# 关于阻止xss攻击的方法及其实现

**班级：19外包2班 姓名： 钟学炫 学号：1920502254**

摘要：XSS漏洞可以分为三种类型：反射型、持久型和基于DOM的XSS漏洞。该文主要研究这三种不同类型的内容、探讨阻止xss攻击的方法以及实现这些方法。

关键词：xss攻击；xss防范；web站点；xss漏洞；xss拦截；

**Absrtact**: XSS vulnerabilities can be divided into three types: reflective, persistent and DOM based XSS vulnerabilities. This paper mainly studies these three different types of content, discusses the methods to prevent XSS attacks and the implementation of these methods.

**Keywords**: XSS attack; XSS prevention; Web site; XSS vulnerability; XSS interception;

## 引言

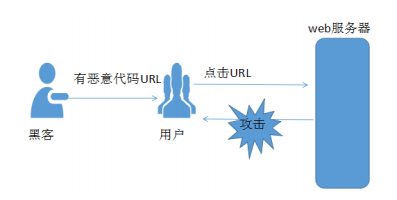
随着计算机科学与技术的飞速发展，web技术已经越来越盛行。现今，web应用已经遍布在我们生活的各个方面，例如：小程序，电商，健康码，场所码，支付等。然而，web安全问题不断浮现，这些应用也常被黑客攻击，因此，web应用的安全不容忽视。Xss目前是比较流行的web应用漏洞，则应当对XSS攻击进行防范。

Xss攻击通常说的是黑客通过利用网页在开发时留下的某些漏洞，对其注入一些恶意的脚本，从而改变网页的状态。当客户端对该网页进行浏览时，浏览器会对该html进行渲染，并且会执行一些隐藏的恶意脚本，从而被黑客控制整个客户端的浏览器。XSS是目前web站点里面普遍存在的漏洞，利用静态的文本或者动态的文本的方式进行恶意代码的编写，从而达到载入到客户端的计算机的目的。

## xss漏洞的相关分类

* 1. **反射型xss**

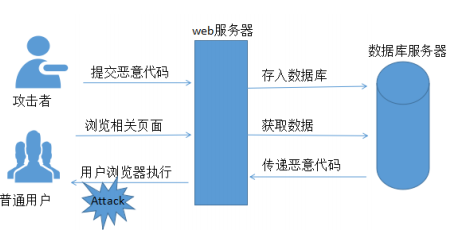
反射型 xss 漏洞是客户端对服务器提交数据，当服务器对数据处理和发生响应后，若在此过程中，没有对数据进行有效地过滤，那么这段 xss 代码将会出现在浏览器中，从而对用户的电脑进行攻击。



**图 1反射型xss攻击原理**

* 1. **存储型xss**

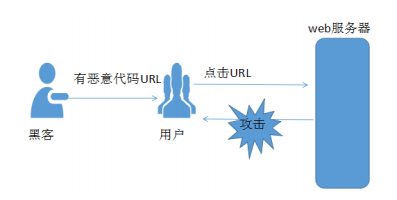
存储型 xss 漏洞是服务器端没有对客户端提交的数据没做有效的过滤和处理，并因此将数据存入服务器的数据库，当客户端浏览网页时，服务器从其数据库中加载数据并在web网页中显示，从而形成存储型XSS 漏洞。



**图 2存储型xss攻击原理**

* 1. **DOM型xss**

DOM型xss漏洞是纯前端的问题，是由于前端没有对客户端提交的数据进行相关有效的过滤和处理，从而让其在网页上运行。



**图 3 DOM型xss攻击原理**

## xss攻击的复现

操作过程：首先建立一个web站点，该web站点的前端、后端都不会对任何客户端的字符进行某些限制，转义，过滤等。这里采用的技术是：前端Jsp，后端SpringMvc、Mybatis、数据库MySql以及运行容器Apache Tomcat，使用的测试浏览器为360极速浏览器，采用的总体架构：MVC架构。

* 1. **DOM型xss**

DOM型xss是不经过后端和数据库的，直接在前端就会出现的一种xss攻击。

在Web中写一个jsp文件，该文件用来模拟实现用户端数据的输入。实现的主要代码如下：

<div id="xss">  
 内容：<input id="text" name="text" type="text"><br>  
 <button onclick="*domxss*()">提交</button>  
</div>  
<div id="dom"></div>  
<script>  
 function *domxss*(){  
 var str=document.getElementById("text").value;  
 document.getElementById("dom").innerHTML = "<a href='"+str+"'>xss</a>";  
 }  
</script>



**图 4 DOM型xss攻击-演示**



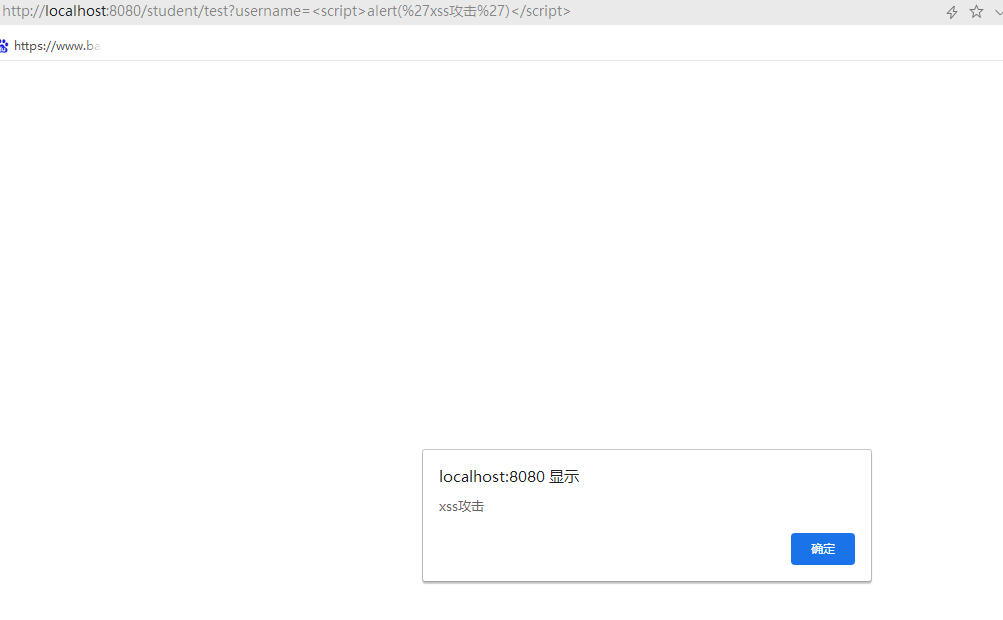
**图 5 DOM型xss攻击-代码查看**

* 1. **反射型xss**

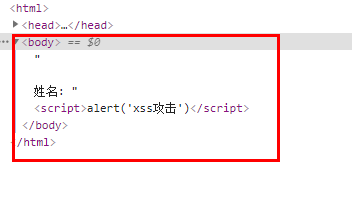
反射型攻击是一种经过后端数据处理的，而不存入数据库的一种攻击。

在Web中写一个jsp文件，该文件用来模拟实现用户端url数据的输入。实现的主要代码如下：

<%String name= request.getParameter("username"); %>  
姓名: <%= name%>



**图 6反射型xss攻击-演示**



**图 7反射型xss攻击-代码查看**

* 1. **存储型xss**

存储型xss是一种前端经过后端处理后传入数据库进行存储的一种攻击。

1. 前端的代码实现：

在Web中写一个jsp文件，该文件用来模拟实现用户端数据的输入。实现的主要代码如下：

<form action="${pageContext.request.contextPath}/student/insert">  
 班级编号:<input type="text" name="classId"><br>  
 学生姓名:<input type="text" name="name"><br>  
 性别:<input type="text" name="sex"><br>  
 生日:<input type="text" name="birthday"><br>  
 <input type="submit" value="提交">  
</form>

除此之外，再写入一个读取后端传过来的数据，对其向客户端进行展示，代码如下：

<table>  
 <tr>  
 <th>学生编号</th>  
 <th>班级编号</th>  
 <th>学生姓名</th>  
 <th>学生性别</th>  
 <th>学生生日</th>  
 </tr>  
 <c:forEach items="${studentList}" var="student">  
 <tr>  
 <td>${student.stuId}</td>  
 <td>${student.classId}</td>  
 <td>${student.name}</td>  
 <td>${student.sex}</td>  
 <td>${student.birthday}</td>  
 </tr>  
 </c:forEach>  
</table>

1. 后端的代码实现：

①控制层（Controller）主要是对前端的请求结合模型以及业务层进行处理，然后用视图响应给客户端，其所实现的主要代码如下：

@Controller  
@RequestMapping("/student")  
public class StudentController {  
 @Autowired  
 private StudentService studentService;  
 @RequestMapping("/findAll")  
 public ModelAndView findAll(){  
 List<Studentcopy> studentList=studentService.findAll();  
 ModelAndView mv= new ModelAndView();  
 mv.addObject("studentList",studentList);  
 mv.setViewName("test1");  
 return mv;  
 }  
 @ResponseBody  
 @RequestMapping(value="/insert",produces = "text/html;charset=UTF-8")  
 public String insert(Studentcopy studentcopy){  
 return studentService.insert(studentcopy)>0?"插入成功":"插入失败";  
 }  
}

②服务层（Service）主要是数据库的相关操作，进行业务上的逻辑整合，从而返回所需的数据给控制层，以便展示给客户端，其所实现的主要代码如下：

@Service("studentService")  
public class StudentServiceImpl implements StudentService {  
 @Autowired  
 private StudentcopyMapper studentcopyMapper;  
 @Override  
 public List<Studentcopy> findAll() {  
 return studentcopyMapper.selectByExample(new StudentcopyExample());  
 }  
  
 @Override  
 public int insert(Studentcopy studentcopy) {  
 return studentcopyMapper.insert(studentcopy);  
 }  
}

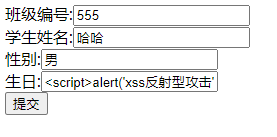
③数据访问对象（DAO）层主要是对数据库的底层的相关操作，这些执行具有一定的原子性，不组成任何业务上的逻辑，返回所需的数据给业务层，以便展示给客户端，这里使用mybatis技术，用xml配置方式实现dao层。其所实现的主要代码如下：

<mapper namespace="org.example.daoback.StudentcopyMapper" >  
 <resultMap id="BaseResultMap" type="org.example.beanback.Studentcopy" >  
 <id column="stu\_id" property="stuId" jdbcType="INTEGER" />  
 <result column="class\_id" property="classId" jdbcType="INTEGER" />  
 <result column="name" property="name" jdbcType="VARCHAR" />  
 <result column="birthday" property="birthday" jdbcType="VARCHAR" />  
 <result column="sex" property="sex" jdbcType="VARCHAR" />  
 </resultMap>  
 <sql id="Example\_Where\_Clause" >  
 <where >  
 <foreach collection="oredCriteria" item="criteria" separator="or" >  
 <if test="criteria.valid" >  
 <trim prefix="(" suffix=")" prefixOverrides="and" >  
 <foreach collection="criteria.criteria" item="criterion" >  
 <choose >  
 <when test="criterion.noValue" >  
 and ${criterion.condition}  
 </when>  
 <when test="criterion.singleValue" >  
 and ${criterion.condition} #{criterion.value}  
 </when>  
 <when test="criterion.betweenValue" >  
 and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}  
 </when>  
 <when test="criterion.listValue" >  
 and ${criterion.condition}  
 <foreach collection="criterion.value" item="listItem" open="(" close=")" separator="," >  
 #{listItem}  
 </foreach>  
 </when>  
 </choose>  
 </foreach>  
 </trim>  
 </if>  
 </foreach>  
 </where>  
 </sql>  
 <sql id="Update\_By\_Example\_Where\_Clause" >  
 <where >  
 <foreach collection="example.oredCriteria" item="criteria" separator="or" >  
 <if test="criteria.valid" >  
 <trim prefix="(" suffix=")" prefixOverrides="and" >  
 <foreach collection="criteria.criteria" item="criterion" >  
 <choose >  
 <when test="criterion.noValue" >  
 and ${criterion.condition}  
 </when>  
 <when test="criterion.singleValue" >  
 and ${criterion.condition} #{criterion.value}  
 </when>  
 <when test="criterion.betweenValue" >  
 and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}  
 </when>  
 <when test="criterion.listValue" >  
 and ${criterion.condition}  
 <foreach collection="criterion.value" item="listItem" open="(" close=")" separator="," >  
 #{listItem}  
 </foreach>  
 </when>  
 </choose>  
 </foreach>  
 </trim>  
 </if>  
 </foreach>  
 </where>  
 </sql>  
 <sql id="Base\_Column\_List" >  
 stu\_id, class\_id, name, birthday, sex  
 </sql>  
 <select id="selectByExample" resultMap="BaseResultMap" parameterType="org.example.beanback.StudentcopyExample" >  
 select  
 <if test="distinct" >  
 distinct  
 </if>  
 <include refid="Base\_Column\_List" />  
 from studentcopy  
 <if test="\_parameter != null" >  
 <include refid="Example\_Where\_Clause" />  
 </if>  
 <if test="orderByClause != null" >  
 order by ${orderByClause}  
 </if>  
 </select>  
 <select id="selectByPrimaryKey" resultMap="BaseResultMap" parameterType="java.lang.Integer" >  
 select   
 <include refid="Base\_Column\_List" />  
 from studentcopy  
 where stu\_id = #{stuId,jdbcType=INTEGER}  
 </select>  
 <delete id="deleteByPrimaryKey" parameterType="java.lang.Integer" >  
 delete from studentcopy  
 where stu\_id = #{stuId,jdbcType=INTEGER}  
 </delete>  
 <delete id="deleteByExample" parameterType="org.example.beanback.StudentcopyExample" >  
 delete from studentcopy  
 <if test="\_parameter != null" >  
 <include refid="Example\_Where\_Clause" />  
 </if>  
 </delete>  
 <insert id="insert" parameterType="org.example.beanback.Studentcopy" >  
 insert into studentcopy (stu\_id, class\_id, name,   
 birthday, sex)  
 values (#{stuId,jdbcType=INTEGER}, #{classId,jdbcType=INTEGER}, #{name,jdbcType=VARCHAR},   
 #{birthday,jdbcType=VARCHAR}, #{sex,jdbcType=VARCHAR})  
 </insert>  
 <insert id="insertSelective" parameterType="org.example.beanback.Studentcopy" >  
 insert into studentcopy  
 <trim prefix="(" suffix=")" suffixOverrides="," >  
 <if test="stuId != null" >  
 stu\_id,  
 </if>  
 <if test="classId != null" >  
 class\_id,  
 </if>  
 <if test="name != null" >  
 name,  
 </if>  
 <if test="birthday != null" >  
 birthday,  
 </if>  
 <if test="sex != null" >  
 sex,  
 </if>  
 </trim>  
 <trim prefix="values (" suffix=")" suffixOverrides="," >  
 <if test="stuId != null" >  
 #{stuId,jdbcType=INTEGER},  
 </if>  
 <if test="classId != null" >  
 #{classId,jdbcType=INTEGER},  
 </if>  
 <if test="name != null" >  
 #{name,jdbcType=VARCHAR},  
 </if>  
 <if test="birthday != null" >  
 #{birthday,jdbcType=VARCHAR},  
 </if>  
 <if test="sex != null" >  
 #{sex,jdbcType=VARCHAR},  
 </if>  
 </trim>  
 </insert>  
 <select id="countByExample" parameterType="org.example.beanback.StudentcopyExample" resultType="java.lang.Integer" >  
 select count(\*) from studentcopy  
 <if test="\_parameter != null" >  
 <include refid="Example\_Where\_Clause" />  
 </if>  
 </select>  
 <update id="updateByExampleSelective" parameterType="map" >  
 update studentcopy  
 <set >  
 <if test="record.stuId != null" >  
 stu\_id = #{record.stuId,jdbcType=INTEGER},  
 </if>  
 <if test="record.classId != null" >  
 class\_id = #{record.classId,jdbcType=INTEGER},  
 </if>  
 <if test="record.name != null" >  
 name = #{record.name,jdbcType=VARCHAR},  
 </if>  
 <if test="record.birthday != null" >  
 birthday = #{record.birthday,jdbcType=VARCHAR},  
 </if>  
 <if test="record.sex != null" >  
 sex = #{record.sex,jdbcType=VARCHAR},  
 </if>  
 </set>  
 <if test="\_parameter != null" >  
 <include refid="Update\_By\_Example\_Where\_Clause" />  
 </if>  
 </update>  
 <update id="updateByExample" parameterType="map" >  
 update studentcopy  
 set stu\_id = #{record.stuId,jdbcType=INTEGER},  
 class\_id = #{record.classId,jdbcType=INTEGER},  
 name = #{record.name,jdbcType=VARCHAR},  
 birthday = #{record.birthday,jdbcType=VARCHAR},  
 sex = #{record.sex,jdbcType=VARCHAR}  
 <if test="\_parameter != null" >  
 <include refid="Update\_By\_Example\_Where\_Clause" />  
 </if>  
 </update>  
 <update id="updateByPrimaryKeySelective" parameterType="org.example.beanback.Studentcopy" >  
 update studentcopy  
 <set >  
 <if test="classId != null" >  
 class\_id = #{classId,jdbcType=INTEGER},  
 </if>  
 <if test="name != null" >  
 name = #{name,jdbcType=VARCHAR},  
 </if>  
 <if test="birthday != null" >  
 birthday = #{birthday,jdbcType=VARCHAR},  
 </if>  
 <if test="sex != null" >  
 sex = #{sex,jdbcType=VARCHAR},  
 </if>  
 </set>  
 where stu\_id = #{stuId,jdbcType=INTEGER}  
 </update>  
 <update id="updateByPrimaryKey" parameterType="org.example.beanback.Studentcopy" >  
 update studentcopy  
 set class\_id = #{classId,jdbcType=INTEGER},  
 name = #{name,jdbcType=VARCHAR},  
 birthday = #{birthday,jdbcType=VARCHAR},  
 sex = #{sex,jdbcType=VARCHAR}  
 where stu\_id = #{stuId,jdbcType=INTEGER}  
 </update>

④模型（BEAN）层主要是传送时数据的封装。其所实现的主要代码如下：

public class Studentcopy {  
 private Integer stuId;  
 private Integer classId;  
 private String name;  
 private String birthday;  
 private String sex;  
 public Integer getStuId() {  
 return stuId;  
 }  
 public void setStuId(Integer stuId) {  
 this.stuId = stuId;  
 }  
 public Integer getClassId() {  
 return classId;  
 }  
 public void setClassId(Integer classId) {  
 this.classId = classId;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name == null ? null : name.trim();  
 }  
 public String getBirthday() {  
 return birthday;  
 }  
 public void setBirthday(String birthday) {  
 this.birthday = birthday == null ? null : birthday.trim();  
 }  
 public String getSex() {  
 return sex;  
 }  
 public void setSex(String sex) {  
 this.sex = sex == null ? null : sex.trim();  
 }  
}

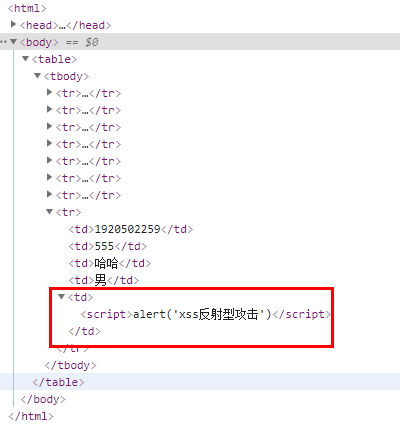
public class StudentcopyExample {  
 protected String orderByClause;  
 protected boolean distinct;  
 protected List<Criteria> oredCriteria;  
 public StudentcopyExample() {  
 oredCriteria = new ArrayList<Criteria>();  
 }  
 public void setOrderByClause(String orderByClause) {  
 this.orderByClause = orderByClause;  
 }  
 public String getOrderByClause() {  
 return orderByClause;  
 }  
 public void setDistinct(boolean distinct) {  
 this.distinct = distinct;  
 }  
 public boolean isDistinct() {  
 return distinct;  
 }  
 public List<Criteria> getOredCriteria() {  
 return oredCriteria;  
 }  
 public void or(Criteria criteria) {  
 oredCriteria.add(criteria);  
 }  
 public Criteria or() {  
 Criteria criteria = createCriteriaInternal();  
 oredCriteria.add(criteria);  
 return criteria;  
 }  
 public Criteria createCriteria() {  
 Criteria criteria = createCriteriaInternal();  
 if (oredCriteria.size() == 0) {  
 oredCriteria.add(criteria);  
 }  
 return criteria;  
 }  
 protected Criteria createCriteriaInternal() {  
 Criteria criteria = new Criteria();  
 return criteria;  
 }  
 public void clear() {  
 oredCriteria.clear();  
 orderByClause = null;  
 distinct = false;  
 }  
 protected abstract static class GeneratedCriteria {  
 protected List<Criterion> criteria;  
  
 protected GeneratedCriteria() {  
 super();  
 criteria = new ArrayList<Criterion>();  
 }  
 public boolean isValid() {  
 return criteria.size() > 0;  
 }  
 public List<Criterion> getAllCriteria() {  
 return criteria;  
 }  
 public List<Criterion> getCriteria() {  
 return criteria;  
 }  
 protected void addCriterion(String condition) {  
 if (condition == null) {  
 throw new RuntimeException("Value for condition cannot be null");  
 }  
 criteria.add(new Criterion(condition));  
 }  
 protected void addCriterion(String condition, Object value, String property) {  
 if (value == null) {  
 throw new RuntimeException("Value for " + property + " cannot be null");  
 }  
 criteria.add(new Criterion(condition, value));  
 }  
 protected void addCriterion(String condition, Object value1, Object value2, String property) {  
 if (value1 == null || value2 == null) {  
 throw new RuntimeException("Between values for " + property + " cannot be null");  
 }  
 criteria.add(new Criterion(condition, value1, value2));  
 }  
 public Criteria andStuIdIsNull() {  
 addCriterion("stu\_id is null");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdIsNotNull() {  
 addCriterion("stu\_id is not null");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdEqualTo(Integer value) {  
 addCriterion("stu\_id =", value, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdNotEqualTo(Integer value) {  
 addCriterion("stu\_id <>", value, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdGreaterThan(Integer value) {  
 addCriterion("stu\_id >", value, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdGreaterThanOrEqualTo(Integer value) {  
 addCriterion("stu\_id >=", value, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdLessThan(Integer value) {  
 addCriterion("stu\_id <", value, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdLessThanOrEqualTo(Integer value) {  
 addCriterion("stu\_id <=", value, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdIn(List<Integer> values) {  
 addCriterion("stu\_id in", values, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdNotIn(List<Integer> values) {  
 addCriterion("stu\_id not in", values, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdBetween(Integer value1, Integer value2) {  
 addCriterion("stu\_id between", value1, value2, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andStuIdNotBetween(Integer value1, Integer value2) {  
 addCriterion("stu\_id not between", value1, value2, "stuId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdIsNull() {  
 addCriterion("class\_id is null");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdIsNotNull() {  
 addCriterion("class\_id is not null");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdEqualTo(Integer value) {  
 addCriterion("class\_id =", value, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdNotEqualTo(Integer value) {  
 addCriterion("class\_id <>", value, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdGreaterThan(Integer value) {  
 addCriterion("class\_id >", value, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdGreaterThanOrEqualTo(Integer value) {  
 addCriterion("class\_id >=", value, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdLessThan(Integer value) {  
 addCriterion("class\_id <", value, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdLessThanOrEqualTo(Integer value) {  
 addCriterion("class\_id <=", value, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdIn(List<Integer> values) {  
 addCriterion("class\_id in", values, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdNotIn(List<Integer> values) {  
 addCriterion("class\_id not in", values, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdBetween(Integer value1, Integer value2) {  
 addCriterion("class\_id between", value1, value2, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andClassIdNotBetween(Integer value1, Integer value2) {  
 addCriterion("class\_id not between", value1, value2, "classId");  
 return (Criteria) this;  
 }  
 public Criteria andNameIsNull() {  
 addCriterion("name is null");  
 return (Criteria) this;  
 }  
 public Criteria andNameIsNotNull() {  
 addCriterion("name is not null");  
 return (Criteria) this;  
 }  
 public Criteria andNameEqualTo(String value) {  
 addCriterion("name =", value, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameNotEqualTo(String value) {  
 addCriterion("name <>", value, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameGreaterThan(String value) {  
 addCriterion("name >", value, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameGreaterThanOrEqualTo(String value) {  
 addCriterion("name >=", value, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameLessThan(String value) {  
 addCriterion("name <", value, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameLessThanOrEqualTo(String value) {  
 addCriterion("name <=", value, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameLike(String value) {  
 addCriterion("name like", value, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameNotLike(String value) {  
 addCriterion("name not like", value, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameIn(List<String> values) {  
 addCriterion("name in", values, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameNotIn(List<String> values) {  
 addCriterion("name not in", values, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameBetween(String value1, String value2) {  
 addCriterion("name between", value1, value2, "name");  
 return (Criteria) this;  
 }  
 public Criteria andNameNotBetween(String value1, String value2) {  
 addCriterion("name not between", value1, value2, "name");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayIsNull() {  
 addCriterion("birthday is null");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayIsNotNull() {  
 addCriterion("birthday is not null");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayEqualTo(String value) {  
 addCriterion("birthday =", value, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayNotEqualTo(String value) {  
 addCriterion("birthday <>", value, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayGreaterThan(String value) {  
 addCriterion("birthday >", value, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayGreaterThanOrEqualTo(String value) {  
 addCriterion("birthday >=", value, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayLessThan(String value) {  
 addCriterion("birthday <", value, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayLessThanOrEqualTo(String value) {  
 addCriterion("birthday <=", value, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayLike(String value) {  
 addCriterion("birthday like", value, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayNotLike(String value) {  
 addCriterion("birthday not like", value, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayIn(List<String> values) {  
 addCriterion("birthday in", values, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayNotIn(List<String> values) {  
 addCriterion("birthday not in", values, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayBetween(String value1, String value2) {  
 addCriterion("birthday between", value1, value2, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andBirthdayNotBetween(String value1, String value2) {  
 addCriterion("birthday not between", value1, value2, "birthday");  
 return (Criteria) this;  
 }  
 public Criteria andSexIsNull() {  
 addCriterion("sex is null");  
 return (Criteria) this;  
 }  
 public Criteria andSexIsNotNull() {  
 addCriterion("sex is not null");  
 return (Criteria) this;  
 }  
 public Criteria andSexEqualTo(String value) {  
 addCriterion("sex =", value, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexNotEqualTo(String value) {  
 addCriterion("sex <>", value, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexGreaterThan(String value) {  
 addCriterion("sex >", value, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexGreaterThanOrEqualTo(String value) {  
 addCriterion("sex >=", value, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexLessThan(String value) {  
 addCriterion("sex <", value, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexLessThanOrEqualTo(String value) {  
 addCriterion("sex <=", value, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexLike(String value) {  
 addCriterion("sex like", value, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexNotLike(String value) {  
 addCriterion("sex not like", value, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexIn(List<String> values) {  
 addCriterion("sex in", values, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexNotIn(List<String> values) {  
 addCriterion("sex not in", values, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexBetween(String value1, String value2) {  
 addCriterion("sex between", value1, value2, "sex");  
 return (Criteria) this;  
 }  
 public Criteria andSexNotBetween(String value1, String value2) {  
 addCriterion("sex not between", value1, value2, "sex");  
 return (Criteria) this;  
 }  
 }  
 public static class Criteria extends GeneratedCriteria {  
 protected Criteria() {  
 super();  
 }  
 }  
 public static class Criterion {  
 private String condition;  
 private Object value;  
 private Object secondValue;  
 private boolean noValue;  
 private boolean singleValue;  
 private boolean betweenValue;  
 private boolean listValue;  
 private String typeHandler;  
 public String getCondition() {  
 return condition;  
 }  
 public Object getValue() {  
 return value;  
 }  
 public Object getSecondValue() {  
 return secondValue;  
 }  
 public boolean isNoValue() {  
 return noValue;  
 }  
 public boolean isSingleValue() {  
 return singleValue;  
 }  
 public boolean isBetweenValue() {  
 return betweenValue;  
 }  
 public boolean isListValue() {  
 return listValue;  
 }  
 public String getTypeHandler() {  
 return typeHandler;  
 }  
 protected Criterion(String condition) {  
 super();  
 this.condition = condition;  
 this.typeHandler = null;  
 this.noValue = true;  
 }  
 protected Criterion(String condition, Object value, String typeHandler) {  
 super();  
 this.condition = condition;  
 this.value = value;  
 this.typeHandler = typeHandler;  
 if (value instanceof List<?>) {  
 this.listValue = true;  
 } else {  
 this.singleValue = true;  
 }  
 }  
 protected Criterion(String condition, Object value) {  
 this(condition, value, null);  
 }  
 protected Criterion(String condition, Object value, Object secondValue, String typeHandler) {  
 super();  
 this.condition = condition;  
 this.value = value;  
 this.secondValue = secondValue;  
 this.typeHandler = typeHandler;  
 this.betweenValue = true;  
 }  
 protected Criterion(String condition, Object value, Object secondValue) {  
 this(condition, value, secondValue, null);  
 }  
 }  
}



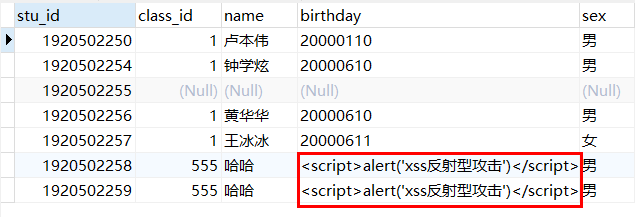
**图 8 存储型xss攻击-演示**



**图 9 存储型xss攻击-演示**



**图 10 存储型xss攻击-代码查看**



**图 11 存储型xss攻击-数据库查看**

## xss攻击防范手段

* 1. **对字符的转义**

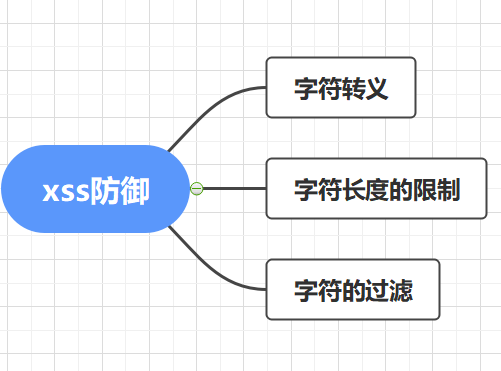
字符的转义是将一些敏感字符，例如< 、> 、; 、\ 、/ 等，用其他标识符进行替代，从而阻止黑客对网页提交一些恶意的JavaScript脚本等。

* 1. **对字符的长度的限制**

字符的长度限制也是能在一定程度上阻止黑客对网站进行的xss攻击，因为，往往黑客所提交的一些恶意的JavaScript脚本都是长度过于长，超过了正常客户端所提交的正常文本，所以能在一定程度上对客户端进行xss攻击的防御。

* 1. **对字符的过滤**

字符的过滤是对客户端输入的一些文本进行过滤，例如，利用正则表达式、利用if判断等对客户端的输入文本进行必要的过滤，从而能够有效阻止一些非法文本的输入，阻止恶意的JavaScript脚本的注入。



**图 12 xss防御手段**

## Xss主要防御手段的实现

* 1. **前端利用正则表达式对字符的过滤**

前端页面可以利用正则表达式，通过给定的规则，过滤替代掉一些特殊字符，从而不会使黑客或者攻击者利用xss攻击传入一些恶意的JavaScript的代码，使得该站点的使用者受到攻击。

这里在前端页面加入如下的正则过滤表达式：

function *domxss*() {  
 var s=document.getElementById("text").value;  
 var pattern = new RegExp("[`~!@#$^&\*()=|{}':;',\\[\\].<>~！@#￥……&\*（）——|{}【】‘；：”“'。，、？]");  
 var rs = "";  
 for (var i = 0; i < s.length; i++) {  
 rs = rs + s.substr(i, 1).replace(pattern, '');  
 }  
 document.getElementById("dom").innerHTML = "<a href='" + rs + "'>xss</a>";  
}

通过对该正则表达式的测试，可以看到这对字符过滤是有效的，它把测试的时候，所填入的xss攻击代码中的特殊字符进行了替代，从而无法让其生效，如下图所示：



**图 13 正则表达式过滤测试**

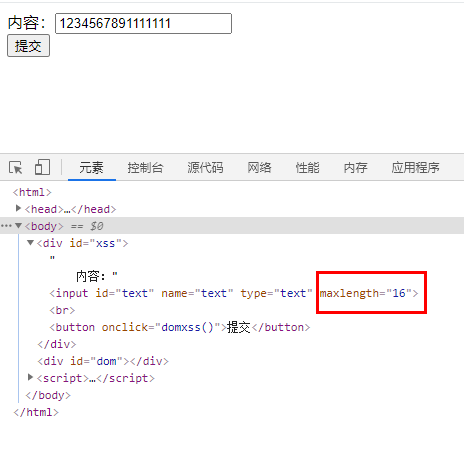
* 1. **前端对字符长度的限制**

前端页面对使用者所输入的字符进行长度限制，从而可以在一定程度上避免一些JavaScript的恶意代码的注入。

这里在前端将输入框限制为不大于16个字符，修改的代码如下：

<input id="text" name="text" type="text" maxlength="16"><br>

通过对该字符长度限制的测试，可以看到这对字符限制是有效的，如下图所示：



**图 14 字符长度测试**

* 1. **前端对字符的转义**

前端页面对使用者所输入的字符进行字符的转义，从而可以避免黑客或攻击者注入一些对页面有危害的JavaScript代码，使得页面使用者受到黑客的危害。

这里在前端将函数内所获得的dom元素的内容进行转义，修改的代码如下

function *domxss*() {  
 let str = document.getElementById("text").value;  
 str = *encodeURIComponent*(str);  
 document.getElementById("dom").innerHTML = "<a href='" + str + "'>xss</a>";  
}

通过对该字符的转义的测试，可以看到字符转义对xss攻击是十分有效的，它直接把代码中的特殊字符转为替代的字符串，从而让浏览器识别为普通的字符串，而不是JavaScript代码，如下图所示：



**图 15 字符转义测试**

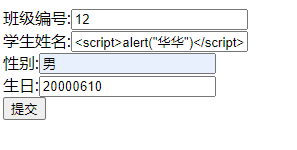
* 1. **后端利用正则表达式对字符的过滤**

后端开发者可以利用正则表达式，通过给定的规则，过滤替代掉一些特殊字符，从而不会使黑客或者攻击者利用xss攻击传入一些恶意的JavaScript的代码，使得该站点的使用者受到攻击。它与前端过滤不同，前端过滤只是暂时的缓解那些对技术较弱的攻击者，对于技术比较高超的攻击者或黑客，它可以通过修改前端的代码或者通过一些修改所抓到的包的软件，从而绕过前端限制，对站点实施xss攻击。

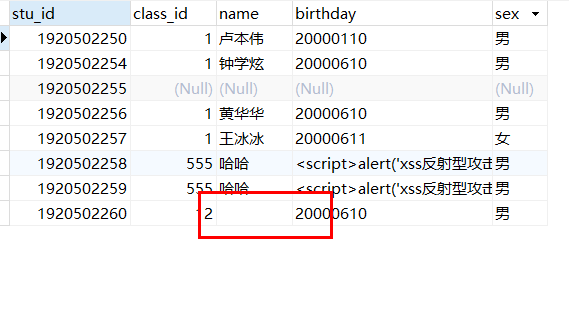
这里在后端代码中加入如下的正则过滤表达式：

@ResponseBody  
@RequestMapping(value="/insert",produces = "text/html;charset=UTF-8")  
public String insert(Studentcopy studentcopy){  
 String regEx\_script = "<script[^>]\*?>[\\s\\S]\*?<\\/script>";  
 String name=studentcopy.getName();  
 Pattern p\_script = Pattern.*compile*(regEx\_script, Pattern.*CASE\_INSENSITIVE*);  
 Matcher m\_script = p\_script.matcher(name);  
 name = m\_script.replaceAll("");  
 studentcopy.setName(name);  
 return studentService.insert(studentcopy)>0?"插入成功":"插入失败";  
}

通过对该正则表达式的测试，可以看到这对字符过滤是有效的，它把测试的时候，所填入的xss攻击代码中的特殊字符进行了替代，从而无法让其生效，数据库中也是看不到过滤的内容，如下图所示：



**图 16 字符转义-测试**



**图 17 字符转义-数据库查看**

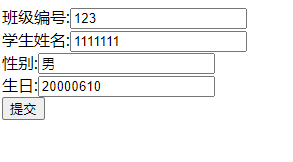
* 1. **后端对字符长度的限制**

后端页面对使用者所输入的字符进行长度限制，从而可以在一定程度上避免一些JavaScript的恶意代码的注入。

这里在后端将输入框限制为不大于3个字符，修改的代码如下：

if(name.length()>4){  
 return "名字长度不能超过3";  
}

通过对该字符长度限制的测试，可以看到这对字符限制是有效的，如下图所示：



**图 18 字符长度限制-测试**



**图 19 字符长度限制-测试结果**

* 1. **后端对字符的转义**

后端对前端使用者所传入的字符进行字符的转义，从而可以避免黑客或攻击者注入一些对页面有危害的JavaScript代码，使得页面使用者受到黑客的危害。

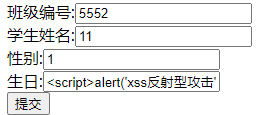
这里在后端加入一个基础的Controller，该Controller对字符进行统一的转义，从而让其他继承它的Controller也全部对字符转义的功能生效，添加的代码如下

public class BaseController {  
 @InitBinder  
 protected void initBinder(WebDataBinder binder) {  
 *// String类型转换，将所有传递进来的String进行HTML编码，防止XSS攻击* binder.registerCustomEditor(String.class, new PropertyEditorSupport() {  
 @Override  
 public void setAsText(String text) {  
 setValue(text == null ? null : StringEscapeUtils.*escapeJavaScript*(text.trim()));  
 }  
 @Override  
 public String getAsText() {  
 Object value = getValue();  
 return value != null ? value.toString() : "";  
 }  
 });  
 }  
}

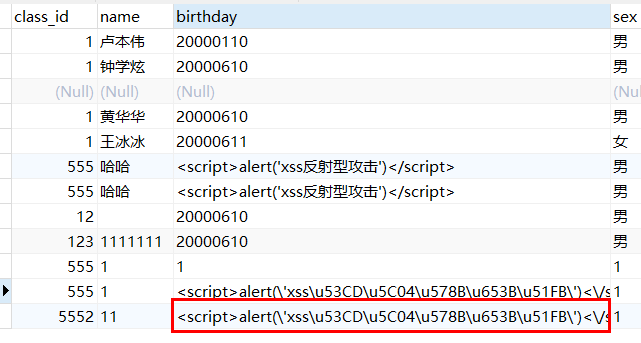
除此之外，这里还要引入Apache旗下的commons-lang包，从而实现字符的转义功能，添加的包如下：

<dependency>  
 <groupId>commons-lang</groupId>  
 <artifactId>commons-lang</artifactId>  
 <version>2.6</version>  
</dependency>

通过对该字符的转义的测试，可以看到字符转义对xss攻击是十分有效的，它直接把代码中的特殊字符转为替代的字符串，从而让浏览器识别为普通的字符串，而不是JavaScript代码，如下图所示：



**图 20 字符转义-测试**



**图 21 字符转义-数据库查看**

* 1. **后端设置HttpOnly**

设置 Cookie 为 HttpOnly 这样可以阻止 javascript访 问 Cookie, 达到保护敏感数据的目的；这里后端既可以在spring中实现，也可以在其运行容器Tomcat中实现HttpOnly的方法，spring和tomcat中的方法实现起来都是较为简单，所以这里就选择其中一个来实现，即选择Tomcat运行容器来实现。

这里需要修改Tomcat运行容器中的数个文件内容，首先修改web.xml文件：

**<session-config>**

**<session-timeout>30</session-timeout>**

**<http-only>true</http-only>**

**</session-config>**

接着修改context.xml文件，在该文件的context标签下加入如下代码：

**useHttpOnly="true"**

最后修改server.xml文件，对connector配置secure属性，代码如下：

**secure="true"**

## 小结

本文介绍了存储型 XSS 的攻击原理、特点和触发方式，并提出 XSS 防御策略，也对这些防御策略进行实现，除外，还对其进行了实际测试。鉴于此，可以看出xss攻击对web站点的重要性，所以作为一个程序员，对xss攻击不应当忽视，而是应当作为头等问题解决。