□□□□□□ عمل لإطار الشامل الدليل □□

تمت تابة هذه النسخة المدعومة بـ □□□□□□□□□4□.

جدول المحتويات

▯ الخاتمة

مقدمة

دعمًا يوفر .▯▯▯▯ لغة باستخدام المؤسسات مستوى على تطبيقات لبناء شيوعًا الأطر أكثر أحد هو ▯▯▯▯▯▯ البيئي، ▯▯▯▯▯▯ نظام من مختلفة جوانب سنغطي المدونة، هذه في .▯▯▯▯ تطبيقات لتطوير شاملاً بنيويًا المتقدمة والميزات والاختبار، والجدولة، ،▯▯▯▯▯▯▯ خدمات وبناء البيانات، وإدارة ،▯▯▯▯ ▯▯▯▯▯▯ ▯▯▯▯ ذلك في بما مثل ▯▯▯▯▯▯ ▯▯▯▯▯▯ ▯▯▯▯ ▯▯▯.

عمل إطار ▯▯▯▯▯▯ ▯▯▯▯

تطبيقات تطوير عملية لتبسيط مصمم ،▯▯▯▯ لغة على يعتمد المصدر مفتوح عمل إطار هو ▯▯▯▯ ▯▯▯▯ ▯▯▯▯▯▯ تطبيقات بإنشاء للمطورين يسمح مما التكوين، وسهلة سريعة عمل بيئة ▯▯▯▯ ▯▯▯▯▯▯ يوفر .▯▯▯▯▯▯ عملية تسهل التي والأدوات المكتبات من العديد بدمج ▯▯▯▯ ▯▯▯▯▯▯ يتميز وسهولة. بسرعة بذاتها قائمة للتطبيقات. التلقائي والتكوين ،▯▯▯▯▯▯ أو ▯▯▯▯▯ عبر التبعيات وإدارة المدمج، ▯▯▯▯▯▯ مثل التطوير،

الرئيسية المميزات لـ ▯▯▯▯▯▯ ▯▯▯▯:

1. **التكوين سهولة**: يوفر ▯▯▯▯▯▯ ▯▯▯▯ تكوينًا تلقائيًا للعديد من الإعدادات، مما يقلل من الحاجة إلى تكوين يدوي.
2. **تطبيقات قائمة بذاتها**: يمكنك تشغيل تطبيقات ▯▯▯▯▯▯ ▯▯▯▯ كتطبيقات مستقلة دون الحاجة إلى خوادم خارجية.
3. **إدارة التبعيات**: يدعم ▯▯▯▯▯▯ ▯▯▯▯ إدارة التبعيات عبر ▯▯▯▯▯، و▯▯▯▯▯▯ مما يسهل عملية إضافة المكتبات المطلوبة.
4. **مراقبة وصيانة**: يوفر ▯▯▯▯▯▯ ▯▯▯▯ أدوات لمراقبة وصيانة التطبيقات، مثل ▯▯▯▯▯▯▯▯.
5. **تكامل مع ▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯**: يتكامل ▯▯▯▯▯▯ ▯▯▯▯ بسلاسة مع باقي إطار عمل ▯▯▯▯▯▯، مثل ▯▯▯▯▯▯ ▯▯▯ ▯▯▯▯▯▯، و▯▯▯▯▯▯ ▯▯▯، و▯▯▯▯▯ ▯▯▯▯▯▯، و▯▯▯▯▯▯▯▯.

مثال بسيط لتطبيق ▯▯▯▯▯▯ ▯▯▯▯:

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class HelloWorldApplication {

    public static void main(String[] args) {
        SpringApplication.run(HelloWorldApplication.class, args);
    }

    @GetMapping("/")
    public String hello() {
        return "Hello, World!";
    }
}
```

في هذا المثال، يتم إنشاء تطبيق □□□□□□ □□□□ بسيط يعرض رسالة □□□□□□, □□□□□!□ عند الوصول إلى المسار الرئيسي □/□.

كيفية تشغيل التطبيق:

1. قم بتنزيل المشروع أو إنشائه باستخدام □□□□□□ □□□□□□□□□□.
2. افتح المشروع في بيئة التطوير الخاصة بك مثل□ □□□□□□□□ □□□□ أو □□□□□□□□□.
3. قم بتشغيل التطبيق عن طريق تنفيذ الأمر `mvn spring-boot:run` في سطر الأوامر.
4. افتح المتصفح واذهب إلى `http://localhost:8080` لرؤية الرسالة □□□□□□, □□□□□!□.

يعد □□□□□□ □□□□ خيارًا مثاليًا لتطوير تطبيقات الويب والخدمات المصغرة □□□□□□□□□□□□□□□ بسرعة وكفاءة.

البدء مع □□□□□□ □□□□ □□□□ □□□□□□ □□□□ يجعل من السهل إنشاء تطبيقات □□□□□□ قائمة بذاتها وجاهزة للإنتاج. يأخذ وجهة نظر محددة حول منصة □□□□□□ والمكتبات الخارجية، مما يسمح لك بالبدء بأقل قدر من التهيئة.

- □ **الإعداد الأولي:** ابدأ بإنشاء مشروع □□□□□□ □□□□ جديد باستخدام □□□□□□ □□□□□□□□□□. يمكنك اختيار التبعيات التي تحتاجها، مثل □□□□□□ □□□، □□□□□□و □□□□ □□□، □□□□□□و □□□□ □□□□□□ □□□.

- **الشرح التوضيحي**: تعرف على الشرح التوضيحي الرئيسي مثل @SpringBootApplication، والذي هو مزيج من @Configuration، و@EnableAutoConfiguration، و@ComponentScan.
- **الخادم المدمج**: يستخدم خوادم مدمجة مثل ، أو ، أو لتشغيل تطبيقك، لذلك لا تحتاج إلى نشر ملفات على خادم خارجي.

حقن التبعيات حقن التبعيات أو هو مبدأ أساسي في إطار عمل . يسمح بإنشاء مكونات ذات اقتران ضعيف، مما يجعل الكود الخاص بك أكثر نمطية وأسهل في الاختبار.

- @Autowired: تُستخدم هذه الشرحة لحقن التبعيات تلقائيًا. يمكن تطبيقها على المُنشئات، الحقول، والطرق. ستقوم ميزة حقن التبعيات في بحقن الحبوب المتعاونة تلقائيًا في حبتك.

مثال على حقن الحقول:

```
@Autowired
private UserRepository userRepository;

//
```

مثال على حقن المُنشئ:

```
private final UserRepository userRepository;
```

```
```java
    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    //
{
```

مثال على حقن الأسلوب :
```

~~~
  private UserRepository userRepository;

```java
    @Autowired
    public void setUserRepository(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    //
{
~~~

- @Component, @Service, @Repository: هذه هي تخصيصات لـ @Component □□□□□□□□□□، تستخدم للإشارة إلى أن الفصل هو □□□□□□ □□□□. كما أنها تعمل كتلميحات لدور الفصل الذي تم تعليقه.
- @Component: هذا نمط عام لأي مكون يتم إدارته بواسطة □□□□□□. يمكن استخدامه لوضع علامة على أي فئة كـ □□□□ في □□□□□□□.

مثال:

```java
@Component
public class EmailValidator {

    public boolean isValid(String email) {
        //
        return true;
    }
}
```

- @Service: هذه الشرح هي تخصيص لـ @Component وثستخدم لتمييز الفئة كخدمة. تُستخدم عادةً في طبقة الخدمة، حيث يتم تنفيذ منطق الأعمال.

مثال:

```java
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;
```

```java
public User findUserById(Long id) {
    return userRepository.findById(id).orElse(null);
}
}
```

العربية: إلى الكود ترجمة

```java
public User findUserById(Long id) {
    return userRepository.findById(id).orElse(null);
}
}
```

فقط تكون الترجمة ترجمته. يتم ولا □□□□□□ برمجية بلغة مكتوب لأنه هو كما يبقى الكود ملاحظة: للكود. المرافقة الشروحات أو التوضيحية للنصوص

- أن إلى للإشارة ثستخدم @Component. للتعليمة تخصيص أيضًا هي التعليمة هذه :@Repository استثناءات بترجمة تقوم كما الكائنات. وحذف وتحديث والبحث واسترجاع لتخزين آلية توفر الفئة في البيانات إلى الوصول لاستثناءات الهرمي التسلسل إلى الاستمرارية □□□□□□.

  مثال:

  ```java
  @Repository
  public interface UserRepository extends JpaRepository<User, Long> {
      //
  }
  ```

عمل إطار وتساعد وإيجازًا، للقراءة قابلية أكثر بك الخاص □□□□□□ تكوين تجعل التوضيحية التعليقات هذه المختلفة. □□□□□□□ الحبوب بين التبعيات وربط إدارة في □□□□□□

مكونات بين للتواصل وسيلة □□□□□□□□ الأحداث ثعتبر ،□□□□□□ عمل إطار في □□□□□□ **في الأحداث** بينما ما، شيء حدوث حول إشعارات بإرسال للمكونات الآلية هذه تسمح مباشرة. غير بطريقة التطبيق باستخدام النموذج هذا تنفيذ يتم لها. والاستجابة الإشعارات هذه إلى الاستماع أخرى لمكونات يمكن ApplicationEvent و ApplicationListener.

□□□□□□ في الأحداث عمل كيفية

1. **مخصص حدث إنشاء**: يمكنك إنشاء حدث مخصص عن طريق توسيع الفئة ApplicationEvent.

```java
public class CustomEvent extends ApplicationEvent {
    private String message;

    public CustomEvent(Object source, String message) {
        super(source);
        this.message = message;
    }

    public String getMessage() {
        return message;
    }
}
```

2. **نشر الحدث**: يمكنك نشر الحدث باستخدام ApplicationEventPublisher.

```java
@Autowired
private ApplicationEventPublisher publisher;

public void publishCustomEvent(final String message) {
    CustomEvent customEvent = new CustomEvent(this, message);
    publisher.publishEvent(customEvent);
}
```

3. **الاستماع إلى الحدث**: يمكنك الاستماع إلى الحدث عن طريق تنفيذ واجهة ApplicationListener.

```java
@Component
public class CustomEventListener implements ApplicationListener<CustomEvent> {
    @Override
    public void onApplicationEvent(CustomEvent event) {
        System.out.println("Received custom event - " + event.getMessage());
    }
}
```

المهام فصل لتحقيق في الأحداث تُستخدم في الأحداث **استخدام الأحداث في**
البريد عبر إشعارات لإرسال استخدامها يمكن المثال، سبيل على الصيانة. قابلية وتحسين
البيانات. تغيير بعد المؤقت التخزين ذاكرة لتحديث أو جديد، مستخدم تسجيل بعد الإلكتروني

عند تحرير بريد إلكتروني إلى مستخدمين، قد تحتاج إلى إرسال بريد ترحيبي تلقائيًا. لنفترض أن لديك تطبيق إدارة المستخدمين، وتريد إرسال بريد ترحيبي تلقائيًا عند تسجيل مستخدم جديد.

1. إنشاء حدث تسجيل المستخدم:

```
public class UserRegisteredEvent extends ApplicationEvent {
    private String email;

    public UserRegisteredEvent(Object source, String email) {
        super(source);
        this.email = email;
    }

    public String getEmail() {
        return email;
    }
}
```

2. نشر الحدث عند تسجيل المستخدم:

```
@Service
public class UserService {
    @Autowired
    private ApplicationEventPublisher publisher;

    public void registerUser(String email) {
        // Logic to register user
        publisher.publishEvent(new UserRegisteredEvent(this, email));
    }
}
```

3. الاستماع إلى الحدث وإرسال البريد الإلكتروني:

```
@Component
public class EmailService implements ApplicationListener<UserRegisteredEvent> {
    @Override
    public void onApplicationEvent(UserRegisteredEvent event) {
        sendWelcomeEmail(event.getEmail());
    }
```

```
        private void sendWelcomeEmail(String email) {
            // Logic to send email
            System.out.println("Sending welcome email to " + email);
        }
    }
```

أكثر الكود يجعل مما المستخدم، تسجيل من طق عن الإلكتروني البريد إرسال من طق فصل يتم الطريقة، بهذه الصيانة. في وسهولة تنظيماً

إليها. والاستماع التطبيق في الأحداث إنشاء في الأحداث آلية لك تتيح

- الأحداث: المخصصة يمكنك إنشاء أحداث مخصصة عن طريق توسيع الفئة ApplicationEvent. على المثال: سبيل

```
public class MyCustomEvent extends ApplicationEvent {
    private String message;

public MyCustomEvent(Object source, String message) {
    super(source);
    this.message = message;
}
```

إلى: أعلاه الكود ترجمة تمت

```
public MyCustomEvent(Object source, String message) {
    super(source);
    this.message = message;
}
```

في هذا الكود، يتم تعريف مُنشئ فئة MyCustomEvent يأخذ كمعاملات Object source و String message. يتم استدعاء المُنشئ الأساسي باستخدام super(source)، ثم يتم تعيين قيمة الرسالة المخزنة في this.message.

```
    public String getMessage() {
        return message;
    }
}
```

- مستمعو الأحداث: استخدم @EventListener أو قم بتنفيذ ApplicationListener للتعامل مع الأحداث. على سبيل المثال:

```
@Component
public class MyEventListener {

@EventListener
public void handleMyCustomEvent(MyCustomEvent event) {
    System.out.println( "        Spring     - " + event.getMessage());
}
```

- ناشرو الأحداث: استخدم ApplicationEventPublisher لنشر الأحداث مخصصة. على سبيل المثال:

```
@Component
public class MyEventPublisher {

    @Autowired
    private ApplicationEventPublisher applicationEventPublisher;

public void publishCustomEvent(final String message) {
    System.out.println(  "       .    ");
    MyCustomEvent customEvent = new MyCustomEvent(this, message);
    applicationEventPublisher.publishEvent(customEvent);
}
}
```

[illegible] باستخدام البيانات إدارة

[illegible] هي جزء من مشروع [illegible] الذي يهدف إلى تبسيط [illegible] فوق طبقة تجريدية [illegible]. يوفر [illegible] باستخدام العلائقية البيانات قواعد إلى الوصول [illegible]، إنشاء، قراءة، تحديث [illegible] إلى الحاجة دون حذف [illegible] مما يجعل من السهل تنفيذ عمليات [illegible] التقليدي، المتكرر. الكود من الكثير كتابة

المميزات الرئيسية:

1. **تبسيط الكود**: تقليل من الحاجة إلى كتابة كود [illegible] التقليدي المتكرر.

2. **دعم الكيانات البسيطة**: يدعم الكيانات البسيطة التي تعكس جداول قاعدة البيانات.
3. **استعلامات مخصصة**: يسمح بكتابة استعلامات □□□ مخصصة بسهولة.
4. **تكامل مع □□□□□□**: يتكامل بسلاسة مع إطار عمل □□□□□□، مما يسهل إدارة الكيانات والتبعيات.

مثال بسيط:

```java
import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;

@Table("users")
public class User {
    @Id
    private Long id;
    private String name;
    private String email;

    // Getters and Setters
}

import org.springframework.data.repository.CrudRepository;

public interface UserRepository extends CrudRepository<User, Long> {
    User findByEmail(String email);
}
```

في هذا المثال، يتم تعريف كيان User الذي يعكس جدول users في قاعدة البيانات. يتم استخدام UserRepository للوصول إلى البيانات وتنفيذ عمليات □□□□ بسهولة.

الخلاصة: □□□□□□ □□□□ □□□□ هو أداة قوية لتبسيط الوصول إلى قواعد البيانات العلائقية باستخدام □□□□، مع الحفاظ على البساطة والمرونة.

يوفر □□□□□□ □□□□ □□□□ وصولًا بسيطًا وفعالًا إلى □□□□.

- □ **المستودعات □□□□□□□□□□□□□□**: قم بتعريف المستودعات لإجراء عمليات □□□□ □إنشاء، قراءة، تحديث، حذف□. على سبيل المثال:

  ```java
  public interface UserRepository extends CrudRepository<User, Long> {
  }
  ```

سبيل على مخصصة. استعلامات لتحديد @Query مثل التوضيحية التعليقات استخدم الاستعلامات: □
المثال:

```
@Query("SELECT * FROM users WHERE username = :username")
User findByUsername(String username);
```

تطوير تبسيط إلى يهدف الذي □□□□ □□□□□□ مشروع من جزء هو □□□ □□□□ □□□□□□ □□□ □□□□ □□□□□□ □□□ □□□□ □□□□□□ تجريدية طبقة □□□ □□□□ □□□□□□ يوفر البيانات. إلى الوصول تقنيات على تعتمد التي □□□□ تطبيقات الحاجة دون البيانات إلى الوصول عمليات تنفيذ عملية يسهل مما ،□□□□ □□□□□□□□□□□ □□□□□ □□□ فوق المتكرر. الكود من الكثير كتابة إلى

□□□ □□□□ □□□□□□ لـ الرئيسية الميزات

1. **الكود تقليل المتكرر:** يوفر □□□□□□ □□□□ □□□ واجهات برمجية جاهزة للاستخدام للتعامل مع الكيانات يدويًا. الكود كتابة إلى الحاجة دون حذف□ تحديث، قراءة، □إنشاء، □□□□ وعمليات □□□□□□□□□□
2. **الاستعلامات المخصصة:** يمكنك كتابة استعلامات مخصصة باستخدام لغة □□□□ □□□□□ □□□□□□□ البرمجية. الواجهات في الطرق أسماء استخدام طريق عن أو □□□□□ □□□□□ □□□□□□□□□□
3. **التكامل مع □□□□□□:** □□□□□□ □□□□ □□□ يتكامل بسلاسة مع إطار عمل □□□□□□، مما يسمح باستخدام □□□□□□ □□□□□□□□□□□ التبعيات وحقن □□□□□□□□□□□ □□□□□□□□□□□□ المعاملات إدارة مثل ميزات □□□□□.
4. **الدعم للاستعلامات الديناميكية:** يمكنك إنشاء استعلامات ديناميكية باستخدام Specification و Criteria API.
5. **التقسيم □□□□□□□□□□□□ والفرز □□□□□□□□□□:** يوفر □□□□□□ □□□□ □□□ دعمًا مدمجًا للتقسيم والفرز، مما يسهل التعامل مع مجموعات كبيرة من البيانات.

مثال بسيط

لنفترض أن لديك كيان User وتريد تنفيذ عمليات □□□□ عليه. يمكنك إنشاء واجهة برمجية تمتد JpaRepository كما يلي:

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

public interface UserRepository extends JpaRepository<User, Long> {
    //
}
```

مع للعامل [redacted] تحكم وحدة أو [redacted] خدمة في الواجهة هذه استخدام يمكنك ذلك، بعد البيانات:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    public User getUserById(Long id) {
        return userRepository.findById(id).orElse(null);
    }

    public User saveUser(User user) {
        return userRepository.save(user);
    }

    public void deleteUser(Long id) {
        userRepository.deleteById(id);
    }
}
```

الخلاصة

[redacted] باستخدام [redacted] تطبيقات في البيانات قواعد مع للعامل وسهلة فعالة طريقة يوفر [redacted]. بشأن القلق من بدلاً العمال من تطبيقاتك على التركيز يمكنك هو، الذي التجريد طبقة بفضل البيانات. إلى الوصول تنفيذ تفاصيل

يجعل من السهل تنفيذ مستودعات تعتمد على □□□ □□□□ □□□ □□□□□□.

- تعيين الكيانات: قم بتعريف الكيانات باستخدام @Entity وقم بتعيينها إلى جداول قاعدة البيانات. على سبيل المثال:

```java
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;
    // getters and setters
}
```

- **المستودعات □□□□□□□□□□□□□□**: أنشئ واجهات المستودعات عن طريق توسيع JpaRepository. على سبيل المثال:

```java
public interface UserRepository extends JpaRepository<User, Long> {
}
```

- طرق الاستعلام: استخدم طرق الاستعلام لإجراء عمليات قاعدة البيانات. على سبيل المثال:

```java
List<User> findByUsername(String username);
```

□□□□□□ □□□□ □□□□□ □□□□□□ □□□□ □□□□□ □□□□□ هو جزء من مشروع □□□□□□ □□□□ الذي يهدف إلى تبسيط الوصول إلى قواعد البيانات □□□□□، وتحديدًا □□□□□. □□□□□ هو مخزن بيانات مفتوح المصدر يعتمد على الذاكرة، ويستخدم كقاعدة بيانات، وذاكرة تخزين مؤقت، ووسيط للرسائل. يوفر □□□□□□ □□□□ □□□□□ طبقة تجريدية تسمح للمطورين بالتفاعل مع □□□□□ بسهولة باستخدام واجهات برمجية مألوفة في □□□□□□.

المميزات الرئيسية

1. **التكامل السلس مع □□□□□□**: يمكن دمج □□□□□□ □□□□ □□□□□ بسهولة مع تطبيقات □□□□□□ الأخرى، مما يوفر تجربة تطوير متسقة.
2. **دعم للأنواع المختلفة من البيانات**: يدعم □□□□□□ □□□□ □□□□□ أنواعًا مختلفة من البيانات مثل □□□□□□□, □□□□□□, □□□□□, □□□□, و □□□□□□ □□□□.
3. **التعامل مع الاتصالات**: يوفر إدارة اتصالات □□□□□ عبر RedisConnectionFactory، مما يسهل تكوين وإدارة الاتصالات.

4. **مع لماعتلل مع اناناكلا**: نكمي نييعت اناناكلا [illegible] إلى لكاهي اناناي [illegible] مادختساب `RedisTemplate` أو `Repository` [illegible].
5. **مع اماماعتساللل**: رفوي ًامعد اماماعتساللل مادختساب ريياعم ،ةطيسب امم حمسي ثحبلاب نع اناناي البال ةنزخمال في [illegible].

لاثم طيسب

امي ف يلي لاثم طيسب حضوي ةيفيك نيوكت مادختساو [illegible] في قيبطت [illegible]:

```
@SpringBootApplication
public class RedisExampleApplication {

    public static void main(String[] args) {
        SpringApplication.run(RedisExampleApplication.class, args);
    }

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory connectionFactory) {
        RedisTemplate<String, Object> template = new RedisTemplate<>();
        template.setConnectionFactory(connectionFactory);
        return template;
    }
}

@Service
public class RedisService {

    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    public void setValue(String key, Object value) {
        redisTemplate.opsForValue().set(key, value);
    }

    public Object getValue(String key) {
        return redisTemplate.opsForValue().get(key);
```

```
    }
}
```

في هذا المثال، نقوم بتكوين RedisTemplate واستخدامه لتخزين واسترجاع البيانات من □□□□□.

الخلاصة

□□□□□□ □□□□ □□□□□ □□□□□ □□□□ □□□□□□ يوفر طريقة قوية وسهلة للتفاعل مع □□□□□ في تطبيقات □□□□□□. سواء كنت تحقق على يساعدك أن يمكن □□□□□□ □□□□ □□□□□□ فإن موجود، تطبيق تحسين أو جديد تطبيق ببناء تقوم بفعالية. أهدافك

يوفر □□□□□□ □□□□ □□□□□ □□□□□ البنية التحتية للوصول إلى البيانات المعتمدة على □□□□□.

- □□□□□□□□□□□□□□: استخدم RedisTemplate للتفاعل مع □□□□□. على سبيل المثال:

```
@Autowired
private RedisTemplate<String, Object> redisTemplate;

public void save(String key, Object value) {
    redisTemplate.opsForValue().set(key, value);
}
```

الترجمة:

```
public void save(String key, Object value) {
    redisTemplate.opsForValue().set(key, value);
}
```

في هذه الدالة، يتم حفظ قيمة معينة □value□ في قاعدة بيانات □□□□□ باستخدام مفتاح محدد □key□. يتم استخدام redisTemplate للوصول إلى عمليات □□□□□ وحفظ القيمة.

```
public Object find(String key) {
    return redisTemplate.opsForValue().get(key);
}
```

تمت ترجمة الكود أعلاه إلى:

```
public Object find(String key) {
    return redisTemplate.opsForValue().get(key);
}
```

ترجمته. يتم ولا [illegible] برمجية بلغة مكتوب لأنه يتغير لم الكود ملاحظة:

- المثال: سبيل على .@Repository باستخدام [illegible] مستودعات إنشاء قم مستودعات:

```
@Repository
public interface RedisRepository extends CrudRepository<RedisEntity, String> {
}
```

إلى الوصول [illegible] نظائر [illegible] ودعم [illegible] المعاملات إدارة [illegible] يسهل [illegible] ودعم المعاملات.

- المثال: سبيل على المعاملات. لإدارة @Transactional استخدم المعاملات: إدارة

```
@Transactional
public void saveUser(User user) {
    userRepository.save(user);
}
```

- المثال: سبيل على التخزين. من لفصل [illegible] نمط [illegible] نمط: قم بتنفيذ نمط [illegible]

```
public class UserDao {
    @Autowired
    private JdbcTemplate jdbcTemplate;

public User findById(Long id) {
    return jdbcTemplate.queryForObject("SELECT * FROM users WHERE id = ?", new Object[]{id}, new UserRo
}
}
```

العربية: إلى الكود ترجمة

```
public User findById(Long id) {
    return jdbcTemplate.queryForObject("SELECT * FROM users WHERE id = ?", new Object[]{id}, new UserRo
}
}
```

ترجمته. يتم ولا [illegible] برمجية بلغة مكتوب لأنه هو كما يبقى الكود ملاحظة:

□□□□ و □□□ □□□□ □□□□□ □□□□□□□□□ □□□□□□□□□□□□□ هي واجهة برمجة تطبيقات □□□□□ تُستخدم في لغة □□□□ للتفاعل مع قواعد البيانات. تسمح □□□□ للتطبيقات بتنفيذ استعلامات □□□ وإدارة البيانات في قواعد البيانات العلائقية. توفر □□□□ مجموعة من الواجهات والفئات التي تمكن المطورين من الاتصال بقاعدة البيانات، وإرسال استعلامات □□□، ومعالجة النتائج.

□□□ □□□□□□□□□□□□□□□□□□ □□□□□□□□ هي تقنية تُستخدم لتحويل البيانات بين نظام أنواع غير متوافق في لغات البرمجة الموجهة للكائنات مثل □□□□□ وقواعد البيانات العلائقية. تُسهل □□□ على المطورين العمل مع قواعد البيانات باستخدام الكائنات بدلاً من كتابة استعلامات □□□ مباشرة. من أشهر أدوات □□□ في □□□□ هي □□□□□□□□□ و □□□ □□□□□ □□□□□□□□□□□ □□□□.

الفرق بين □□□□ و □□□: □ □□□□: يتطلب كتابة استعلامات □□□ يدويًا وإدارة الاتصالات والنتائج بشكل صريح. □ □□□: يخفف من الحاجة إلى كتابة استعلامات □□□ يدويًا عن طريق تعيين الكائنات إلى جداول قاعدة البيانات تلقائيًا.

مثال بسيط باستخدام □□□□:

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcExample {
    public static void main(String[] args) {
        try {
            Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "us
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT * FROM users");

            while (resultSet.next()) {
                System.out.println(resultSet.getString("username"));
            }

            resultSet.close();
            statement.close();
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
}
```

لاثم طيسب مادختساب □□□ □□□□□□□□□□□□:

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateExample {
    public static void main(String[] args) {
        SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
        Session session = sessionFactory.openSession();
        session.beginTransaction();

        User user = new User();
        user.setUsername("john_doe");
        session.save(user);

        session.getTransaction().commit();
        session.close();
        sessionFactory.close();
    }
}
```

في نيلاثمال ،اهلاعأ كنكمي أن ىرت فيك أن □□□ طسبئ ةيلمع لعافتال مع ةدعاق اناي□بال ةنراقم بـ □□□□.

رفوي □□□□□□ امًعدأ لماش لـ □□□□ و □□□ نيي□عت□ اتانئاكلا إلى □. الع□اقات

□ □□□□□□□□□□□□□: طيسبت اتيلمع □□□□ مادختساب JdbcTemplate. ىلع ليبس :لاثمال

```
    @Autowired
    private JdbcTemplate jdbcTemplate;

  public List<User> findAll() {
      return jdbcTemplate.query("SELECT * FROM users", new UserRowMapper());
  }
```

العربية: إلى أعلاه الكود ترجمة

```
public List<User> findAll() {
    return jdbcTemplate.query("SELECT * FROM users", new UserRowMapper());
}
```

في هذا الكود، يتم استخدام دالة findAll التي تقوم باسترجاع قائمة من الكائنات من نوع User. يتم تنفيذ استعلام jdbcTemplate.query الذي يقوم بجلب جميع السجلات من جدول users، ويتم تحويل النتائج إلى كائنات User باستخدام UserRowMapper.

- **[illegible]**: قم بدمج [illegible] مع [illegible] لدعم [illegible]. على سبيل المثال:

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;
    // getters and setters
}
```

[illegible] خدمات بناء

في عالم [illegible] في [illegible] يوفر [illegible] عدة خيارات لإنشاء عملاء [illegible] للتفاعل مع خدمات الويب. فيما يلي بعض الأدوات الرئيسية التي يمكن استخدامها:

1. **[illegible]**:
   RestTemplate هو أداة كلاسيكية في [illegible] لتنفيذ طلبات [illegible]. إنه متزامن [illegible] وواجهة برمجية بسيطة لإرسال الطلبات واستقبال الاستجابات.

   ```
   RestTemplate restTemplate = new RestTemplate();
   String url = "https://api.example.com/resource";
   String response = restTemplate.getForObject(url, String.class);
   System.out.println(response);
   ```

2. **[illegible]**:
   WebClient هو بديل غير متزامن [illegible] لـ RestTemplate مصمم على [illegible] [illegible]. إنه مناسب للتطبيقات التي تحتاج إلى التعامل مع طلبات غير متزامنة.

```
WebClient webClient = WebClient.create("https://api.example.com");
Mono<String> response = webClient.get()
    .uri("/resource")
    .retrieve()
    .bodyToMono(String.class);
response.subscribe(System.out::println);
```

3. [illegible]:

   RestClient وه ةجاو ةيجمرب ةديدج في [illegible] 6 فدهت ىلإ طيسبت ءاشنإ ءلمع [illegible]. هنإ رفويو ةجاو ةيجمرب رثكأ ةثادح. ةنورومو

```
RestClient restClient = RestClient.create();
String response = restClient.get()
    .uri("https://api.example.com/resource")
    .retrieve()
    .body(String.class);
System.out.println(response);
```

4. [illegible] [illegible]:

   Feign وه ةبتكم حمست ءاشنإب ءلمع [illegible] ةقيرطب ةينالعإ [illegible]. نكمي امادختسا عم [illegible] [illegible] ءاشنإل ءلمع [illegible]. ةبسهولة

```
@FeignClient(name = "exampleClient", url = "https://api.example.com")
public interface ExampleClient {
    @GetMapping("/resource")
    String getResource();
}
```

5. [illegible] [illegible]:

   RestEasy وه ةبتكم رخأ نكمي امادختسا ءاشنإل ءلمع [illegible]. امنإ رفوت ةجاو ةيجمرب ةيوق ذيفنتل تابلط [illegible].

```
ResteasyClient client = new ResteasyClientBuilder().build();
ResteasyWebTarget target = client.target("https://api.example.com/resource");
String response = target.request().get(String.class);
System.out.println(response);
```

يكون فقد متزامنة، بيئة في تعمل كنت إذا بك. الخاص التطبيق احتياجات على تعتمد المناسبة الأداة اختيار الأفضل. الخيار هو WebClient فإن متزامنة، غير مع الحاجة إلى بحاجة كنت إذا أما جيدًا. خيارًا RestTemplate [illegible] لعجي من لهسا الساء بناء ءلمع [illegible].

□ □□□□□□□□□□□□: استخدم RestTemplate لإجراء طلبات □□□□. على سبيل المثال:

```java
@Autowired
private RestTemplate restTemplate;

public String getUserInfo(String userId) {
    return restTemplate.getForObject("https://api.example.com/users/" + userId, String.class);
}
```

تمت ترجمة الكود أعلاه إلى:

```java
public String getUserInfo(String userId) {
    return restTemplate.getForObject("https://api.example.com/users/" + userId, String.class);
}
```

ملاحظة: الكود لم يتغير لأنه مكتوب بلغة برمجية □□□□□□ ولا يتم ترجمته.

□ □□□□□□□□□□: استخدم WebClient للطلبات التفاعلي غير المتزامنة. على سبيل المثال:

```java
@Autowired
private WebClient.Builder webClientBuilder;

public Mono<String> getUserInfo(String userId) {
    return webClientBuilder.build()
            .get()
            .uri("https://api.example.com/users/" + userId)
            .retrieve()
            .bodyToMono(String.class);
}
```

□□□□□□□□□□□ □□□□□□□□□□□ هي مكتبة تُستخدم في تطبيقات □□□□□□ □□□□□ لتبسيط عملية استدعاء خدمات □□□□□□□. توفر واجهة برمجية سهلة الاستخدام لإنشاء عميل □□□□، مما يسمح للمطورين بالتفاعل مع خدمات الويب دون الحاجة إلى كتابة كود □□□□ يدويًا. باستخدام □□□□□□□□□□، يمكنك تعريف واجهة برمجية مع التعليقات التوضيحية □□□□□□□□□□□□□ التي تصف كيفية استدعاء خدمة معينة، وتقوم المكتبة بإنشاء التنفيذ المناسب تلقائيًا.

مثال بسيط لاستخدام □□□□□□□□□□:

```java
@FeignClient(name = "example-service", url = "https://api.example.com")
public interface ExampleServiceClient {

    @GetMapping("/example-endpoint")
    String getExampleData();
}
```

التوضيحي التعليق تستخدم التي ExampleServiceClient برمجية واجهة تعريف يتم المثال، هذا في التعليق تستخدم getExampleData الطريقة بها. الخاص وعنوان الخدمة اسم لتحديد @FeignClient سيقوم الطريقة، هذه استدعاء عند استدعاؤها. سيتم التي النهاية نقطة لتحديد @GetMapping التوضيحي وإرجاع https://api.example.com/example-endpoint إلى طلب بإجراء تلقائيًا النتيجة.

للتحميل التلقائي التوازن مثل المتقدمة الميزات من العديد أيضًا يدعم وغيرها. والتشفير، الأخطاء، مع والتعامل،

تصريحي. ويب خدمة عميل هو

- @FeignClient. باستخدام التوضيحية التعليقات مع واجهات وأنشئ مشروعك إلى أضف الإعداد: المثال: سبيل على

```java
@FeignClient(name = "user-service", url = "https://api.example.com")
public interface UserServiceClient {
    @GetMapping("/users/{id}")
    String getUserInfo(@PathVariable("id") String userId);
}
```

- الأخطاء تشفير وفك المُعترضات باستخدام عملاء تخصيص التهيئة: . المثال: سبيل على

```java
@Bean
public RequestInterceptor requestInterceptor() {
    return requestTemplate -> requestTemplate.header("Authorization", "Bearer token");
}
```

والجدولة المهام، الإلكتروني، البريد

الإلكتروني. البريد لإرسال دعمًا يوفر **الإلكتروني البريد دعم**

 [illegible]: استخدم JavaMailSender لإرسال البريد الإلكتروني. المثال: سبيل على

```
    @Autowired
    private JavaMailSender mailSender;

public void sendEmail(String to, String subject, String body) {
     SimpleMailMessage message = new SimpleMailMessage();
     message.setTo(to);
     message.setSubject(subject);
     message.setText(body);
     mailSender.send(message);
}
```

 [illegible]: إنشاء رسائل بريد إلكتروني غنية بالمرفقات وتحتوي [illegible]. المثال: سبيل على

```
    @Autowired
    private JavaMailSender mailSender;

public void sendRichEmail(String to, String subject, String body, File attachment) throws MessagingEx
     MimeMessage message = mailSender.createMimeMessage();
     MimeMessageHelper helper = new MimeMessageHelper(message, true);
     helper.setTo(to);
     helper.setSubject(subject);
     helper.setText(body, true);
     helper.addAttachment(attachment.getName(), attachment);
     mailSender.send(message);
}
```

[illegible] دعم [illegible] لتنفيذ المهام المجدولة والمهام التي تستغرق وقتًا طويلًا من العمل في الخلفية.

 @Scheduled: جدولة المهام باستخدام @Scheduled. المثال: سبيل على

```
    @Scheduled(fixedRate = 5000)
    public void performTask() {
        System.out.println("Scheduled task running every 5 seconds");
    }
```

 المهام غير المتزامنة: تشغيل المهام بشكل غير متزامن باستخدام @Async. المثال: سبيل على

```java
@Async
public void performAsyncTask() {
    System.out.println("Async task running in background");
}
```

الاختبار في □□□□□□

عند العمل مع إطار عمل □□□□□□، يعد الاختبار جانبًا مهمًا لضمان جودة الكود وسلوك التطبيق كما هو متوقع. يوفر □□□□□□ مجموعة من الأدوات والمكتبات التي تسهل كتابة الاختبارات للفئات والمكونات المختلفة.

أنواع الاختبارات في □□□□□□

1. **اختبار الوحدة** □□□□ □□□□□□□ □: يتم اختبار كل وحدة □عادةً ما تكون كلاس□ بشكل منفصل عن باقي التطبيق. يمكن استخدام إطار عمل مثل □□□□□ لكتابة هذه الاختبارات.
2. **اختبار التكامل** □□□□□□□ □□□□□□□□□□□ □: يتم اختبار تفاعل عدة وحدات معًا للتأكد من أنها تعمل بشكل صحيح مع بعضها البعض. يوفر □□□□□□ إمكانية اختبار تكامل المكونات معًا.
3. **اختبار من البداية إلى النهاية** □□□□□□□ □□□□□□□□□□ □: يتم اختبار التطبيق بأكمله من البداية إلى النهاية، عادةً باستخدام أدوات مثل □□□□□□□□ أو □□□□□□ □□□□ □□□□.

أدوات الاختبار في □□□□□□

- □□□□□: إطار عمل شائع لكتابة الاختبارات في □□□□.
- □□□□□□□: مكتبة لإنشاء كائنات وهمية □□□□□□□ لاختبار التفاعلات بين الكائنات.
- □□□□□□ □□□□: يوفر □□□□□□ إطار عمل خاص للاختبار يتكامل مع □□□□□ ويوفر ميزات مثل تحميل سياق التطبيق □□□□□□□□□□□□□□□□□□□□□ للاختبارات.

مثال على اختبار وحدة باستخدام □□□□□ و □□□□□□□

```java
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
```

```
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

public class MyServiceTest {

    @Mock
    private MyRepository myRepository;

    @InjectMocks
    private MyService myService;

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    public void testGetData() {
        when(myRepository.findData()).thenReturn("Mocked Data");

        String result = myService.getData();

        assertEquals("Mocked Data", result);
        verify(myRepository, times(1)).findData();
    }
}
```

في هذا المثال، يتم اختبار MyService باستخدام MyRepository وهمي كائن [illegible]. يتم التحقق من أن findData() يتم استدعاؤها مرة واحدة فقط. وأن التوقع المقدم getData() يعيد

اختبار التكامل باستخدام [illegible]

```
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

import org.junit.jupiter.api.Test;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;

@SpringBootTest
@AutoConfigureMockMvc
public class MyControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testGetEndpoint() throws Exception {
        mockMvc.perform(get("/my-endpoint"))
                .andExpect(status().isOk())
                .andExpect(content().string("Expected Response"));
    }
}
```

أن من للتحقق MockMvc باستخدام MyController في □□□□□□□□□□ نهاية نقطة اختبار يتم المثال، هذا في متوقع. هو كما هي الاستجابة

الخلاصة باستخدام الصحيح. وسلوكه التطبيق جودة لضمان أساسية عملية هو □□□□□□ في الاختبار مختلف تغطي فعالة اختبارات كتابة يمكنك، □□□□ □□□□□□ و □□□□□□□ و □□□□□ مثل المناسبة الأدوات التطبيق. جوانب

وهمية كائنات لإنشاء تُستخدم □□□□ عالم في شائعة مكتبة هي □□□□□□□ □□□□□□□ **باستخدام الاختبار** سلوك بمحاكاة لك السماح طريق عن الكود اختبار عملية □□□□□□□ تُسهل الوحدة. اختبارات في □□□□□□□ وسرعة. تركيزًا أكثر الاختبارات يجعل مما الخارجية، أو المعقدة الكائنات

لماذا نستخدم □□□□□□□؟

1. **عزل الكود:** تسمح لك □□□□□□□ باختبار الكود بشكل معزول عن التبعيات الخارجية مثل قواعد البيانات أو خدمات الويب.

2. **التحكم في السلوك**: يمكنك تحديد السلوك المتوقع للكائنات الوهمية، مما يسمح لك باختبار سيناريوهات مختلفة بسهولة.
3. **التحقق من التفاعلات**: يمكنك التحقق من أن الكود الذي تقوم باختباره يتفاعل مع الكائنات الوهمية بالطريقة المتوقعة.

مثال بسيط لنفترض أن لديك فئة UserService تعتمد على UserRepository لاسترداد بيانات المستخدم. يمكنك استخدام □□□□□□□ لإنشاء كائن وهمي لـ UserRepository واختبار UserService دون الحاجة إلى قاعدة بيانات حقيقية.

```java
import static org.mockito.Mockito.*;
import static org.junit.Assert.*;

public class UserServiceTest {

    @Test
    public void testGetUser() {
        //              UserRepository
        UserRepository userRepository = mock(UserRepository.class);

        //
        when(userRepository.findUserById(1)).thenReturn(new User(1, "John Doe"));

        //         UserService
        UserService userService = new UserService(userRepository);

        //          getUser
        User user = userService.getUser(1);

        //
        assertEquals("John Doe", user.getName());
    }
}
```

في هذا المثال: □ قمنا بإنشاء كائن وهمي لـ UserRepository باستخدام mock(UserRepository.class). □ قمنا بتعريف السلوك المتوقع للكائن الوهمي باستخدام when(...).thenReturn(...). □ قمنا باختبار طريقة getUser في UserService والتحقق من النتيجة باستخدام assertEquals.

والتحكم وهمية كائنات بإنشاء لك تسمح حيث ،[illegible] في الوحدة لاختبار قوية أداة هي [illegible] الخاصة
في سلوكها. بسهولة باستخدام [illegible]، كنك يمكن عزل الذي الكود تريد اختباره من والتحقق التفاعلات عه
في العمل. بشكل الأخرى التبعيات

الاختبار. لأغراض [illegible] الوهمية الكائنات لإنشاء قوية مكتبة هي [illegible]

المثال: سيبين على وهمية. كائنات لإنشاء @InjectMocks و @Mock استخدام التبعيات: محاكاة

```
@RunWith(MockitoJUnitRunner.class)
public class UserServiceTest {
    @Mock
    private UserRepository userRepository;

@InjectMocks
private UserService userService;

@Test
public void testFindUserById() {
    User user = new User();
    user.setId(1L);
    Mockito.when(userRepository.findById(1L)).thenReturn(Optional.of(user));
}
```

ترجمة:

```
@Test
public void testFindUserById() {
    User user = new User();
    user.setId(1L);
    Mockito.when(userRepository.findById(1L)).thenReturn(Optional.of(user));
}
```

ملاحظة: وتغيرات دوال أسماء على يحتوي لأنه ترجمته يتم لم البرمجية الكتلة في الموجود الكود
البرمجة. في عادةً ترجمتها يتم لا أمور وهي بالإنجليزية،

```
        User result = userService.findUserById(1L);
        assertNotNull(result);
        assertEquals(1L, result.getId().longValue());
    }
}
```

إلى: أعلاه الكود ترجمة تم

```
        User result = userService.findUserById(1L);
        assertNotNull(result);
        assertEquals(1L, result.getId().longValue());
    }
}
```

في ثترجم لا عادةً وهي بالإنجليزية، ووظائف متغيرات أسماء على يحتوي لأنه ترجمته يتم لم الكود ملاحظة:
البرمجة.

- سبيل على .□□□□□□□□□ □□□□□ الوهمية الكائنات مع التفاعلات من التحقق السلوك: من التحقق
  المثال:

```
Mockito.verify(userRepository, times(1)).findById(1L);
```

.□□□ □□□□□□ تحكم وحدات اختبار لك يتيح □□□□□□□ □□□□□□□ باستخدام الاختبار

- المثال: سبيل على بك. الخاصة الاختبار فئات في MockMvc بتكوين قم الإعداد:

```
@RunWith(SpringRunner.class)
@WebMvcTest(UserController.class)
public class UserControllerTest {
    @Autowired
    private MockMvc mockMvc;

@Test
public void testGetUser() throws Exception {
    mockMvc.perform(get("/users/1"))
        .andExpect(status().isOk())
        .andExpect(content().contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath("$.id").value(1));
}
}
```

إلى: أعلاه الكود ترجمة تمت

```java
@Test
public void testGetUser() throws Exception {
    mockMvc.perform(get("/users/1"))
        .andExpect(status().isOk())
        .andExpect(content().contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath("$.id").value(1));
}
}
```

يتم ال تقنية أسماء وهي بالإنجليزية، ومتغيرات وظائف أسماء على يحتوي لأنه يتغير لم الكود ملاحظة:
عادةً. ترجمتها

- المثال: سبيل على. □□□□ طلبات لمحاكاة الطلبات بناة استخدم الطلبات: بناة

```java
mockMvc.perform(post("/users")
    .contentType(MediaType.APPLICATION_JSON)
    .content("{\"username\":\"john\",\"password\":\"secret\"}"))
    .andExpect(status().isCreated());
```

والإدارة المراقبة

ميزات يوفر الذي □□□□ □□□□□□ عمل إطار من جزء هو □□□□□□□□□ □□□□ □□□□□□ □□□□□□□□□ □□□□ □□□□□□ معلومات وعرض التطبيق، صحة وفحص مختلفة، مقاييس لجمع استخدامه يمكنك تطبيقاتك. وإدارة لمراقبة pom.xml ملف إلى المناسبة التبعيات إضافة طريق عن الميزات هذه تفعيل يتم الكثير. ذلك وغير التكوين، build.gradle أو، مثل تكوينها حسب الحاجة.

إضافة يمكنك، □□□□□ باستخدام □□□□ □□□□□□ مشروع إلى □□□□□□□□□ لإضافة المثال، سبيل على pom.xml ملف إلى التالية التبعية:

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

المثال، سبيل على. □□□□ عبر □□□□□□□□ يوفرها التي المختلفة النهاية نقاط إلى الوصول يمكنك ذلك، وبعد /actuator/health. إلى □□□ طلب طريق عن الصحة معلومات إلى الوصول يمكنك

```
curl http://localhost:8080/actuator/health
```

التطبيق. صحة حالة على تحتوي □□□□ استجابة إرجاع سيتم

```json
{
    "status": "UP"
}
```

أو application.properties ملف عبر وتعطيلها تفعيلها تريد التي النهاية نقاط تكوين أيضًا يمكنك application.yml. المثال: سبيل على

```
management.endpoints.web.exposure.include=health,info
management.endpoint.health.show-details=always
```

وسيعرض بالظهور، /actuator/info و /actuator/health النهاية بنقاط فقط سيسمح التكوين هذا دائمًا. الصحة تفاصيل

ومراقبة إدارة في تساعدك أن يمكن التي الميزات من واسعة مجموعة يوفر □□□□□□□□□ □□□□ □□□□□□ بفعالية. تطبيقاتك

تطبيقك. وإدارة لمراقبة للإنتاج جاهزة ميزات □□□□□□□□□ □□□□ □□□□□□ يوفر

- /actuator/metrics و /actuator/health مثل طرفية نقاطًا استخدم :□□□□□□□□□□□ الطرفية النقاط المثال: سبيل على والمقاييس. التطبيق صحة لمراقبة

  ```
  curl http://localhost:8080/actuator/health
  ```

- المثال: سبيل على. □□□□□□□□□ لل- مخصصة نهاية نقاط بإنشاء قم المخصصة: النهاية نقاط

  ```java
  @RestController
  @RequestMapping("/actuator")
  public class CustomEndpoint {
      @GetMapping("/custom")
      public Map<String, String> customEndpoint() {
          Map<String, String> response = new HashMap<>();
          response.put("status", "           actuator");
          return response;
      }
  }
  ```

متقدمة مواضيع

□□□□□□□□□□□□□□□□ □□□ □□□□□□ عمل إطار من جزء هي □□□ □□□□□□ □□□□□□ □□□ □□□□□□ □□□□□□ هو □□□□□□ الـ. □□□□□□ تطبيقات عبر المشتركة السلوكيات لتطبيق يُستخدم الذي □□□□□□□□□□□□ معينة. طريقة استدعاء بعد أو قبل مثل البرنامج، تنفيذ في معينة نقطة عند تنفيذه يتم الذي الإجراء

هناك عدة أنواع من الـ □□□□□□ في □□□□□□:

1. □□□□□□ □□□□□□: معينة. طريقة استدعاء قبل تنفيذه يتم
2. □□□□□ □□□□□□□□□ □□□□□□: استثناءات□. بدون بنجاح معينة طريقة استدعاء بعد تنفيذه يتم
3. □□□□□ □□□□□□□□ □□□□□□: معينة. طريقة استدعاء أثناء استثناء رمي تم إذا تنفيذه يتم
4. □□□□□ □□□□□□□□□ □□□□□□: النتيجة عن النظر بغض معينة طريقة استدعاء بعد تنفيذه يتم سواء□ تم بنجاح أو مع استثناء□.
5. □□□□□□ □□□□□□: الطريقة تنفيذ في بالتحكم يسمح مما معينة، طريقة استدعاء حول تنفيذه يتم كامل. بشكل

مثال على استخدام □□□□□□ □□□□□□

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LoggingAspect {

    @Before("execution(* com.example.service.*.*(..))")
    public void logBefore() {
        System.out.println("Method is about to be called.");
    }
}
```

في هذا المثال، يتم تنفيذ الـ logBefore قبل استدعاء أي طريقة في الحزمة com.example.service.

مثال على استخدام □□□□□□ □□□□□□

```
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
```

```java
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class TimingAspect {

    @Around("execution(* com.example.service.*.*(..))")
    public Object measureExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {
        long startTime = System.currentTimeMillis();
        Object result = joinPoint.proceed();
        long endTime = System.currentTimeMillis();
        System.out.println("Method execution time: " + (endTime - startTime) + "ms");
        return result;
    }
}
```

في هذا المثال، يتم قياس وقت تنفيذ أي طريقة في الحزمة com.example.service لمستغرق. الوقت وإخراج

أكثر الكود يجعل مما التطبيق، عبر المشتركة السلوكيات إدارة في كبيرة مرونة يوفر [illegible] [illegible] [illegible]

الصيانة. في وسهولة نظافة

الجوانب حن والموجهة للمرجمة متقدمة تُوفر [illegible] إمكانيات [illegible] بـ الخاصة [illegible] التطبيقات برمجة واجهة [illegible].

 @Aspect: قم فيرعتب بانجوال امدختساب @Aspect. على يلي سبب المثال:

```java
@Aspect
@Component
public class LoggingAspect {
    @Before("execution(* com.example.service.*.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        System.out.println(  "     :     " + joinPoint.getSignature().getName());
    }
}

@After("execution(* com.example.service.*.*(..))")
public void logAfter(JoinPoint joinPoint) {
```

```java
    System.out.println(  "      :      " + joinPoint.getSignature().getName());
}
```

- نقاط الالتحاق □□□□□ □□□□□□□: استخدم نقاط الالتحاق لتحديد الأماكن التي يجب أن ثطبق فيها الجوانب □□□□□□□□□. على سبيل المثال:

```java
@Pointcut("execution(* com.example.service.*.*(..))")
public void serviceMethods() {}

@Around("serviceMethods()")
public Object logAround(ProceedingJoinPoint joinPoint) throws Throwable {
    System.out.println(  "      :      " + joinPoint.getSignature().getName());
    Object result = joinPoint.proceed();
    System.out.println(  "      :      " + joinPoint.getSignature().getName());
    return result;
}
```

الخلاصة

□□□□□□ هو إطار عمل قوي ومتعدد الاستخدامات يمكنه تبسيط تطوير التطبيقات على مستوى المؤسسات. من خلال الاستفادة من ميزات □□□□□□ □□□□ □□□□□□ و□□□□□□ □□□□ و□□□□ وغيرها من مشاريع □□□□□□، يمكن للمطورين بناء تطبيقات قوية وقابلة للتوسع وسهلة الصيانة بكفاءة. مع إضافة أدوات مثل □□□□□□ □□□□ □□□□□□□□ وأطر اختبار، يمكنك التأكد من أن تطبيقاتك جاهزة للإنتاج ومختَبرة بشكل جيد.