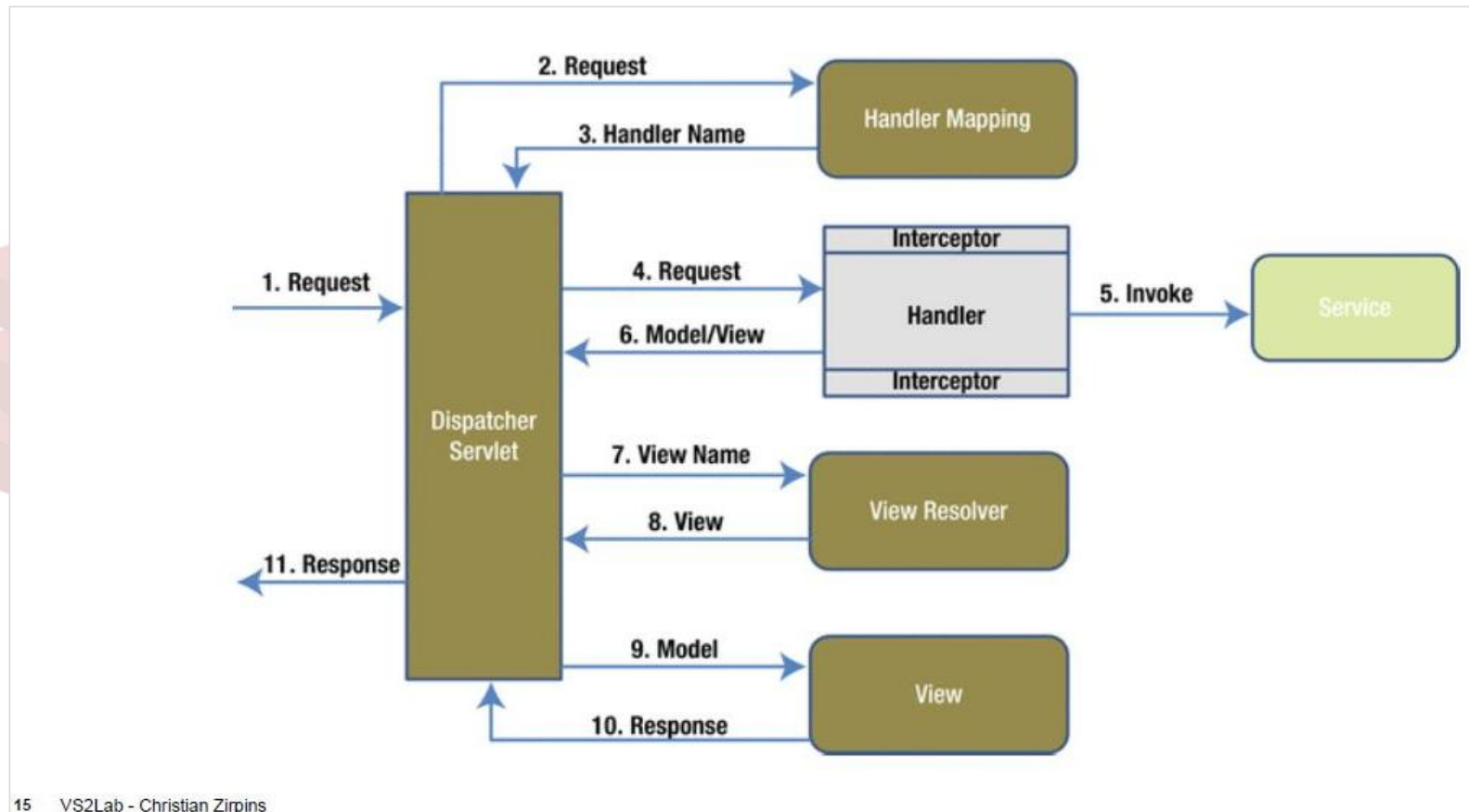
Cookie kleines Stück Text, gespeichert durch Browser auf dem PC des Users, z.B. zur Authentifizierung, Warenkorbinfos, Session Ids

Session serverseitige Speicherung von Information, die über Interaktionen hinweg erhalten bleiben soll



redis

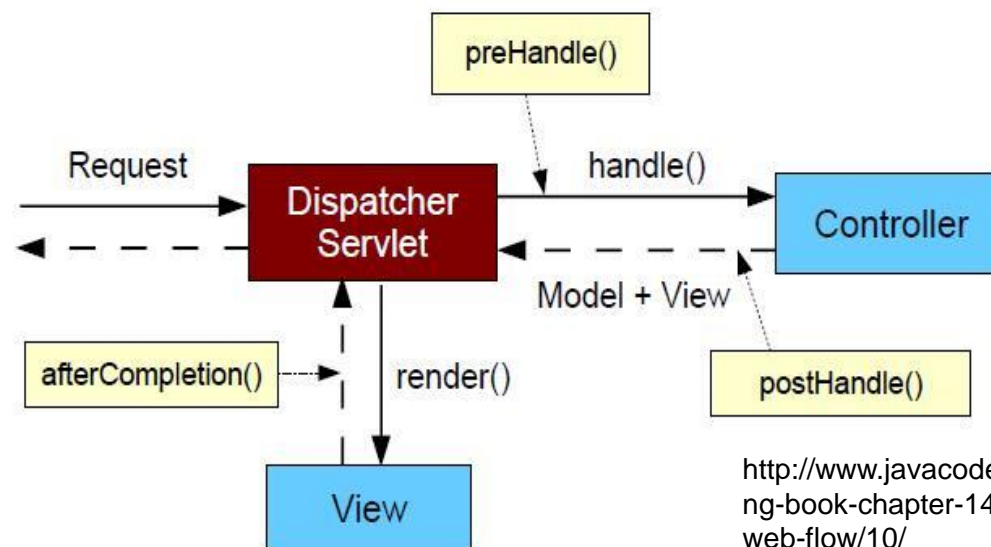
Spring MVC Architektur



15 VS2Lab - Christian Zirpins

Interceptor

- für Aufgaben, die für alle oder eine Einheit von Requests notwendig sind
- Requests können an drei Punkten unterbrochen werden:
 - vor Ausführung des Handlers (Controllers)
 - nach Ausführung des Handlers/Controllers, aber vor dem Rendern der View
 - nach dem Rendern der View



<http://www.javacodebook.com/2015/03/17/spring-book-chapter-14-spring-mvc-and-spring-web-flow/10/>

Interface HandlerInterceptor

```
public interface HandlerInterceptor{  
    boolean preHandle(HttpServletRequest request,  
        HttpServletResponse response, Object handler);  
    void postHandle(HttpServletRequest request,  
        HttpServletResponse response, Object handler,  
        ModelAndView modelAndView);  
    void afterCompletion(HttpServletRequest request,  
        HttpServletResponse response, Object handler, Exception ex);  
}
```

HandlerInterceptorAdapter

abstrakte Klasse, implementiert HandlerInterceptor Interface und dessen Methoden

```
public class SimpleInterceptor extends HandlerInterceptorAdapter {  
  
    private static final Logger logger =  
        Logger.getLogger(SimpleInterceptor.class);  
  
    public boolean preHandle(HttpServletRequest request,  
                             HttpServletResponse response,  
                             Object handler) throws Exception {  
        logger.info("Inside the prehandle");  
        return false;  
    }  
}
```

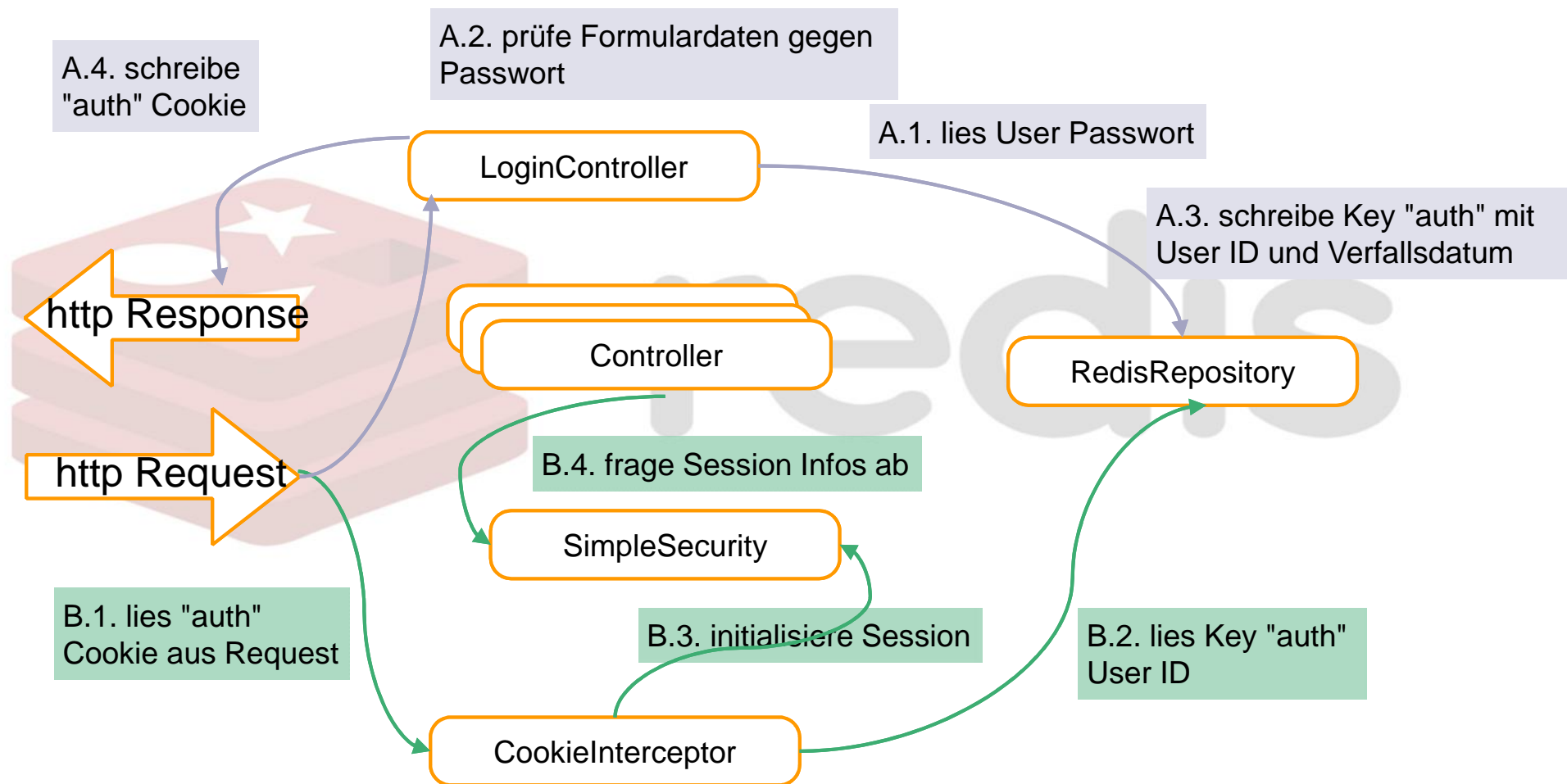

Spring MVC Interceptor Konfiguration

- Interceptoren in Spring Application Context Xml File konfigurieren oder
- In einer Java Klasse, die die Klasse **WebMvcConfigurerAdapter** erweitert, WebMvcConfigurerAdapter enthält die addInterceptors Methode, die als Zugriff zur InterceptorRegistry dient.

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = { "org.my.package" })
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LocaleChangeInterceptor());
        registry.addInterceptor(new SimpleInterceptor()).addPathPatterns("/auth/**");
    }
}
```

Mögliches Prinzip einer Session-Verwaltung

A: Erzeugen einer neuen Session nach Login



B: Nutzung der bestehenden Session bei Folgeanfragen

CookieInterceptor: Lesen von Session Tokens aus Request Cookies

- Ein spezifischer HandlerInterceptorAdapter liest bei jedem Request die Cookies aus und durchsucht sie nach einem gültigen Session Token.
- Existiert ein gültiges Token, dann wird damit die Session initialisiert.

```
public class SimpleCookieInterceptor extends HandlerInterceptorAdapter {
    @Autowired
    private StringRedisTemplate template;

    @Override
    public boolean preHandle(HttpServletRequest req, HttpServletResponse res, Object handler) throws Exception {
        Cookie[] cookies = req.getCookies();

        if (!ObjectUtils.isEmpty(cookies))
            for (Cookie cookie : cookies)
                if (cookie.getName().equals("auth")) {
                    String auth = cookie.getValue();
                    if (auth != null) {
                        String uid = template.opsForValue().get("auth:" + auth + ":uid");
                        if (uid != null) {
                            String name = (String) template.boundHashOps("uid:" + uid + ":user").get("name");
                            SimpleSecurity.setUser(name, uid);
                        }
                    }
                }

        return true;
    } ... // clean up SimpleSession State in the end (skipped here) }
```

SimpleSecurity: Utility-Klasse für Session Informationen

- Einfache Utility-Klasse, die den Login-Status pro Zugriff verwaltet.
- Die Klasse wird durch einen Interzeptor initialisiert und zurückgesetzt.

```
public abstract class SimpleSecurity {
    private static final ThreadLocal<UserInfo> user = new NamedThreadLocal<UserInfo>("microblog-id");

    private static class UserInfo {
        String name;
        String uid;
    }

    public static void setUser(String name, String uid) {
        UserInfo userInfo = new UserInfo();
        userInfo.name = name;
        userInfo.uid = uid;
        user.set(userInfo);
    }

    public static boolean isUserSignedIn(String name) {
        UserInfo userInfo = user.get();
        return userInfo != null && userInfo.name.equals(name);
    }

    public static boolean isSignedIn() { ... }
    public static String getName() { ... }
    public static String getUid() { ... }
    ... }
}
```

LoginController: Controller-Klasse zum Aufsetzen neuer Sessions

- Der Controller verarbeitet ein Login Formular mit User Daten.
- Authentifizierung und Token-Management erfolgen im Repository.
- Bei erfolgreichem Login wird ein Cookie mit dem Token gesetzt.

```
@Controller
public class LoginController {
    @Autowired private RedisRepository repository;
    private static final Duration TIMEOUT = Duration.ofMinutes(15);

    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public String login(@ModelAttribute("user") @Valid User user, HttpServletResponse response, Model model) {
        if (repository.auth(user.getName(), user.getPass())) {
            String auth = repository.addAuth(user.getName(), TIMEOUT.getSeconds(), TimeUnit.SECONDS);
            Cookie cookie = new Cookie("auth", auth);
            response.addCookie(cookie);
            model.addAttribute("user", user.getName());
            return "users/" + user.getName();
        }
        model.addAttribute("user", new User());
        return "login";
    }

    @RequestMapping(value = "/blog/logout", method = RequestMethod.GET)
    public String logout() {
        if (SimpleSecurity.isSignedIn()) {
            String name = SimpleSecurity.getName();
            repository.deleteAuth(name);
        }
        return "redirect:/";
    }
}
```

RedisRepository: zur Speicherung temporärer Session Tokens

- Das Repository speichert User Hashes (uid:1:user) sowie Auth Hashes (uid:1:auth) und reverse Keys (auth:X:uid) mit Verfallsdatum.
- Darauf basiert Authentifizierung und Token Management (add, delete).

```
@Repository
public class RedisRepository {
    @Autowired private StringRedisTemplate template;

    public boolean auth(String uname, String pass) {
        String uid = template.opsForValue().get("uname:" + uname + ":uid");
        BoundHashOperations<String, String, String> userOps = template.boundHashOps("uid:" + uid + ":user");
        return userOps.get("pass").equals(pass);    }

    public String addAuth(String uname, long timeout, TimeUnit tUnit) {
        String uid = template.opsForValue().get("uname:" + uname + ":uid");
        String auth = UUID.randomUUID().toString();
        template.boundHashOps("uid:" + uid + ":auth").put("auth", auth);
        template.expire("uid:" + uid + ":auth", timeout, tUnit);
        template.opsForValue().set("auth:" + auth + ":uid", uid, timeout, tUnit);
        return auth;    }

    public void deleteAuth(String uname) {
        String uid = template.opsForValue().get("uname:" + uname + ":uid");
        String authKey = "uid:" + uid + ":auth";
        String auth = (String) template.boundHashOps(authKey).get("auth");
        List<String> keysToDelete = Arrays.asList(authKey, "auth:" + auth + ":uid");
        template.delete(keysToDelete);    }}
```

Redis

- <https://redis.io/>
- <http://redis.io/topics/quickstart>
- <http://redis.io/topics/data-types-intro>
- <https://redislabs.com/ebook/redis-in-action>
- <http://try.redis.io/>
- <http://github.com/MicrosoftArchive/redis>
- http://blog.mjrusso.com/2010/10/17/redis-from-the-ground-up.html#heading_toc_j_0
- <http://de.slideshare.net/dvirsky/kicking-ass-with-redis>

Datenmodellierung

- <http://de.slideshare.net/ryanbriones/the-beauty-of-simplicity-mastering-database-design-with-redis>
- <https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/>
- <http://www.ebaytechblog.com/2014/10/10/nosql-data-modeling/>
- <https://entwickler.de/online/datenbanken/datenmodellierung-in-nicht-relationalen-datenbanken-137872.html>

Spring Data Redis

- <http://www.javabeat.net/spring-data-redis-example/>
- <https://www.packtpub.com/books/content/building-applications-spring-data-redis>
- <http://www.baeldung.com/spring-data-redis-tutorial>
- <https://omanandj.wordpress.com/2013/07/26/redis-using-spring-data-part-2-3/>
- [http://docs.spring.io/spring-data/data-redis/docs/current/api/org.springframework.data.redis/core/RedisTemplate.html](http://docs.spring.io/spring-data/data-redis/docs/current/api/org.springframework.data.redis.core.RedisTemplate.html)
- <http://redis.io/topics/twitter-clone>
- <http://www.ibm.com/developerworks/library/os-springredis/>
- <http://memorynotfound.com/spring-redis-application-configuration-example>
- <https://blog.openshift.com/build-cloud-enabled-java-redis-applications-with-spring-on-openshift/>
- <http://thysmichels.com/2014/01/11/spring-redis-cache-service-on-heroku/>
- <http://javakart.blogspot.de/2012/12/spring-data-redis-hello-world-example.html>

Demo-Projekt:

git clone https://github.com/zirpins/vs2lab.git

Stand 25.10.2017