

# Rest

rest是在spring-data 的基础上，可以将repository 自动输出为 rest资源，目前spring data rest支持 spring data jpa ， spring data mongodl gemfire 以及 spring data cassandra

这里以 spring data jpa为例。新建spring boot项目， 依赖为 JAP， Rest repository

java 实体类

```
@Entity
@Table(name="t_student")
public class Student {

    @Id
    @GeneratedValue
    private Integer id;

    @Column(name = "t_name")
    private String name;

    @Column(name = "t_age")
    private Integer age;
```

DAO

```
import org.springframework.data.jpa.repository.JpaRepository;

import com.jx.entity.Student;

public interface StudentDao extends JpaRepository<Student,
```

使用示例：注意实体Bean 是 Student， 访问url 是 Students 。若要 改成 127.0.0.1:8080/api/students 可以在 application.properties 配置 path=api



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8080/students/1'. The browser tabs include '应用', '在线工具', '开源', and 'Spring Framework'. The main content area displays a JSON response for a student with the name '三毛' and age 19. The response includes a self-link and a student link, both pointing to 'http://127.0.0.1:8080/students/1'.

```
{
  name: "三毛",
  age: 19,
  _links: {
    self: {
      href: "http://127.0.0.1:8080/students/1"
    },
    student: {
      href: "http://127.0.0.1:8080/students/1"
    }
  }
}
```

```
{
  - _embedded: {
    - students: [
      - {
        name: "三毛",
        age: 19,
        - _links: {
          - self: {
            href: "http://127.0.0.1:8080/students/1"
          },
          - student: {
            href: "http://127.0.0.1:8080/students/1"
          }
        }
      },
      - {
        name: "鲁迅",
        age: 20,
        - _links: {
          - self: {
            href: "http://127.0.0.1:8080/students/2"
          },
          - student: {
            href: "http://127.0.0.1:8080/students/2"
          }
        }
      },
      - {
        name: "张三丰",
        age: 7,
        - _links: {
          - self: {
            href: "http://127.0.0.1:8080/students/3"
          },
          - student: {
            href: "http://127.0.0.1:8080/students/3"
          }
        }
      }
    ]
  },
  - _links: {
    - self: {
      href: "http://127.0.0.1:8080/students/?page,size,sort",
      templated: true
    }
  }
}
```



rest.zip  
62.55KB

## 缓存

spring boot中支持以 spring.cache 开头来配置缓存。如果什么都不配置,使用默认缓存SimpleCacheConfiguration,即使用ConcurrentMap。使用缓存还需要在spring boot启动类上加一个注解@EnableCaching。

### 一些配置项

```
spring.cache.type= # 可选 generic, ehcache, hazelcast, infinispan, jcache  
# guava, simple, none  
spring.cache.cache-names= # 程序启动时创建缓存名称  
spring.cache.ehcache.config= # ehcache 配置文件地址  
spring.cache.hazelcast.config= # hazelcast 配置文件地址  
spring.cache.infinispan.config= # infinispan 配置文件地址  
spring.cache.jcache.config= # jcache 配置文件地址  
spring.cache.jcache.provider= #当多个 jcache 实现在类路径中的时候,指定 jcache  
spring.cache.guava.spec= # guava specs
```

### @Cacheable

@Cacheable 的作用 主要针对方法配置,能够根据方法的请求参数对其结果进行缓存,如果有缓存对象了,那么直接返回!缓存。

#### @Cacheable 作用和配置方法

参数	解释	example
value	缓存的名称,在 spring 配置文件中定义,必须指定至少一个	例如: @Cacheable(value="mycache") @Cacheable(value={"cache1","cache2"})
key	缓存的 key,可以为空,如果指定要按照 SpEL 表达式编写,如果不指定,则缺省按照方法的所有参数进行组合	@Cacheable(value="testcache",key="#userName")
condition	缓存的条件,可以为空,使用 SpEL 编写,返回 true 或者 false,只有为 true 才进行缓存	@Cacheable(value="testcache",condition="#userName.length()>2")

### @CachePut

@CachePut 的作用 主要针对方法配置,能够根据方法的请求参数对其结果进行缓存,和 @Cacheable 不同的是,它每次都会

#### @CachePut 作用和配置方法

参数	解释	example
value	缓存的名称,在 spring 配置文件中定义,必须指定至少一个	@CachePut(value="my cache")
key	缓存的 key,可以为空,如果指定要按照 SpEL 表达式编写,如果不指定,则缺省按照方法的所有参数进行组合	@CachePut(value="testcache",key="#userName")
condition	缓存的条件,可以为空,使用 SpEL 编写,返回 true 或者 false,只有为 true 才进行缓存	@CachePut(value="testcache",condition="#userName.length()>2")

## @CacheEvict

@CacheEvict 的作用 主要针对方法配置，能够根据一定的条件对缓存进行清空

### @CacheEvict 作用和配置方法

参数	解释	example
value	缓存的名称，在 spring 配置文件中定义，必须指定至少一个	@CacheEvict(value="my cache")
key	缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合	@CacheEvict(value="testcache",key="#userName")
condition	缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存	@CacheEvict(value="testcache",condition="#userName.length()>2")
allEntries	是否清空所有缓存内容，缺省为 false，如果指定为 true，则方法调用后将立即清空所有缓存	@CacheEvict(value="testcache",allEntries=true)
beforeInvocation	是否在方法执行前就清空，缺省为 false，如果指定为 true，则在方法还没有执行的时候就清空缓存，缺省情况下，如果方法执行抛出异常，则不会清空缓存	@CacheEvict(value="testcache",beforeInvocation=true)

更多缓存注解可以参考：[http://blog.csdn.net/qq\\_23121681/article/details/53995666](http://blog.csdn.net/qq_23121681/article/details/53995666)

项目示例：使用默认缓存



cache.zip  
67.45KB

## MongoDB

需要引入mongodb支持。以及配置文件

spring.data.mongodb.host=120.79.183.246

spring.data.mongodb.port=27016

spring.data.mongodb.database=mydb

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.mapping.Field;

@Document
public class People {

    @Id
    private String id;

    @Field(value = "t_name")
    private String name;

    @Field(value = "t_age")
    private Integer age;

    @Field(value = "t_friends")
    private List<People> friends;

    @Override
    public String toString() {
        return "People [id=" + id + ", name=" + name + ", age=" + age + ", friends=" + friends + "];"
    }
}

import org.springframework.data.mongodb.repository.MongoRepository;

import com.jx.entity.People;

public interface PeopleDao extends MongoRepository<People, String>{

}
```



```

@RestController
@RequestMapping("people")
public class PeopleController {

    @Autowired
    private PeopleService peopleService;

    @RequestMapping("add")
    public String add(People people) {
        String msg = "ok";
        try {
            peopleService.add(people);
        } catch (Exception e) {
            msg = "fail";
        }
        return msg;
    }

    @RequestMapping("query/{id}")
    public String query(@PathVariable("id") String id) {
        return peopleService.findOne(id).toString();
    }
}

@RunWith(SpringRunner.class)
@SpringBootTest
public class PeopleTest {

    @Test
    public void contextLoads() {}
    private MockMvc mockMvc; // 模拟MVC对象, 通过MockMvcBuilders.webAppContextSetup(this.wac).build()初始化
    @Autowired
    private WebApplicationContext wac; // 注入WebApplicationContext
    // @Autowired
    // private MockHttpSession session; // 注入模拟的http session
    // @Autowired
    // private MockHttpServletRequest request; // 注入模拟的http request
    @Before // 在测试开始前初始化工作
    public void setup() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(this.wac).build();
    }

    @Test
    public void testAdd() throws Exception {
        ResultActions resultActions = mockMvc.perform(
            post("/people/add")
                .param("id", "10001")
                .param("name", "张三丰")
                .param("age", "150")
        );

        MvcResult result = resultActions
            .andExpect(status().isOk()) // 断言访问状态码200
            .andReturn(); // 返回执行请求的结果
        System.out.println(result.getResponse().getContentAsString());
    }
}

```



mongodb.zip  
70.08KB

## Redis



redis.zip  
65.15KB

下面方法放到spring boot执行类中，可以自定义 redistemplate用来覆盖 spring 提供了。因为spring 提供的是基于二进制文件的查看不方便

```
@Bean
@SuppressWarnings({ "rawtypes", "unchecked" })
public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory
redisConnectionFactory)
    throws UnknownHostException {
    RedisTemplate<Object, Object> template = new RedisTemplate<Object,
Object>();
    template.setConnectionFactory(redisConnectionFactory);

    Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer(Object.class);
    ObjectMapper om = new ObjectMapper();
    om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
    om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
    jackson2JsonRedisSerializer.setObjectMapper(om);

    template.setValueSerializer(jackson2JsonRedisSerializer); //1
    template.setKeySerializer(new StringRedisSerializer()); //2

    template.afterPropertiesSet();
    return template;
}
```